20th International Workshop on Algorithms in Bioinformatics

WABI 2020, September 7–9, 2020, Pisa, Italy (Virtual Conference)

Edited by

Carl Kingsford
Nadia Pisanti

LIPICS

Editors

Carl Kingsford
Carnegie Mellon University, USA
carlk@cs.cmu.edu

Nadia Pisanti
University of Pisa, Italy
pisanti@di.unipi.it

ACM Classification 2012
Mathematics of computing → Discrete mathematics; Applied computing → Computational biology; Mathematics of computing → Information theory; Theory of computation → Design and analysis of algorithms

ISBN 978-3-95977-161-0

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-161-0.

Publication date
August, 2020

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/LIPIcs.WABI.2020.0

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Regular Paper

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This proceedings volume contains papers presented at the 20th Workshop on Algorithms in Bioinformatics (WABI 2020), which was virtually held in Pisa, Italy, September 7–9, 2020.

The Workshop on Algorithms in Bioinformatics is an annual conference established in 2001 to cover all aspects of algorithmic work in bioinformatics, computational biology, and systems biology. The conference is intended as a forum for presentation of new insights about discrete algorithms and machine-learning methods that address important problems in biology (particularly problems based on molecular data and phenomena); that are founded on sound models; that are computationally efficient; and that have been implemented and tested in simulations and on real datasets. The meeting's focus is on recent research results, including significant work-in-progress, as well as identifying and exploring directions of future research.

This 20th edition of WABI took place in the year of the COVID-19 pandemic emergency. This presented some logistic challenges, but also served to highlight the importance of the development of advanced computational methodologies for understanding biological problems. Over the 20 instances of WABI, computational biology has grown significantly in importance, and now computational analysis methods — some furthered significantly over the years at WABI — are crucial to quickly understanding the evolution and function of the agent of a global health crisis.

WABI 2020 was organized within the ALGO federation of conferences that in 2020 included WABI, ESA (European Symposium on Algorithms), ALGOCLOUD (International Symposium on Algorithmic Aspects of Cloud Computing), ALGOSENSORS (International Symposium on Algorithms and Experiments for Wireless Sensor Networks), ATMOS (International Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems), and WAOA (Workshop on Approximation and Online Algorithms).

Because of the COVID-19 pandemic, the ALGO Organizing Committee decided to run the affiliated conferences online exclusively. On the other hand, the scientific aspects were not affected, except for a somewhat lower number of submissions that fortunately maintained the high quality of WABI standards. The activity of the Programme Committee of WABI took place as usual: peer review, discussion, selection of accepted papers, and publishing of the proceedings have been accomplished in the same way as in all WABI editions.

In 2020, a total of 38 manuscripts were submitted to WABI from which 19 were selected for presentation at the conference and are included in this proceedings volume as full papers. Extended versions of selected papers have been invited for publication in a thematic series in the journal Algorithms for Molecular Biology (AMB), published by BioMed Central. The 19 papers selected for the conference underwent a thorough peer review, involving at least three (and often four or five) independent reviewers per submitted paper, followed by discussions among the WABI Program Committee members. The selected papers cover a wide range of topics including phylogenetic trees and networks, biological network analysis, sequence alignment and assembly, genomic-level evolution, sequence and genome analysis, RNA and protein structure, topological data analysis, and more. They are ordered randomly within this volume.

We thank all the authors of submitted papers and the members of the WABI 2020 Program Committee and their subreviewers for their efforts that made this conference possible. We are also grateful to the WABI Steering Committee for their help and advice. We thank all the conference participants, session chairs, and speakers who contributed to

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

a great scientific program. In particular, we are indebted to the keynote speaker of the conference, Dan Gusfield (University of California Davis), for his presentation, and to the two WABI invited speakers Valentina Boeva (ETH Zurich and Institute Cochin Paris) and Erik Garrison (University of California Santa Cruz). WABI 2020 is grateful for the support of the University of Pisa. We thank ALGO 2020 Organizing Committee for setting up the event in these complicated times due to the pandemic emergency.

Previous proceedings of WABI appeared in LNCS/LNBI volumes 2149 (WABI 2001, Aarhus), 2452 (WABI 2002, Rome), 2812 (WABI 2003, Budapest), 3240 (WABI 2004, Bergen), 3692 (WABI 2005, Mallorca), 4175 (WABI 2006, Zurich), 4645 (WABI 2007, Philadelphia), 5251 (WABI 2008, Karlsruhe), 5724 (WABI 2009, Philadelphia), 6293 (WABI 2010, Liverpool), 6833 (WABI 2011, Saarbrücken), 7534 (WABI 2012, Ljubljana), 8126 (WABI 2013, Sophia Antipolis), 8701 (WABI 2014, Wroclaw), 9289 (WABI 2015, Atlanta), and 9838 (WABI 2016, Aarhus). As of 2016, they appeared in LIPICS volumes 88 (WABI 2017, Boston), 113 (WABI 2018, Helsinki), and 143 (WABI 2019, Boston).

Carl Kingsford & Nadia Pisanti

Program Committee

Carl Kingsford (Co-Chair)
Carnegie Mellon University

Nadia Pisanti (Co-Chair)
University of Pisa, Italy

Tatsuya Akutsu
Kyoto University, Japan

Carl Barton
EBI, UK

Anne Bergeron
Universite du Quebec a Montreal, Canada

Paola Bonizzoni
Università di Milano-Bicocca, Italy

Christina Boucher
University of Florida, USA

Alessandra Carbone
Université Pierre et Marie Curie, France

Rayan Chikhi
Pasteur Institute, France

Lenore Cowen
Tufts University, USA

Gianluca Della Vedova
University of Milano-Bicocca, Italy

Nadia El-Mabrouk
University of Montreal, Canada

Anna Gambin
Warsaw University, Poland

Raffaele Giancarlo
University of Palermo, Italy

Dan Gusfield
UC Davis, USA

Bjarni Halldorsson
deCODE genetics and Reykjavik University, Iceland

Katharina Huber
University of East Anglia, UK

Gunnar Klau
Heinrich Heine University Düsseldorf, Germany

Gregory Kucherov
University of Paris Est, France

Ritu Kundu
National University of Singapore, Singapore

Manuel Lafond
Université de Sherbrooke, Canada

Stefano Lonardi
University of California Riverside, USA

Veli Makinen
University of Helsinki, Finland

Guillaume Marçais
Carnegie Mellon University, USA

Tobias Marschall
Heinrich Heine University, Düsseldorf, Germany

Bernard Moret
Ecole Polytechnique Fédérale de Lausanne, Switzerland

Vincent Moulton
University of East Anglia, UK

Luay Nakhleh
Rice University, USA

William Stafford Noble
University of Washington, USA

Solon Pissis
CWI Amsterdam, the Netherlands

Alberto Policriti
University of Udine, Italy

Mihai Pop
University of Maryland, USA

Teresa Przytycka NCBI
NIH, USA

Sven Rahmann
University of Duisburg-Essen, Germany

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Knut Reinert
FU Berlin, Germany

Eric Rivals
LIRMM – Université de Montpellier, France

Giovanna Rosone
University of Pisa, Italy

Marie-France Sagot
INRIA Rhône-Alpes, France

Alexander Schoenhuth
Bielefeld University, Germany

Jens Stoye
Bielefeld University, Germany

Krister Swenson
CNRS & Université de Montpellier, France

Sharma Thankachan
University of Central Florida, USA

Alexandru Tomescu
University of Helsinki, Finland

Hélène Touzet
CNRS, CRIStAL Lille, France

Esko Ukkonen
University of Helsinki, Finland

Tomas Vinar
Comenius University, Slovakia

Tandy Warnow
University of Illinois at Urbana-Champaign, USA

Prudence W.H. Wong
University of Liverpool, UK

Louxin Zhang
National University of Singapore, Singapore

Michal Ziv-Ukelson
Ben Gurion University of the Negev, Israel

Approximate Search for Known Gene Clusters in New Genomes Using PQ-Trees

Galia R. Zimerman
Ben Gurion University of the Negev, Beer Sheva, Israel
zimgalia@gmail.com

Dina Svetlitsky
Ben Gurion University of the Negev, Beer Sheva, Israel
dina.svetlitsky@gmail.com

Meirav Zehavi
Ben Gurion University of the Negev, Beer Sheva, Israel
meiravze@bgu.ac.il

Michal Ziv-Ukelson[1]
Ben Gurion University of the Negev, Beer Sheva, Israel
michaluz@cs.bgu.ac.il

Abstract

We define a new problem in comparative genomics, denoted PQ-Tree Search, that takes as input a PQ-tree T representing the known gene orders of a gene cluster of interest, a gene-to-gene substitution scoring function h, integer parameters d_T and d_S, and a new genome S. The objective is to identify in S approximate new instances of the gene cluster that could vary from the known gene orders by genome rearrangements that are constrained by T, by gene substitutions that are governed by h, and by gene deletions and insertions that are bounded from above by d_T and d_S, respectively. We prove that the PQ-Tree Search problem is NP-hard and propose a parameterized algorithm that solves the optimization variant of PQ-Tree Search in $O^*(2^\gamma)$ time, where γ is the maximum degree of a node in T and O^* is used to hide factors polynomial in the input size.

The algorithm is implemented as a search tool, denoted PQFinder, and applied to search for instances of chromosomal gene clusters in plasmids, within a dataset of 1,487 prokaryotic genomes. We report on 29 chromosomal gene clusters that are rearranged in plasmids, where the rearrangements are guided by the corresponding PQ-tree. One of these results, coding for a heavy metal efflux pump, is further analysed to exemplify how PQFinder can be harnessed to reveal interesting new structural variants of known gene clusters.

2012 ACM Subject Classification Applied computing → Bioinformatics

Keywords and phrases PQ-Tree, Gene Cluster, Efflux Pump

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.1

Related Version An extended version of the paper is archived in `https://arxiv.org/abs/2007.03589` [45].

Supplementary Material The code for the PQFinder tool as well as all the data needed to reconstruct the results are publicly available on GitHub (`https://github.com/GaliaZim/PQFinder`).

Funding The research of G.R.Z. and M.Z. was partially supported by the Israel Science Foundation (grant no. 1176/18). The research of G.R.Z., D.S. and M.Z.U. was partially supported by the Israel Science Foundation (grant no. 939/18).
Galia R. Zimerman: The research of G.R.Z. was partially supported by the Planning and Budgeting Committee of the Council for Higher Education in Israel and by the Frankel Center for Computer Science at Ben Gurion University.

Acknowledgements Many thanks to Lev Gourevitch for his excellent implementation of a PQ-tree builder. We also thank the anonymous WABI reviewers for their very helpful comments.

[1] M.Z and M.Z.U are joint corresponding authors.

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti; Article No. 1; pp. 1:1–1:24
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Recent advances in pyrosequencing techniques, combined with global efforts to study infectious diseases, yield huge and rapidly-growing databases of microbial genomes [38, 42]. These big new data statistically empower genomic-context based approaches to functional analysis: the biological principle underlying such analysis is that groups of genes that are located close to each other across many genomes often code for proteins that interact with one another, suggesting a common functional association. Thus, if the functional association and annotation of the clustered genes is already known in one (or more) of the genomes, this information can be used to infer functional characterization of homologous genes that are clustered together in another genome.

Groups of genes that are co-locally conserved across many genomes are denoted *gene clusters*. The locations of the group of genes comprising a gene cluster in the distinct genomes are denoted *instances*. Gene clusters in prokaryotic genomes often correspond to (one or several) operons; those are neighbouring genes that constitute a single unit of transcription and translation. However, the order of the genes in the distinct instances of a gene cluster may not be the same.

The discovery (i.e. data-mining) of conserved gene clusters in a given set of genomes is a well studied problem [8, 21, 44]. However, with the rapid sequencing of prokaryotic genomes a new problem is inspired: Namely, given an already known gene cluster that was discovered and studied in one genomic dataset, to identify all the instances of the gene cluster in a given new genomic sequence.

One exemplary application for this problem is the search for chromosomal gene clusters in plasmids. Plasmids are circular genetic elements that are harbored by prokaryotic cells where they replicate independently from the chromosome. They can be transferred horizontally and vertically, and are considered a major driving force in prokaryotic evolution, providing mutation supply and constructing new operons with novel functions [28], for example antibiotic resistance [20]. This motivates biologists to search for chromosomal gene clusters in plasmids, and to study structural variations between the instances of the found gene clusters across the two distinct replicons. However, in addition to the fact that plasmids evolve independently from chromosomes and in a more rapid pace [14], their sequencing, assembly and annotation involves a more noisy process [29].

To accommodate all this, the proposed search approach should be an approximate one, sensitive enough to tolerate some amount of genome rearrangements: transpositions and inversions, missing and intruding genes, and classification of genes with similar function to distinct orthology groups due to sequence divergence or convergent evolution. Yet, for the sake of specificity and search efficiency, we consider confining the allowed variations by two types of biological knowledge: (1) bounding the allowed rearrangement events considered by the search, based on some grammatical model trained specifically from the known gene orders of the gene cluster, and (2) governing the gene-to-gene substitutions considered by the search by combining sequence homology with functional-annotation based semantic similarity.

(1) Bounding the allowed rearrangement events. The PQ-tree [9] is a combinatorial data structure classically used to represent gene clusters [6]. A PQ-tree of a gene cluster describes its hierarchical inner structure and the relations between instances of the cluster succinctly, aids in filtering meaningful from apparently meaningless clusters, and also gives a natural and meaningful way of visualizing complex clusters. A PQ-tree is a rooted tree with three types of nodes: *P-nodes*, *Q-nodes* and leaves. The children of a P-node can appear in any order, while the children of a Q-node must appear in either left-to-right or

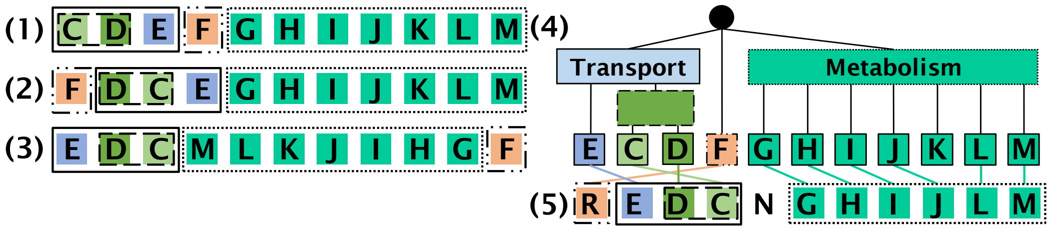

(1) C D E F G H I J K L M
(2) F D C E G H I J K L M
(3) E D C M L K J I H G F
(4)
Transport
Metabolism
E C D F G H I J K L M
(5) R E D C N G H I J L M

Figure 1 A gene cluster containing most of the genes of the *PhnCDEFGHIJKLMNOP* operon [25] and the corresponding PQ-tree. The *Phn* operon encodes proteins that utilize phosphonate as a nutritional source of phosphorus in prokaryotes. The genes *PhnCDE* encode a phosphonate transporter, the genes *PhnGHIJKLM* encode proteins responsible for the conversion of phosphonates to phosphate, and the gene *PhnF* encodes a regulator. **(1)-(3).** The three distinct gene orders found among 47 chromosomal instances of the *Phn* gene cluster. **(4).** A PQ-tree representing the *Phn* gene cluster, constructed from its three known gene orders shown in 1-3. **(5).** An example of a *Phn* gene cluster instance identified by the PQ-tree shown in (4), and the one-to-one mapping between the leaves of the PQ-tree and the genes comprising the instance. The instance genes are rearranged differently from the gene orders shown in 1-3 and yet can be derived from the PQ-tree. In this mapping, gene F is substituted by gene R, gene N is an intruding gene (i.e., deleted from the instance string), and gene K is a missing gene (i.e., deleted from the PQ-tree).

right-to-left order. (In the special case when a node has exactly two children, it does not matter whether it is labeled as a P-node or a Q-node.) Booth and Lueker [9], who introduced this data structure, were interested in representing a set of permutations over a set U, i.e. every member of U appears exactly once as a label of a leaf in the PQ-tree. We, on the other hand, allow each member of U to appear as a label of a leaf in the tree any non-negative number of times. Therefore, we will henceforth use the term *string* rather than *permutation* when describing the gene orders derived from a given PQ-tree.

An example of a PQ-tree is given in Figure 1. It represents a *Phn* gene cluster that encodes proteins that utilize phosphonate as a nutritional source of phosphorus in prokaryotes [25]. The biological assumptions underlying the representation of gene clusters as PQ-trees is that operons evolve via progressive merging of sub-operons, where the most basic units in this recursive operon assembly are colinearly conserved sub-operons [17]. In the case where an operon is assembled from sub-operons that are colinearly dependent, the conserved gene order could correspond, e.g., to the order in which the transcripts of these genes interact in the metabolic pathway in which they are functionally associated [43]. Thus, transposition events shuffling the order of the genes within this sub-operon could reduce its fitness. On the other hand, inversion events, in which the genes participating in this sub-operon remain colinearly ordered are accepted. This case is represented in the PQ-tree by a Q-node (marked with a rectangle). In the case where an operon is assembled from sub-operons that are not colinearly co-dependent, convergent evolution could yield various orders of the assembled components [17]. This case is represented in the PQ-tree by a P-node (marked with a circle). Learning the internal topology properties of a gene cluster from its corresponding gene orders and constructing a query PQ-tree accordingly, could empower the search to confine the allowed rearrangement operations so that colinear dependencies among genes and between sub-operons are preserved.

(2) Governing the gene-to-gene substitutions. A prerequisite for gene cluster discovery is to determine how genes relate to each other across all the genomes in the dataset. In our experiment, genes are represented by their membership in Clusters of Orthologous

Groups (COGs) [37], where the sequence similarity of two genes belonging to the same COG serves as a proxy for homology. Despite low sequence similarity, genes belonging to two different COGs could have a similar function, which would be reflected in the functional description of the respective COGs. Using methods from natural language processing [31], we compute for each pair of functional descriptions a score reflecting their semantic similarity. Combining sequence and functional similarity could increase the sensitivity of the search and promote the discovery of systems with related functions.

Our Contribution and Roadmap. In this paper we define a new problem in comparative genomics, denoted PQ-Tree Search **(in Section 2)**, that takes as input a PQ-tree T (the query) representing the known gene orders of a gene cluster of interest, a gene-to-gene substitution scoring function h, integer parameters d_T and d_S, and a new genome S (the target). The objective is to identify in S a new approximate instance of the gene cluster that could vary from the known gene orders by genome rearrangements that are constrained by T, by gene substitutions that are governed by h, and by gene deletions and insertions that are bounded from above by d_T and d_S, respectively. We prove that PQ-Tree Search is NP-hard **(Theorem 9 in Appendix A)**.

We define an optimization variant of PQ-Tree Search and propose an algorithm **(in Section 3)** that solves it in $O(n\gamma {d_T}^2 {d_S}^2(m_p \cdot 2^\gamma + m_q))$ time, where n is the length of S, m_p and m_q denote the number of P-nodes and Q-nodes in T, respectively, and γ denotes the maximum degree of a node in T. In the same time and space complexities, we can also report all approximate instances of T in S and not only the optimal one.

The algorithm is implemented as a search tool, denoted PQFinder. The code for the tool as well as all the data needed to reconstruct the results are publicly available on GitHub (`https://github.com/GaliaZim/PQFinder`). The tool is applied to search for instances of chromosomal gene clusters in plasmids, within a dataset of 1,487 prokaryotic genomes. In our preliminary results **(given in Section 5)**, we report on 29 chromosomal gene clusters that are rearranged in plasmids, where the rearrangements are guided by the corresponding PQ-tree. One of these results, coding for a heavy metal efflux pump, is further analysed to exemplify how PQFinder can be harnessed to reveal interesting new structural variants of known gene clusters.

Previous Related Works. Permutations on strings representing gene clusters have been studied earlier by [5, 15, 22, 32, 39]. PQ-trees were previously applied in physical mapping [2, 10], as well as to other comparative genomics problems [3, 7, 24].

In Landau et al. [24] an algorithm was proposed for representation and detection of gene clusters in multiple genomes, using PQ-trees: the proposed algorithm computes a PQ-tree of k permutations of length n in $O(kn)$ time, and it is proven that the computed PQ-tree is the one with a minimum number of possible rearrangements of its nodes while still representing all k permutations. In the same paper, the authors also present a general scheme to handle gene multiplicity and missing genes in permutations. For every character that appears a times in each of the k strings, the time complexity for the construction of the PQ-tree, according to the scheme in that paper, is multiplied by an $O((a!)^k)$ factor.

Additional applications of PQ-trees to genomics were studied in [1, 4, 30], where PQ-trees were considered to represent and reconstruct ancestral genomes.

However, as far as we know, searching for approximate instances of a gene cluster that is represented as a PQ-tree, in a given new string, is a new computational problem.

2 Preliminaries

Let Π be an NP-hard problem. In the framework of Parameterized Complexity, each instance of Π is associated with a *parameter* k, and the goal is to confine the combinatorial explosion in the running time of an algorithm for Π to depend only on k. Formally, Π is *fixed-parameter tractable (FPT)* if any instance (I, k) of Π is solvable in time $f(k) \cdot |I|^{\mathcal{O}(1)}$, where f is an arbitrary computable function of k. Nowadays, Parameterized Complexity supplies a rich toolkit to design or refute the existence of FPT algorithms [11, 12, 16].

PQ-Tree: Representing the Pattern. The possible reordering of the children nodes in a PQ-tree may create many equivalent PQ-trees. Booth and Lueker [9] defined two PQ-trees T, T' as *equivalent* (denoted $T \equiv T'$) if one tree can be obtained by legally reordering the nodes of the other; namely, randomly permuting the children of a P-node, and reversing the children of a Q-node. To allow for deletions in the PQ-trees, a generalization of their definition is given in Definition 1 below. Here, *smoothing* is a recursive process in which if by deleting leaves from a tree T, some internal node x of T is left without children, then x is also deleted, but its deletion is not counted (i.e. only leaf deletions are counted).

▶ **Definition 1** (Quasi-Equivalence Between PQ-Trees). *For any two PQ-trees, T and T', the PQ-tree T is* quasi-equivalent to T' with a limit d, *denoted $T \succeq_d T'$, if T' can be obtained from T by (a) randomly permuting the children of some of the P-nodes of T, (b) reversing the children of some of the Q-nodes of T, and (c) deleting up to d leaves from T and applying the corresponding smoothing. (The order of the operations does not matter.)*

Figure S2 shows two equivalent PQ-trees (Figure S2a, Figure S2b) that are each quasi-equivalent with $d = 1$ to the third PQ-tree (Figure S2c). The *frontier* of a PQ-tree T, denoted $F(T)$, is the sequence of labels on the leaves of T read from left to right. For example, the frontier of the PQ-tree in Figure 1 is $ECDFGHIJKLM$. It is interesting to consider the set of frontiers of all the equivalent PQ-trees, defined in [9] as a *consistent set* and denoted by $C(T) = \{F(T') : T \equiv T'\}$. Intuitively, $C(T)$ is the set of all leaf label sequences defined by the PQ-tree structure and obtained by legally reordering its nodes. Here, we generalize the consistent set definition to allow a bounded number of deletions from T, using quasi-equivalence.

▶ **Definition 2** (d-Bounded Quasi-Consistent Set). $C_d(T) = \{F(T') : T \succeq_d T'\}$.

clearly $C_0(T) = C(T)$, and so in a setting where $d = 0$ the latter notation is used. For a node x of a PQ-tree T, the subtree of T rooted in x is denoted by $T(x)$, the set of leaves in $T(x)$ is denoted by $\mathsf{leaves}(x)$, and the *span* of x (denoted $\mathsf{span}(x)$) is defined as $|\mathsf{leaves}(x)|$.

PQ-Tree Search and Related Terminology. An instance of the PQ-Tree Search problem is a tuple (T, S, h, d_T, d_S), where T is a PQ-tree with m leaves, m_p P-nodes, m_q Q-nodes and every leaf x in T has a label $\mathsf{label}(x) \in \Sigma_T$; $S = \sigma_1 \dots \sigma_n \in \Sigma_S^n$ is a string of length n representing the input genome; $d_T \in \mathbb{N}$ specifies the number of allowed deletions from T; $d_S \in \mathbb{N}$ specifies the number of allowed deletions from S; and h is a *boolean substitution function*, describing the possible substitutions between the leaf labels of T and the characters of the given string, S. Formally, h is a function that receives a pair (σ_t, σ_s), where $\sigma_t \in \Sigma_T$ is one of the labels on the leaves of T, and $\sigma_s \in \Sigma_S$ is one of the characters of the given string S, and returns $True$ if σ_t can be replaced with σ_s, and $False$, otherwise. Considering the biological problem at hand, Σ_T and Σ_S are both sets of genes. For $1 \leq i \leq j \leq n$,

$S' = S[i : j] = \sigma_i ... \sigma_j$ is a substring of S beginning at index i and ending at index j. The substring S' is a *prefix* of S if $S' = S[1 : j]$ and it is a *suffix* of S if $S' = S[i : n]$. In addition, we denote σ_i, the i^{th} character of S, by $S[i]$.

The objective of PQ-Tree Search is to find a one-to-one mapping $\mathcal{M}$ between the leaves of T and the characters of a substring S' of S, that comprises a set of pairs each having one of three forms: the substitution form, $(x, \sigma_s(\ell))$, where x is a leaf in T, $\sigma_s \in \Sigma_S$, $h(\mathsf{label}(x), \sigma_s) = True$ and $\ell \in \{1, \dots, n\}$ is the index of the occurrence of σ_s in S that is mapped to the leaf x; the character deletion form, $(\varepsilon, \sigma_s(\ell))$, which marks the deletion of the character $\sigma_s \in \Sigma_S$ at index ℓ of S; the leaf deletion form, (x, ε), which marks the deletion of x, a leaf node of T.

To account for the number of deletions of characters of S' and leaves of T in $\mathcal{M}$, the number of pairs in $\mathcal{M}$ of the form (ε, σ) are marked by $\mathsf{del}_S(\mathcal{M})$ and the number of pairs in $\mathcal{M}$ of the form (x, ε) are marked by $\mathsf{del}_T(\mathcal{M})$. Applying the substitutions defined in $\mathcal{M}$ to S' resulting in the string $S_{\mathcal{M}}$ is the process in which for every $(x, \sigma_s(\ell)) \in \mathcal{M}$, the character σ_s at index ℓ of S is deleted if $x = \varepsilon$, and otherwise substituted by x. This process is demonstrated in Figure S3b. We say that S' is *derived* from T *under* $\mathcal{M}$ with d_T deletions from the tree and d_S deletions from the string, if $d_T = \mathsf{del}_T(\mathcal{M})$, $d_S = \mathsf{del}_S(\mathcal{M})$ and $S_{\mathcal{M}} \in C_{d_T}(T)$. Thus, by definition, there is a PQ-tree T' such that $F(T') = S_{\mathcal{M}}$ and $T \succeq_{d_T} T'$. Note that the deletions of the nodes in T to obtain the nodes in T' are determined by $\mathcal{M}$. The conversion of T to T' as defined by the derivation is illustrated in Figure S3a. The set of permutations and node deletions performed to obtain T' from T together with the substitutions and deletions from S' specified by $\mathcal{M}$ is named the *derivation* μ of T to S'. We also say that $\mathcal{M}$ *yields* the derivation μ.

For a derivation μ of T to $S' = S[s : e]$, we give the following terms and notations (illustrated in Figure S3). The root of T (denoted $root_T$[2]) is *the node that* μ *derives* or *the root of the derivation* and it is denoted by $\mu.v$. For abbreviation, we say that μ *is a derivation of* $\mu.v$. The substring S' is *the string that* μ *derives*. We name s and e the start and end points of the derivation and denote them by $\mu.s$ and $\mu.e$, respectively. The one-to-one mapping that yields μ is denoted by $\mu.o$. The number of deletions from the tree is denoted by $\mu.del_T$. The number of deletions from the string is denoted by $\mu.del_S$. In addition, if x is a leaf node in T and $(x, \sigma_s(\ell)) \in \mu.o$, then x is *mapped to* $S[\ell]$ *under* μ. The character $S[\ell]$ is said to be *deleted under* μ if $(\varepsilon, \sigma_s(\ell)) \in \mu.o$. If $x \in T(\mu.v)$ is a leaf for which $(x, \varepsilon) \in \mu.o$, then x is *deleted under* μ. For an internal node of T, x, if every leaf in $T(x)$ is deleted under μ, then x is *deleted under* μ, and otherwise x is *kept under* μ.

We define two versions of the PQ-Tree Search problem: a decision version (Definition 3) and an optimisation version (Definition 4).

▶ **Definition 3** (Decision PQ-Tree Search). *Given a string S of length n, a PQ-tree T with m leaves, deletion limits $d_T, d_S \in \mathbb{N}$, and a boolean substitution function h between Σ_S and Σ_T, decide if there is a one-to-one mapping $\mathcal{M}$ that yields a derivation of T to a substring S' of S with up to d_T and up to d_S deletions from T and S', respectively.*

To define an optimization version of the PQ-Tree Search problem it is necessary to have a score for every possible substitution between the characters in Σ_T and the characters in Σ_S. Hence, for this problem variant assume that h is a *substitution scoring function*, that is, $h(\sigma_t, \sigma_s)$ for $\sigma_t \in \Sigma_T, \sigma_s \in \Sigma_S$ is the score for substituting σ_s by σ_t in the derivation, and if σ_t cannot be substituted by σ_s, then $h(\sigma_t, \sigma_s) = -\infty$. In addition, we need a cost function, denoted by δ, for the deletion of a character of S and for the deletion of a leaf of T according

[2] We abuse notation and use the term $root_T$ also to refer to the index of the root in T.

to the label of the leaf. The score of a derivation μ, denoted by $\mu.score$, is the sum of scores of all operations (deletions from the tree, deletions from the string and substitutions) in μ. Now, instead of deciding whether there is a one-to-one mapping that yields a derivation of T to a substring of S, we can search for the one-to-one mapping that yields the best derivation (if there exists such a derivation), i.e. a one-to-one mapping for which $\mu.score$ is the highest.

▶ **Definition 4** (Optimization PQ-Tree Search)**.** *Given a string of length n, S, a PQ-tree with m leaves, T, deletion limits $d_T, d_S \in \mathbb{N}$, a substitution scoring function between Σ_S and Σ_T, h, and a deletion cost function, δ, return the one-to-one mapping, $\mathcal{M}$, that yields the highest scoring derivation of T to a substring S' of S with up to d_T deletions from T and up to d_S deletions from S' (if such a mapping exists).*

3 A Parameterized Algorithm

In this section we develop a dynamic programming (DP) algorithm to solve the optimization variant of PQ-Tree Search (Definition 4). Our algorithm receives as input an instance of PQ-Tree Search (T, S, h, d_T, d_S), where h is a substitution scoring function as defined in Section 2. Our default assumption is that deletions are not penalized, and therefore δ is not given as input. The case where deletions are penalized, as well as additional technical details, are omitted due to lack of space, and can be found in [45]. The output of the algorithm is a one-to-one mapping, $\mathcal{M}$, that yields the best (highest scoring) derivation of T to a substring of S with up to d_T deletions from T and up to d_S deletions from the substring, and the score of that derivation. With a minor modification, the output can be extended to include a one-to-one mapping for every substring of S and the derivations that they yield.

Brief Overview. On a high level, our algorithm consists of three components: the main algorithm, and two other algorithms that are used as procedures by the main algorithm. Apart from an initialization phase, the crux of the main algorithm is a loop that traverses the given PQ-tree, T. For each internal node x, it calls one of the two other algorithms: P-mapping (given in Section 3.3) and Q-mapping (deferred to [45], due to space constraints). These algorithms find and return the best derivations from the subtree of T rooted in x, $T(x)$, to substrings of S, based on the type of x (P-node or Q-node). Then, the scores of the derivations are stored in the DP table.

We now give a brief informal description of the main ideas behind our P-mapping and Q-mapping algorithms. Our P-mapping algorithm is inspired by an algorithm described by Bevern et al. [40] to solve the Job Interval Selection problem. Our problem differs from theirs mainly in its control of deletions. Intuitively, in the P-mapping algorithm we consider the task at hand as a packing problem, where every child of x is a set of intervals, each corresponding to a different substring. The objective is to pack non-overlapping intervals such that for every child of x at most one interval is packed. Then, the algorithm greedily selects a child x' of x and decides either to pack one of its intervals (and which one) or to pack none (in which case x' is deleted). Our Q-mapping algorithm is similar to the P-mapping algorithm, but simpler. It can be considered as an interval packing algorithm as well, however, this algorithm packs the children of x in a specific order.

In the following sections, we describe the main algorithm, the P-mapping algorithm, and afterwards analyse the time complexity.

3.1 The Main Algorithm

We now delve into more technical details. The algorithm constructs a 4-dimensional DP table $\mathcal{A}$ of size $m' \times n \times d_T + 1 \times d_S + 1$, where $m' = m + m_p + m_q$ is the number of nodes in T. The purpose of an entry of the DP table, $\mathcal{A}[j, i, k_T, k_S]$, is to hold the highest score of a derivation of the subtree $T(x_j)$ to a substring S' of S starting at index i with k_T deletions from $T(x_j)$ and k_S deletions from S'. If no such derivation exists, $\mathcal{A}[j, i, k_T, k_S] = -\infty$. Addressing $\mathcal{A}$ with some of its indices given as dots, e.g. $\mathcal{A}[j, i, \cdot, \cdot]$, refers to the subtable of $\mathcal{A}$ that is comprised of all entries of $\mathcal{A}$ whose first two indices are j and i. Some entries of the DP table define illegal derivations, namely, derivations for which the number of deletions are inconsistent with the start index, i, the derived node and S. These entries are called *invalid entries* and their value is defined as $-\infty$ throughout the algorithm.

The algorithm first initializes the entries of $\mathcal{A}$ that are meant to hold scores of derivations of the leaves of T to every possible substring of S. Afterwards, all other entries of $\mathcal{A}$ are filled as follows. Go over the internal nodes of T in postorder. For every internal node, x, go in ascending order over every index, i, that can be a start index for the substring of S derived from $T(x)$ (the possible values of i are explained in the next paragraph). For every x and i, use the algorithm for Q-mapping or P-mapping according to the type of x. Both algorithms receive the same input: a substring S' of S, the node x, its children $x_1, \ldots, x_\gamma$, the collection of possible derivations of the children (denoted by $\mathcal{D}$), which have already been computed and stored in $\mathcal{A}$ (as will be explained ahead) and the deletion arguments d_T, d_S. Intuitively, the substring S' is the longest substring of S starting at index i that can be derived from $T(x)$ given d_T and d_S. After being called, both algorithms return a set of derivations of $T(x)$ to a prefix of $S' = S[i : e]$ and their scores. The set holds the highest scoring derivation for every $E(x_j, i, d_T, 0) \leq e \leq E(x_j, i, 0, d_S)$ and for every legal deletion combination $0 \leq k_T \leq d_T$, $0 \leq k_S \leq d_S$.

We now explain the possible values of i and the definition of S' more formally. To this end, note that given the node x and some numbers of deletions k_T and k_S, the length of the derived substring is $L(x, k_T, k_S) \doteq \mathsf{span}(x) - k_T + k_S$. Thus, on the one hand, a substring of maximum length is obtained when there are no deletions from the tree and d_S deletions from the string. Hence, $S' = S[i : E(x, i, 0, d_S)]$ where $E(x, i, k_T, k_S)$ is the function for the calculation of the end point of a derivation, defined as $E(x, i, k_T, k_S) \doteq i - 1 + L(x, k_T, k_S)$. On the other hand, a shortest substring is obtained when there are d_T deletions from the tree and none from the string. Then, the length of the substring is $L(x, d_T, 0) = \mathsf{span}(x) - d_T$. Hence, the index i runs between 1 and $n - (\mathsf{span}(x) - d_T) + 1$.

We now turn to address the aforementioned input collection $\mathcal{D}$ in more detail. Formally, it contains the best scoring derivations of every child x_j of x to every substring of S' with up to d_T and d_S deletions from the tree and string, respectively. It is produced from the entries $\mathcal{A}[j, i', k_T, k_S]$ (where each entry gives one derivation) for all k_T and k_S, and all i' between i and the end index of S', i.e. $i \leq i' \leq E(x_j, i, 0, d_S)$. For the efficiency of the Q-mapping and P-mapping algorithms, the derivations in $\mathcal{D}$ are arranged in descending order with respect to their end point ($\mu.e$). This does not increase the time complexity of the algorithm, as this ordering is received by previous calls to the Q-mapping and P-mapping algorithms.

In the final stage of the main algorithm, when the DP table is full, the score of a best derivation is the maximum of $\{\mathcal{A}[root_T, i, k_T, k_S] : k_T \leq d_T,\ k_S \leq d_S,\ 1 \leq i \leq n - (\mathsf{span}(root_T) - k_T) + 1\}$. We remark that by tracing back through $\mathcal{A}$ the one-to-one mapping that yielded this derivation can be found.

3.2 P-Node and Q-Node Mapping: Terminology

Before describing the P-mapping algorithm, we set up some terminology, which is useful both for the P-mapping algorithm and the Q-mapping algorithm.

We first define the notion of a partial derivation. In the Q-mapping and P-mapping algorithms, the derivation of the input node, x, is built by considering subsets U of its children. With respect to such a subset U, a derivation μ of x is built as if x had only the children in U, and is called a *partial derivation*. Formally, μ is a partial derivation of a node x if $\mu.v = x$ and there is a subset of children $U' \subseteq \mathsf{children}(x)$ such that the two following conditions are true. First, for every $u \in U'$ all the leaves in $T(u)$ are neither mapped nor deleted under μ - that is, there is no mapping pair $(\ell, y) \in \mu.o$ such that $\ell \in \mathsf{leaves}(u)$. Second, for every $v \in \mathsf{children}(x) \setminus U'$ the leaves in $T(v)$ are either mapped or deleted under μ. For every $u \in U'$, we say that u is *ignored under* μ. Notice that any derivation is a partial derivation, where the set of ignored nodes (U' above) is empty. Since all derivations that are computed in a single call to the P-mapping or Q-mapping algorithms have the same start point i, it can be omitted (for brevity) from the end point function: thus, we denote $E_I(x, k_T, k_S) \doteq L(x, k_T, k_S)$. Then, for a set U of nodes, we define $L(U, k_T, k_S) \doteq \sum_{x \in U} \mathsf{span}(x) + k_S - k_T$ and accordingly $E_I(U, k_T, k_S) \doteq L(U, k_T, k_S)$.

We now define certain collections of derivations with common properties (such as having the same numbers of deletions and end point).

▶ **Definition 5.** *The collection of all the derivations of every node $u \in U$ to suffixes of $S'[1 : E_I(U, k_T, k_S)]$ with exactly k_T deletions from the tree and exactly k_S deletions from the string is denoted by $\mathcal{D}(U, k_T, k_S)$.*

▶ **Definition 6.** *The collection of all the best derivations from the nodes in U to suffixes of $S'[1 : E_I(U, k_T, k_S)]$ with up to k_T deletions from the tree and up to k_S deletions from the string is denoted by $\mathcal{D}_{\le}(U, k_T, k_S)$. Specifically, for every node $u \in U$, $k'_T \le k_T$ and $k'_S \le k_S$, the set $\mathcal{D}_{\le}(U, k_T, k_S)$ holds only one highest scoring derivation of u to a suffix of $S'[1 : E_I(U, k_T, k_S)]$ with k'_T and k'_S deletions from the tree and string, respectively.*[3]

It is important to distinguish between these two definitions. First, the derivations in $\mathcal{D}(U, k_T, k_S)$ have *exactly* k_T and k_S deletions, while the derivations in $\mathcal{D}_{\le}(U, k_T, k_S)$ have *up to* k_T and k_S deletions. Second, in $\mathcal{D}(U, k_T, k_S)$ there can be several derivations that differ only in their score and in the one-to-one mapping that yields them, while in $\mathcal{D}_{\le}(U, k_T, k_S)$, there is only one derivation for every node $u \in U$ and deletion combination pair (k'_T, k'_S). Note that the end points of all of the derivations are equal.

Definition 5 is used for describing the content of an entry of the DP table, where the focus is on the collection of all the derivations of x to S' with exactly k_T and k_S deletions, $\mathcal{D}(\{x\}, k_T, k_S)$. For simplicity, the abbreviation $\mathcal{D}(u, k_T, k_S) = \mathcal{D}(\{u\}, k_T, k_S)$ is used. In every step of the P-mapping and Q-mapping algorithms, a different set of derivations of the children of x is examined, thus, Definition 6 is used for $U \subseteq \mathsf{children}(x)$. In addition, the set of derivations $\mathcal{D}$ that is received as input to the algorithms can be described using Definition 6 as can be seen in Equation (1) below. In this equation, the union is over all $U \subseteq \mathsf{children}(x)$ because in this way the derivations of all the children of x with *every possible*

[3] $\mathcal{D}_{\le}(U, k_T, k_S)$ can be defined using Definition 5: $\mathcal{D}_{\le}(U, k_T, k_S) = \bigcup_{u \in U} \bigcup_{k'_T \le k_T} \bigcup_{k'_S \le k_S} \max_{\substack{\mu \in \mathcal{D}(U, k_T, k_S) \\ \text{s.t.} \\ \mu.del_T = k'_T \\ \mu.del_S = k'_S \\ \mu.v = u}} \mu.score.$

end point are obtained (in contrast to having only $U = \mathsf{children}(x)$, which results in the derivations of all the children of x with the end point $E_I(\mathsf{children}(x), k_T, k_S)$).

$$\mathcal{D} = \bigcup_{U \subseteq \mathsf{children}(x)} \bigcup_{k_T \leq d_T} \bigcup_{k_S \leq d_S} \mathcal{D}_{\leq}(U, k_T, k_S) \tag{1}$$

In the P-mapping algorithm for $C \subseteq \mathsf{children}(x)$, the notation $x^{(C)}$ is used to indicate that the node x is considered as if its only children are the nodes in C. Consequentially, the span of $x^{(C)}$ is defined as $\mathsf{span}(x^{(C)}) \doteq \sum_{c \in C} \mathsf{span}(c)$, and the set $\mathcal{D}(x^{(C)}, k_T, k_S)$ (in Definition 5 where $U = \{x^{(C)}\}$) now refers to a set of *partial* derivations.

3.3 P-Node Mapping: The Algorithm

Recall that the input consists of an internal P-node x, a string S', limits on the number of deletions from the tree T and the string S', d_T and d_S, respectively, and a set of derivations $\mathcal{D}$ (see Equation (1)). The output is $\bigcup_{k_T \leq d_T} \bigcup_{k_S \leq d_S} \arg\max_{\mu \in \mathcal{D}(x, k_T, k_S)} \mu.score$, which is the collection of the best scoring derivations of x to every possible prefix of S' having up to d_T and d_S deletions from the tree and string, respectively. Thus, there are $O(d_T d_S)$ derivations in the output.

The algorithm constructs a 3-dimensional DP table $\mathcal{P}$, which has an entry for every $0 \leq k_T \leq d_T$, $0 \leq k_S \leq d_S$ and subset $C \subseteq \mathsf{children}(x)$. The purpose of an entry $\mathcal{P}[C, k_T, k_S]$ is to hold the best score of a partial derivation in $\mathcal{D}(x^{(C)}, k_T, k_S)$, i.e. a partial derivation rooted in $x^{(C)}$ to a prefix of S' with exactly k_T deletions from the tree and k_S deletions from the string. The children of x that are not in C are *ignored* (as defined in Section 3.2) under the partial derivation stored by the DP table entry $\mathcal{P}[C, k_T, k_S]$, thus they are neither deleted nor counted in the number of deletions from the tree, k_T. (They will be accounted for in the computation of other entries of $\mathcal{P}$.) Similarly to the main algorithm, some of the entries of $\mathcal{P}$ are invalid, and their value is defined as $-\infty$. Every entry $\mathcal{P}[C, k_T, k_S]$ for which $L(C, k_T, k_S) = 0$ and $k_S = 0$ or for which $C = \emptyset$ and $k_T = 0$ is initialized with 0.

After the initialization, the remaining entries of $\mathcal{P}$ are calculated using the recursion rule in Equation (2) below. The order of computation is ascending with respect to the size of the subsets C of the children of x, and for a given $C \subseteq \mathsf{children}(x)$, the order is ascending with respect to the number of deletions from both tree and string.

$$\mathcal{P}[C, k_T, k_S] = \max \begin{cases} \mathcal{P}[C, k_T, k_S - 1] \\ \max\limits_{\mu \in \mathcal{D}_{\leq}(C, k_T, k_S)} \mathcal{P}[C \setminus \{\mu.v\}, k_T - \mu.del_T, k_S - \mu.del_S] + \mu.score \end{cases} \tag{2}$$

Intuitively, every entry $\mathcal{P}[C, k_T, k_S]$ defines some index e' of S' that is the end point of every partial derivation in $\mathcal{D}(x^{(C)}, k_T, k_S)$. Thus, $S'[e']$ must be a part of any partial derivation $\mu \in \mathcal{D}(x^{(C)}, k_T, k_S)$, so, either $S'[e']$ is deleted under μ or it is mapped under μ. The former option is captured by the first case of the recursion rule. If $S'[e']$ is mapped under μ, then due to the hierarchical structure of $T(x)$, it must be mapped under some derivation μ' of one of the children of x that are in C. Thus we receive the second case of the recursion rule. We remark that the case of a node deletion is captured by the initialization.

Once the entire DP table is filled, a derivation of maximum score for every end point and deletion numbers combination can be found in $\mathcal{P}[\mathsf{children}(x), \cdot, \cdot]$. Traversing the said subtable in a specific order guarantees the output derivations are ordered with respect to their end point without further calculations.

3.4 Complexity Analysis of the Main Algorithm

In this section we compare the time complexity of the main algorithm (in Section 3.1) to the naïve solution for PQ-Tree Search. Lemma 8 ahead (proven in Appendix B) gives the time complexity of the main algorithm, and thus it is proven that PQ-Tree Search has an FPT solution with the parameter γ (Theorem 7).

▶ **Theorem 7.** *PQ-Tree Search with parameter γ is FPT. Particularly, it has an FPT algorithm that runs in $O^*(2^\gamma)$ time*[4]*.*

▶ **Lemma 8.** *The algorithm in Section 3.1 runs in $O(n\gamma {d_T}^2 {d_S}^2(m_p 2^\gamma + m_q))$ time and $O(d_T d_S(mn + 2^\gamma))$ space, where γ is the maximum degree of a node in T.*

The naïve solution for PQ-Tree Search solves it in $O(2^{m_q}(\gamma!)^{m_p} nm(d_T + d_S)d_T d_S)$ time. Therefore, we conclude that the time complexity of our algorithm is substantially better, as exemplified by considering two complementary cases. One, when there are only P-nodes in T (i.e.$m_p = O(m)$), the naïve algorithm is super-exponential in γ, and even worse, exponential in m, while ours is exponential only in γ, and hence polynomial for any γ that is constant (or even logarithmic in the input size). Second, when there are only Q-nodes in T (i.e. $m_q = O(m)$), the naïve algorithm is exponential while ours is polynomial.

4 Methods and Datasets

Dataset and Gene Cluster Generation. 1,487 fully sequenced prokaryotic strains with COG ID annotations were downloaded from GenBank (NCBI; ver 10/2012). Among these strains, 471 genomes included a total of 933 plasmids.

The gene clusters were generated using the tool CSBFinder-S [36]. CSBFinder-S was applied to all the genomes in the dataset after removing their plasmids, using parameters $q = 1$ (a colinear gene cluster is required to appear in at least one genome) and $k = 0$ (no insertions are allowed in a colinear gene cluster), resulting in 595,708 colinear gene clusters. Next, ignoring strand and gene order information, colinear gene clusters that contain the exact same COGs were united to form the generalized set of gene clusters. The resulting gene clusters were then filtered to 26,270 gene clusters that appear in more than 30 genomes.

Generation of PQ-Trees. The generation of PQ-trees was performed using a program [19] that implements the algorithm described in [24] for the construction of a PQ-tree from a list of strings comprised from the same set of characters. In the case where a character appeared more than once in a training string, the PQ-tree with a minimum sized consistent set was chosen. The generated PQ-trees varied in size and complexity. The length of their frontier ranged between 4 and 31, and the size of their consistent set ranged between 4 and 362,880.

Implementation and Performance. PQFinder is implemented in Java 1.8. The runs were performed on an Intel Xeon X5680 machine with 192 GB RAM. The time it took to run all plasmid genomes against one PQ-tree ranged between 5.85 seconds (for a PQ-tree with a consistent set of size 4) and 181.5 seconds (for a PQ-tree with a consistent set of size 362,880). In total it took an hour and 47 minutes to run every one of the 779 PQ-trees against every one of the 933 plasmids.

[4] The notation O* is used to hide factors polynomial in the input size.

Substitution Scoring Function. The substitution scoring function reflects the distance between each pair of COGs, that is computed based on sentences describing the functional annotation of the COGs (e.g., "ABC-type sugar transport system, ATPase component"). The "Bag of Words model" was employed, where the functional description of each COG is represented by a sparse vector that is normalized to have a unit Euclidean norm. First, each COG description was tokenized and the occurrences of tokens in each description was counted and normalized using tf–idf term weighting. Then, the cosine similarity between each two vectors was computed, resulting in similarity scores ranging between 0 and 1. The sentences describing COGs are short, therefore each word largely influences the score, even after the tf–idf term weighting. Therefore, words that do not describe protein functions that were found in the top 30 most common words in the description of all COGs were used as stop-words. Two COGs with the same COG IDs were set to have a score of 1.1, and the substitution score between a gene with no COG annotation to any other COG was set to be -0.1. Two COGs with a zero score were penalized to have a score of -0.2 and the deletion of a COG from the query or the target string was set to have a score of zero.

Enrichment Analysis. For each of the four variants in Figure 2.C, a hypergeometric test was performed to measure the enrichment of the corresponding variant in one of the classes in which it appears. A total of 10 p-values were computed and adjusted using the Bonferroni correction; two p-values were found significant (<0.05), reported in Section 5.

Specificity Score. We define a specificity score for a PQ-tree T of a gene cluster named S-score. Let $\tilde{T}$ be the least specific PQ-tree that could have been generated for the genes of the gene cluster based on which T was constructed. Namely, a PQ-tree that allows all permutations of said genes, has height 1 and is rooted in a P-node whose children (being the leaves of the tree) are the leaves of T. Thus, the S-score of T is $\frac{|C(\tilde{T})|}{|C(T)|}$. For a gene cluster of permutations (i.e. there are no duplications), the computation of $|C(T)|$ is as described in Equation (3), where the set of P-nodes in T is denoted by $T.p$.

$$|C(T)| = 2^{m_q} \cdot \prod_{x \in T.p} |\mathsf{children}(x)|! \tag{3}$$

For a gene cluster that has duplications, the set $C(T)$ is generated to learn its size. Let $\mathsf{a}(\ell, T)$ denote the number of appearances of the label ℓ in the leaves of T and let $\mathsf{labels}(T)$ denote the set of all labels of the leaves of T. So, the formula for $|C(\tilde{T})|$ is as in Equation (4). Clearly, for T with no duplications $|C(\tilde{T})| = |F(T)|!$.

$$|C(\tilde{T})| = \frac{|F(T)|!}{\prod_{\ell \in \mathsf{labels}(T)} \mathsf{a}(\ell, T)!} \tag{4}$$

5 Results

5.1 Chromosomal Gene Orders Rearranged in Plasmids

The labeling of each internal node of a PQ-tree as P or Q, is learned during the construction of the tree, based on some interrogation of the gene orders from which the PQ-tree is trained [24]. As a result, the set of strings that can be derived from a PQ-tree T, consists of two parts: (1) all the strings representing the known gene orders from which T was constructed, and (2) additional strings, denoted *tree-guided rearrangements*, that do not appear in the set of gene orders constructing T, but can be obtained via rearrangement operations that are constrained by T. Thus, the tree-guided rearrangements conserve the internal topology properties of the gene cluster, as learned from the corresponding gene orders during the construction of T, such that colinear dependencies among genes and between sub-operons are preserved in the inferred gene orders.

In this section, we used the PQ-trees constructed from chromosomal gene clusters, to examine whether tree-guided rearrangements can be found in plasmids. The objective was to discover gene orders in plasmids that abide by a PQ-tree representing a chromosomal gene cluster, and differ from all the gene orders participating in the PQ-tree's construction. PQ-trees that are constructed from gene clusters that have only one gene order or gene clusters with less than four COGs cannot generate gene orders that differ from the ones participating in their construction. Therefore, only 779 out of 26,270 chromosomal gene clusters were used for the construction of query PQ-trees (the generation of the chromosomal gene clusters is detailed in Section 4). Using our tool PQFinder that implements the algorithm proposed for solving the PQ-Tree Search problem, the query PQ-trees were run against all plasmid genomes. This benchmark was run conservatively without allowing substitutions or deletions from the PQ-tree or from the target string. 380 of the query gene clusters were found in at least one plasmid. The instances of these gene clusters in plasmids are provided in the Supplementary Materials as a session file that can be viewed using the tool CSBFinder-S [36].

Tree-guided rearrangements were found among instances of 29 gene clusters. The PQ-trees corresponding to these gene clusters were sorted by a decreasing S-score, where higher scores are given to a more specific tree (details in Section 4). In this setting, the higher the S-score, the smaller the number of possible gene orders that can be derived from the respective PQ-tree. Interestingly, 21 out of these 29 gene clusters code for transporters, namely 20 importers (ABC-type transport systems) and one exporter (efflux pump). The 10 top ranking results are presented in Table 1.

We selected the third top-ranking PQ-tree in Table 1 for further analysis. This PQ-tree was constructed from seven gene orders of a gene cluster that encodes a heavy metal efflux pump. This gene cluster was found in the chromosomes of 79 genomes (represented by the seven distinct gene orders mentioned above) and in the plasmids of seven genomes. The tree-guided rearrangement instance was found in the strain *Cupriavidus metallidurans CH34*, isolated from an environment polluted with high concentrations of several heavy metals. This strain contains two large plasmids that confer resistance to a large number of heavy metals such as zinc, cadmium, copper, cobalt, lead, mercury, nickel and chromium. We hypothesize that the rearrangement event could have been caused by a heavy metal stress [41]. In the following section we will focus on this PQ-tree to further study its different variants in plasmids.

5.2 RND Efflux Pumps in Plasmids

The heavy metal efflux pump examined in the previous section (corresponding to the third top-ranking PQ-tree in Table 1), was used as a PQFinder query and re-run against all the plasmids in our dataset in order to discover approximate instances of this gene cluster, possibly encoding remotely related variations of the efflux pump it encodes. This time, in order to increase sensitivity, a semantic substitution scoring function (described in Section 4) was used, and the parameters were set to $d_T = 1$ (up to one deletion from the tree, representing missing genes) and $d_S = 3$ (up to three deletions from the plasmid, representing intruding genes). An instance of a gene cluster is accepted if it was derived from the corresponding PQ-tree with a score that is higher than 0.75 of the highest possible score attainable by the query. The plasmid instances detected by PQFinder are displayed in Figure S4.

Table 1 Ten top ranked PQ-trees for which tree-guided rearrangements were found in plasmids. [1]Square brackets represent a Q-node; round brackets represent a P-node. Numbers indicate the respective COG IDs. [2]This column indicates the number of genomes harboring plasmid instances of the respective PQ-tree. The number in brackets indicates the number of genomes harboring a tree-guided gene rearrangement of the corresponding gene cluster. The full table can be found in [45].

	PQ-Tree[1]	S-score	# Genomes[2]	Functional Category
1	[[0683 [[0411 0410] [0559 4177]]] 0583]	22.5	5 (2)	Amino acid transport
2	(1609 [1653 1175 0395] 3839)	10.0	10 (2)	Carbohydrate transport
3	[[1538 [3696 0845]] [0642 0745]]	7.5	7 (1)	Heavy metal efflux
4	[[2115 1070] [4213 [1129 4214]]]	7.5	1 (1)	Carbohydrate transport
5	[1960 [[2011 1135] [2141 1464]]]	7.5	3 (1)	Amino acid transport
6	[[0596 0599] [[3485 3485] 0015]]	7.5	9 (1)	Metabolism
7	[[[1129 1172 1172] 1879] 3254]	7.5	6 (1)	Carbohydrate transport
8	(1609 1869 [[1129 1172] 1879] 0524)	7.5	1 (1)	Carbohydrate transport
9	(0683 [0559 4177] [0411 0410] 0318)	7.5	1 (1)	Amino acid transport
10	(3839 0673 [[0395 1175] 1653])	5.0	10 (1)	Carbohydrate transport

Heavy metal efflux pumps are involved in the resistance of bacteria to a wide range of toxic metal ions [27] and they belong to the resistance-nodulation-cell division (RND) family. In Gram-negative bacteria, RND pumps exist in a tripartite form, comprised from an outer-membrane protein (OMP), an inner membrane protein (IMP), and a periplasmic membrane fusion protein (MFP) that connects the other two proteins. In some cases, the genes of the RND pump are flanked with two regulatory genes that encode the factors of a two-component regulatory system comprising a sensor/histidine kinase (HK) and response regulator (RR) (Figure 2.B). This regulatory system responds to the presence of a substrate, and consequently enhances the expression of the efflux pump genes.

The PQ-tree of this gene cluster (Figure 2.A) shows that the COGs encoding the IMP and MFP proteins always appear as an adjacent pair, the OMP COG is always adjacent to this IMP-MFP pair, and the HK and RR COGs appear as a pair downstream or upstream to the other COGs. COG3696, which encodes the IMP protein, is annotated as a heavy metal efflux pump protein, while the other COGs are common to all RND efflux pumps. Therefore, it is very likely that the respective gene cluster corresponds to a heavy metal RND pump. The absence of an additional periplasmic protein likely indicates that this gene cluster encodes a Czc-like efflux pump that exports divalent metals such as the cobalt, zinc and cadmium exporter in *Cupriavidus metallidurans* [27] (Figure 2.C(1)).

PQFinder discovered instances of this gene cluster in the plasmids of 12 genomes (Figures 2.C(1) and 2.D), and it is significantly enriched in the β-proteobacteria class (hypergeometric p-value= 1.09×10^{-5}, Bonferroni corrected p-value $= 1.09 \times 10^{-4}$). In addition, three other variants of RND pumps were found as instances of the query gene cluster (Figure 2.C(2-4)). The plasmids of three genomes contained instances that were missing the COG corresponding to the OMP gene CzcC (Figure 2.C(2)). This could be caused by a low quality sequencing or assembly of these plasmids. An alternative possible explanation is that a Czc-like efflux pump can still be functional without CzcC; a previous study showed that the deletion of CzcC resulted in the loss of cadmium and cobalt resistance, but most of the zinc resistance was retained [27].

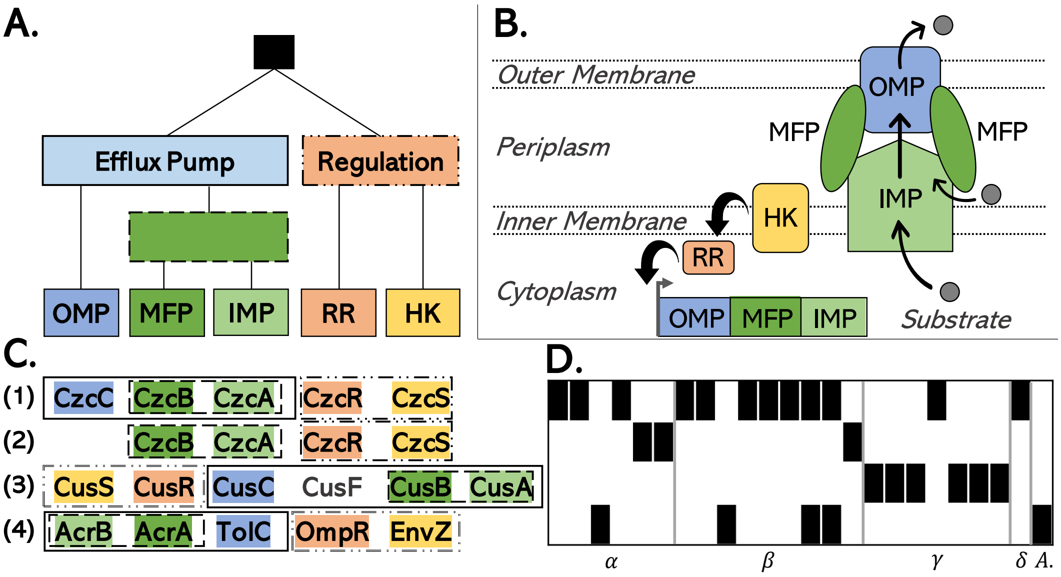


Figure 2 **A.** A PQ-tree of a heavy metal RND efflux pump, corresponding to the third top scoring result in Table 1. **B.** An illustration of an RND efflux pump consisting of an outer-membrane protein (OMP), an inner membrane protein (IMP), and a periplasmic membrane fusion protein (MFP) that connects the other two proteins. In addition, a two-component regulatory system consisting of a sensor/histidine kinase (HK) and response regulator (RR) enhances the transcription of the efflux pump genes. **C.** Representatives of the three different RND efflux pumps found in plasmids. **(1)** A Czc-like heavy metal efflux pump, **(2)** A Czc-like heavy metal efflux pump with a missing OMP gene, **(3)** A Cus-like heavy metal efflux pump, **(4)** An Acr-like multidrug efflux pump. Additional details can be found in the text. **D.** The presence-absence map of the three types of efflux pumps found in the plasmids of different genomes. The rows correspond to the rows in (C), the columns correspond to the genomes in which instances were found, organized according to their taxonomic classes. A black cell indicates that the corresponding efflux pump is present in the plasmids of the genome. The labels below the map indicate the classes $\alpha, \beta, \gamma, \delta$-Proteobacteria and Acidobacteriia.

Some instances identified by the query, found in the plasmids of six genomes, seem to encode a different heavy metal efflux pump (Figure 2.C(3)). This variant includes all COGs from the query, in addition to an intruding COG that encodes a periplasmic protein (CusF). This protein is a predicted copper usher that facilitates access of periplasmic copper towards the heavy metal efflux pump. Indeed, the genomic region of Cus-like efflux pumps that export monovalent metals, such as the silver and copper exporter in *Escherichia coli*, include this periplasmic protein, in contrast to the Czc-like efflux pump [27]. This variant was found in the plasmids of six bacterial genomes belonging to the class γ-proteobacteria (Figure 2.D). This gene cluster is significantly enriched in the γ-proteobacteria class (hypergeometric p-value= 2.13×10^{-4}, Bonferroni corrected p-value $= 2.13 \times 10^{-3}$). Surprisingly, all of these strains, except for one, are annotated as human or animal pathogens. Interestingly, previous studies suggest that the host immune system exploits excess copper to poison invading pathogens [18], which can explain why these pathogens evolved copper efflux pumps.

Another variant of the pump, appearing in five genomes (Figures 2.C(4) and 2.D), resulted from a substitution of the query IMP gene (COG3696) by a different IMP gene (COG0841) belonging to the multidrug efflux pump AcrAB-TolC. The AcrAB-TolC system, mainly studied in *Escherichia coli*, transports a diverse array of compounds with little chemical similarity [13]. AcrAB-TolC is an example of an intrinsic non-specific efflux pump, which is widespread in the chromosomes of Gram-negative bacteria, and likely evolved as a general response to environmental toxins [35]. In this case, the query gene cluster and the identified variant share all COGs, except for the COGs encoding the IMP genes. The differing COGs are responsible for substrate recognition, which naturally differs between the two pumps, as one pump exports heavy metal while the other exports multiple drugs. When considering the functional annotation of these two COGs, we see that the query metal efflux pump COG encoding the IMP gene is annotated as “Cu/Ag efflux pump CusA”, while in the multidrug efflux pump the COG encoding the IMP gene is annotated as “Multidrug efflux pump subunit AcrB”. Thus, in spite of the difference in substrate specificity, the semantic similarity measure employed by PQFinder was able to reflect their functional similarity and allowed the substitution between them, while conferring to the structure of the PQ-tree.

6 Conclusions

In this paper, we defined a new problem in comparative genomics, denoted PQ-Tree Search. The objective of PQ-Tree Search is to identify approximate new instances of a gene cluster in a new genome S. In our model, the gene cluster is represented by a PQ-tree T, and the approximate instances can vary from the known gene orders by genome rearrangements that are constrained by T, by gene substitutions that are governed by a gene-to-gene substitution scoring function h, and by gene deletions and insertions that are bounded from above by integer parameters d_T and d_S, respectively.

We proved that the PQ-Tree Search problem is NP-hard and proposed a parameterized algorithm that solves it in $O^*(2^\gamma)$ time, where γ is the maximum degree of a node in T and O^* is used to hide factors polynomial in the input size.

The proposed algorithm was implemented as a publicly available tool and harnessed to search for tree-guided rearrangements of chromosomal gene clusters in plasmids. We identified 29 chromosomal gene clusters that are rearranged in plasmids, where the rearrangements are guided by the corresponding PQ-tree. A tree-guided rearrangement event of one of these gene clusters, coding for a heavy metal efflux pump, was detected in a bacterial strain that was isolated from an environment polluted with several heavy metals. Thus, a future extension of this study could explore whether similar gene cluster rearrangement events are correlated with environmental stress or other bacterial adaptations.

The said gene cluster was further analysed to characterize its approximate instances in plasmids. An interesting variant of the analysed gene cluster, found among its approximate instances, corresponds to a copper efflux pump. It was found mainly in pathogenic bacteria, and likely constitutes a bacterial defense mechanism against the host immune response. These results exemplify how our proposed tool PQFinder can be harnessed to find meaningful variations of known biological systems that are conserved as gene clusters, and to explore their function and evolution.

One of the downsides to using PQ-trees to represent gene clusters is that very rare gene orders taken into account in the tree construction could greatly increase the number of allowed rearrangements and thus substantially lower the specificity of the PQ-tree. Thus, a natural continuation of our research would be to increase the specificity of the model by considering a stochastic variation of PQ-Tree Search. Namely, defining a PQ-tree in which the internal nodes hold the probability of each rearrangement, and adjusting the algorithm for PQ-Tree Search accordingly. In addition, future extensions of this work could also aim to increase the sensitivity of the model by taking into account gene duplications, gene-merge and gene-split events, which are typical events in gene cluster evolution.

References

1 Zaky Adam, Monique Turmel, Claude Lemieux, and David Sankoff. Common intervals and symmetric difference in a model-free phylogenomics, with an application to streptophyte evolution. *Journal of Computational Biology*, 14(4):436–445, 2007. doi:10.1089/cmb.2007.A005.

2 Farid Alizadeh, Richard M Karp, Deborah K Weisser, and Geoffrey Zweig. Physical mapping of chromosomes using unique probes. *Journal of Computational Biology*, 2(2):159–184, 1995. doi:10.1089/cmb.1995.2.159.

3 Severine Bérard, Anne Bergeron, Cedric Chauve, and Christophe Paul. Perfect sorting by reversals is not always difficult. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(1):4–16, 2007. doi:10.1145/1229968.1229972.

4 Anne Bergeron, Mathieu Blanchette, Annie Chateau, and Cedric Chauve. Reconstructing ancestral gene orders using conserved intervals. In *International Workshop on Algorithms in Bioinformatics*, pages 14–25. Springer, 2004. doi:10.1007/978-3-540-30219-3_2.

5 Anne Bergeron, Sylvie Corteel, and Mathieu Raffinot. The algorithmic of gene teams. In *International Workshop on Algorithms in Bioinformatics*, pages 464–476. Springer, 2002. doi:10.1007/3-540-45784-4_36.

6 Anne Bergeron, Yannick Gingras, and Cedric Chauve. Formal models of gene clusters. *Bioinformatics Algorithms: Techniques and Applications*, 8:177–202, 2008. doi:10.1002/9780470253441.ch8.

7 Anne Bergeron, Julia Mixtacki, and Jens Stoye. Reversal distance without hurdles and fortresses. In *Annual Symposium on Combinatorial Pattern Matching*, pages 388–399. Springer, 2004. doi:10.1007/978-3-540-27801-6_29.

8 Sebastian Böcker, Katharina Jahn, Julia Mixtacki, and Jens Stoye. Computation of median gene clusters. *Journal of Computational Biology*, 16(8):1085–1099, 2009. doi:10.1089/cmb.2009.0098.

9 Kellogg S Booth and George S Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.

10 Thomas Christof, Michael Jünger, John Kececioglu, Petra Mutzel, and Gerhard Reinelt. A branch-and-cut approach to physical mapping of chromosomes by unique end-probes. *Journal of Computational Biology*, 4(4):433–447, 1997. doi:10.1089/cmb.1997.4.433.

11 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

12 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

13 Dijun Du, Zhao Wang, Nathan R James, Jarrod E Voss, Ewa Klimont, Thelma Ohene-Agyei, Henrietta Venter, Wah Chiu, and Ben F Luisi. Structure of the AcrAB–TolC multidrug efflux pump. *Nature*, 509(7501):512–515, 2014. doi:10.1038/nature13205.

14 William G Eberhard. Evolution in bacterial plasmids and levels of selection. *The Quarterly Review of Biology*, 65(1):3–22, 1990. doi:10.1086/416582.

15 Revital Eres, Gad M Landau, and Laxmi Parida. A combinatorial approach to automatic discovery of cluster-patterns. In *International Workshop on Algorithms in Bioinformatics*, pages 139–150. Springer, 2003. doi:10.1007/978-3-540-39763-2_11.

16 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019.

17 Marco Fondi, Giovanni Emiliani, and Renato Fani. Origin and evolution of operons and metabolic pathways. *Research in Microbiology*, 160(7):502–512, 2009. doi:10.1016/j.resmic.2009.05.001.

18 Yue Fu, Feng-Ming James Chang, and David P Giedroc. Copper transport and trafficking at the host–bacterial pathogen interface. *Accounts of Chemical Research*, 47(12):3605–3613, 2014. doi:10.1021/ar500300n.

19 Lev Gourevitch. A program for pq-tree construction. https://github.com/levgou/pqtrees.

20 Susu He, Michael Chandler, Alessandro M Varani, Alison B Hickman, John P Dekker, and Fred Dyda. Mechanisms of evolution in high-consequence drug resistance plasmids. *mBio*, 7(6):e01987–16, 2016. doi:10.1128/mBio.01987-16.

21 Xin He and Michael H Goldwasser. Identifying conserved gene clusters in the presence of homology families. *Journal of Computational Biology*, 12(6):638–656, 2005. doi:10.1089/cmb.2005.12.638.

22 Steffen Heber and Jens Stoye. Algorithms for finding gene clusters. In *International Workshop on Algorithms in Bioinformatics*, pages 252–263. Springer, 2001. doi:10.1007/3-540-44696-6_20.

23 J Mark Keil. On the complexity of scheduling tasks with discrete starting times. *Operations Research Letters*, 12(5):293–295, 1992. doi:10.1016/0167-6377(92)90087-J.

24 Gad M Landau, Laxmi Parida, and Oren Weimann. Gene proximity analysis across whole genomes via pq trees. *Journal of Computational Biology*, 12(10):1289–1306, 2005. doi:10.1089/cmb.2005.12.1289.

25 William W Metcalf and Barry L Wanner. Evidence for a fourteen-gene, phnC to phnP locus for phosphonate metabolism in escherichia coli. *Gene*, 129(1):27–32, 1993. doi:10.1016/0378-1119(93)90692-V.

26 Kazuo Nakajima and S Louis Hakimi. Complexity results for scheduling tasks with discrete starting times. *Journal of Algorithms*, 3(4):344–361, 1982. doi:10.1016/0196-6774(82)90030-X.

27 Dietrich H Nies. Efflux-mediated heavy metal resistance in prokaryotes. *FEMS Microbiology Reviews*, 27(2-3):313–339, 2003. doi:10.1016/S0168-6445(03)00048-2.

28 Vic Norris and Annabelle Merieau. Plasmids as scribbling pads for operon formation and propagation. *Research in Microbiology*, 164(7):779–787, 2013. doi:10.1016/j.resmic.2013.04.003.

29 Alex Orlek, Nicole Stoesser, Muna F Anjum, Michel Doumith, Matthew J Ellington, Tim Peto, Derrick Crook, Neil Woodford, A Sarah Walker, Hang Phan, et al. Plasmid classification in an era of whole-genome sequencing: application in studies of antibiotic resistance epidemiology. *Frontiers in Microbiology*, 8:182, 2017. doi:10.3389/fmicb.2017.00182.

30 Laxmi Parida. Using pq structures for genomic rearrangement phylogeny. *Journal of Computational Biology*, 13(10):1685–1700, 2006. doi:10.1089/cmb.2006.13.1685.

31 Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975. doi:10.1145/361219.361220.

32 Thomas Schmidt and Jens Stoye. Quadratic time algorithms for finding common intervals in two and more sequences. In *Combinatorial Pattern Matching*, pages 347–358. Springer, 2004. doi:10.1007/978-3-540-27801-6_26.

33 Frits CR Spieksma. On the approximability of an interval scheduling problem. *Journal of Scheduling*, 2(5):215–227, 1999. doi:10.1002/(SICI)1099-1425(199909/10)2:5<215::AID-JOS27>3.0.CO;2-Y.

34 Frits CR Spieksma and Yves Crama. *The complexity of scheduling short tasks with few starting times*. Rijksuniversiteit Limburg. Vakgroep Wiskunde, 1992.

35 Mark C Sulavik, Chad Houseweart, Christina Cramer, Nilofer Jiwani, Nicholas Murgolo, Jonathan Greene, Beth DiDomenico, Karen Joy Shaw, George H Miller, Roberta Hare, et al. Antibiotic susceptibility profiles of escherichia coli strains lacking multidrug efflux pump genes. *Antimicrobial Agents and Chemotherapy*, 45(4):1126–1136, 2001. doi:10.1128/AAC.45.4.1126-1136.2001.

36 Dina Svetlitsky, Tal Dagan, and Michal Ziv-Ukelson. Discovery of multi-operon colinear syntenic blocks in microbial genomes. *Bioinformatics*, 2020. doi:10.1093/bioinformatics/btaa503.

37 Roman L Tatusov, Michael Y Galperin, Darren A Natale, and Eugene V Koonin. The cog database: a tool for genome-scale analysis of protein functions and evolution. *Nucleic Acids Research*, 28(1):33–36, 2000. doi:10.1093/nar/28.1.33.

38 Tatiana Tatusova, Stacy Ciufo, Boris Fedorov, Kathleen O'Neill, and Igor Tolstoy. Refseq microbial genomes database: new representation and annotation strategy. *Nucleic Acids Research*, 42(D1):D553–D559, 2014. doi:10.1093/nar/gkt1274.

39 Takeaki Uno and Mutsunori Yagiura. Fast algorithms to enumerate all common intervals of two permutations. *Algorithmica*, 26(2):290–309, 2000. doi:10.1007/s004539910014.

40 René van Bevern, Matthias Mnich, Rolf Niedermeier, and Mathias Weller. Interval scheduling and colorful independent sets. *Journal of Scheduling*, 18(5):449–469, October 2015. doi:10.1007/s10951-014-0398-5.

41 Joachim Vandecraen, Michael Chandler, Abram Aertsen, and Rob Van Houdt. The impact of insertion sequences on bacterial genome plasticity and adaptability. *Critical Reviews in Microbiology*, 43(6):709–730, 2017. PMID: 28407717. doi:10.1080/1040841X.2017.1303661.

42 Alice R Wattam, David Abraham, Oral Dalay, Terry L Disz, Timothy Driscoll, Joseph L Gabbard, Joseph J Gillespie, Roger Gough, Deborah Hix, Ronald Kenyon, et al. Patric, the bacterial bioinformatics database and analysis resource. *Nucleic Acids Research*, 42(D1):D581–D591, 2014. doi:10.1093/nar/gkt1099.

43 Jonathan N Wells, L Therese Bergendahl, and Joseph A Marsh. Operon gene order is optimized for ordered protein complex assembly. *Cell Reports*, 14(4):679–685, 2016. doi:10.1016/j.celrep.2015.12.085.

44 Sascha Winter, Katharina Jahn, Stefanie Wehner, Leon Kuchenbecker, Manja Marz, Jens Stoye, and Sebastian Böcker. Finding approximate gene clusters with gecko 3. *Nucleic Acids Research*, 44(20):9600–9610, 2016. doi:10.1093/nar/gkw843.

45 G. R. Zimerman, D. Svetlitsky, M. Zehavi, and M. Ziv-Ukelson. Approximate search for known gene clusters in new genomes using pq-trees, 2020. arXiv:2007.03589.

A PQ-Tree Search is NP-Hard

In this section we prove Theorem 9 by describing a reduction from the Job Interval Selection problem (JISP) to PQ-Tree Search.

▶ **Theorem 9.** *PQ-Tree Search is* NP*-hard.*

Since its initial definition by Nakajima and Hakimi [26], JISP has seen several equivalent definitions [23, 33, 34, 40]. We use the following formulation for JISPk based on colors. Given γ k-tuples of intervals on the real line, where the intervals of every k-tuple have a different color i ($1 \leq i \leq \gamma$), select exactly one interval of each color (k-tuple) such that no two intervals intersect. The notation I_j^i is used to denote the interval that starts at s_{ij}, ends at f_{ij} (i.e. the interval $[s_{ij}, f_{ij}]$) and has the color i (i.e. it is a part of the i^{th} k-tuple).

JISP3 was shown to be NP-complete by Keil [23]. Crama et al. [34] showed that JISP3 is NP-complete even if all intervals are of length 2. We use these results to show that PQ-Tree Search is NP-hard.

The Reduction. Given an instance, J, of JISP3 where all intervals have length 2, an instance of PQ-Tree Search is created. It is easy to see that shifting all intervals by some constant does not change the problem. Hence, assume that the leftmost starting interval starts at 1. Let L be the rightmost ending point of an interval, so the focus can be only on the segment $[1, L]$ of the real line. Now, an instance of PQ-Tree Search (T, S, h, d_T, d_S) is constructed (an illustrated example is given in Figure S1 below):

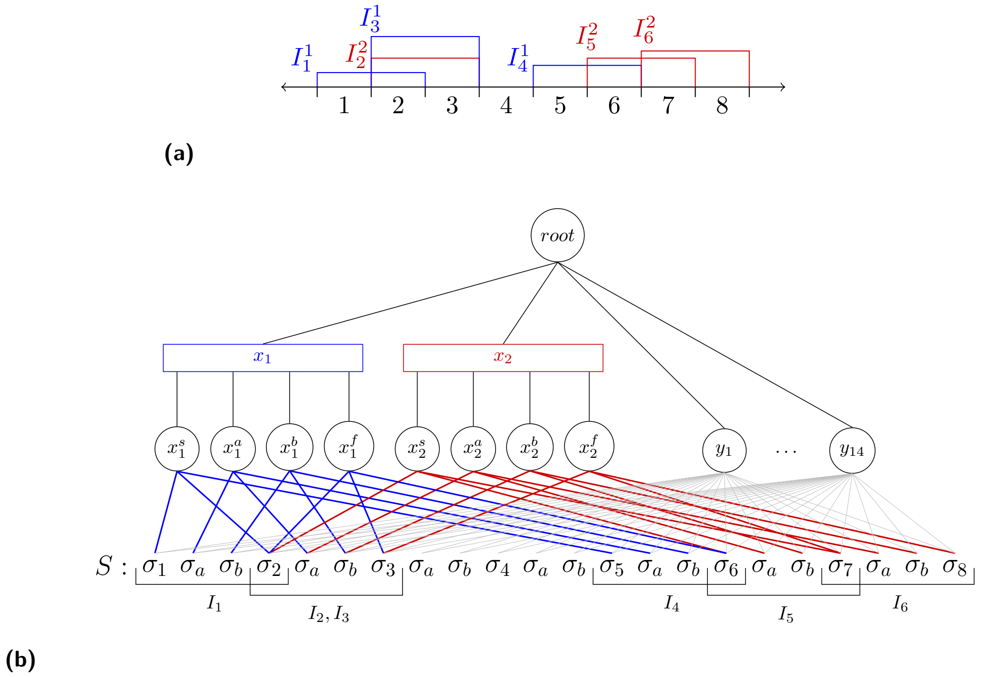

I_1^1
I_2^2
I_3^1
I_4^1
I_5^2
I_6^2
1 2 3 4 5 6 7 8
(a)
root
x_1
x_2
x_1^s x_1^a x_1^b x_1^f x_2^s x_2^a x_2^b x_2^f y_1 … y_{14}
$S: \sigma_1\ \sigma_a\ \sigma_b\ \sigma_2\ \sigma_a\ \sigma_b\ \sigma_3\ \sigma_a\ \sigma_b\ \sigma_4\ \sigma_a\ \sigma_b\ \sigma_5\ \sigma_a\ \sigma_b\ \sigma_6\ \sigma_a\ \sigma_b\ \sigma_7\ \sigma_a\ \sigma_b\ \sigma_8$
I_1
I_2, I_3
I_4
I_5
I_6
(b)

Figure S1 **(a)** The input of the reduction - a JISP3 instance J with intervals of length 2. **(b)** The output of the reduction - a PQ-Tree Search instance (T, S, h, d_T, d_S).

- **The PQ-tree *T*:** The root node, $root_T$, is a P-node with $3L-2-3\gamma$ children: $x_1, \ldots, x_\gamma$, $y_1, \ldots, y_{3L-2-4\gamma}$. The children of $root_T$ are defined as follows: for every color $1 \le i \le \gamma$, create a Q-node x_i with four children x_i^s, x_i^a, x_i^b, x_i^f; for every index $1 \le i \le 3L-2-3\gamma$, create a leaf y_i.
- **The string *S*:** Define $S = \sigma_1\sigma_a\sigma_b\sigma_2\sigma_a\sigma_b \ldots \sigma_a\sigma_b\sigma_L$.
- **The substitution function *h*:** For every interval of the color i, $I_j^i = [s_{ij}, f_{ij}]$, the function h returns $True$ for the following pairs: $(x_i^s, \sigma_{s_{ij}})$, $(x_i^f, \sigma_{f_{ij}})$, (x_i^a, σ_a) and (x_i^b, σ_b). In addition, every leaf y_r can be substituted by every letter of S, namely for every index $1 \le r \le 3L-2-3\gamma$ and for every $s \in \{a, b, 1, \ldots, L\}$ the function h returns $True$ for the pair (y_r, σ_s). For every other pair h returns $False$. For the optimization version of the problem, define a scored substitution function h', such that $h'(u,v) = 1$ if $h(u,v) = True$ and $h'(u,v) = -\infty$ if $h(u,v) = False$.
- **Number of deletions:** Define $d_T = 0$ and $d_S = 0$, i.e. deletions are forbidden from both tree and string.

An example of the reduction is shown in Figure S1. A collection of two 3-tuples (one blue and one red) where each interval is of length 2, i.e a JISP3 instance, is in Figure S1a. Running the reduction algorithm yields the PQ-Tree Search instance in Figure S1b. The pairs that can be substituted (i.e. the pairs for which h returns $True$) are given by the lines connecting the leafs of the PQ-tree and the letters of the string S. The nodes and substitutable pairs created due to the blue and red intervals in the JISP3 instance are marked in blue and red, respectively. The substitutable pairs containing a y node are marked in gray. Note that the colors given in Figure S1b are not a part of the PQ-Tree Search instance, and are given for convenience.

B Time and Space Complexity of the PQ-Tree Search Algorithm

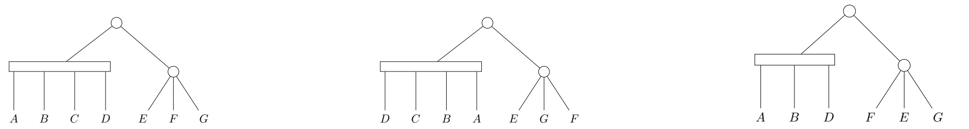
Here we prove Lemma 8.

Proof. The number of leaves in the PQ-tree T is m, hence there are $O(m)$ nodes in the tree, i.e the size of the first dimension of the DP table, $\mathcal{A}$, is $O(m)$. In the algorithm description (Section 3.1) a bound for the possible start indices of substrings derived from nodes in T is given. The node with the largest span in T is the root which has a span of m. The root is mapped to the longest substring when there are d_S deletions from the string. Hence, the size of the second dimension of $\mathcal{A}$ is $\Omega(n-(m+d_S)+1)=\Omega(n)$ (given that $d<m<<n$). The nodes with the smallest spans are the leaves, which have a span of 1, hence the size of the second dimension of $\mathcal{A}$ is $O(n)$. The third and fourth dimensions of $\mathcal{A}$ are of size d_T+1 and d_S+1, respectively. In total, the DP table $\mathcal{A}$ is of size $O(d_Td_Smn)$.

In the initialization step $O(d_Td_Smn)$ entries of $\mathcal{A}$ are computed in $O(1)$ time each. This holds because there are m leaves and n possible start indices for strings of length 1. The d_T and d_S factors come from the initialization of entries with $-\infty$. The P-mapping algorithm is called for every P-node in T and every possible start index i, i.e. the P-mapping algorithm is called $O(nm_p)$ times. Similarly, the Q-mapping algorithm is called $O(nm_q)$ times. Thus, it takes $O(n\ (m_p\cdot\text{Time(P-mapping)}+m_q\cdot\text{Time(Q-mapping)}))$ time to fill the DP table. In the final stage of the algorithm the maximum over the entries corresponding to every combination of deletion number and start index $(0\le k_T\le d_T, 0\le k_S\le d_S, 1\le i\le n-(\mathsf{span}(x)-d_T)+1\})$ is computed. So, it takes $O(d_Td_Sn)$ time to find the maximum score of a derivation. Tracing back through the DP table to find the actual mapping does not increase the time complexity.

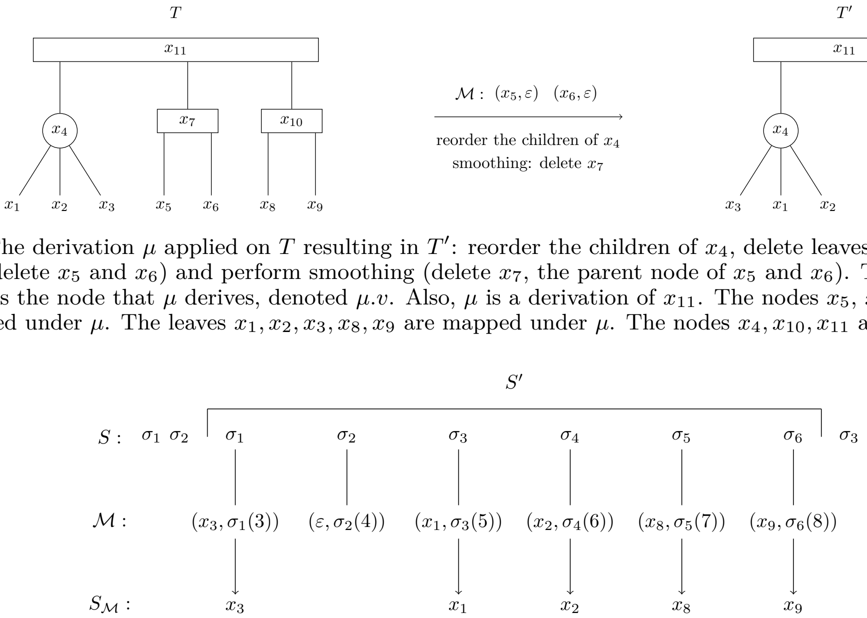
The P-mapping algorithm takes $O(\gamma 2^{\gamma}{d_T}^2{d_S}^2)$ time and $O(d_Td_S2^{\gamma})$ space, and the Q-mapping algorithm takes $O(\gamma {d_T}^2{d_S}^2)$ time and $O(d_Td_S\gamma)$ space. Thus, in total, our algorithm runs in $O(n(m_p\cdot\gamma 2^{\gamma}{d_T}^2{d_S}^2+m_q\cdot\gamma{d_T}^2{d_S}^2))=O(n\gamma{d_T}^2{d_S}^2(m_p\cdot 2^{\gamma}+m_q))$ time. Adding to the space required for the main DP table the space required for the P-mapping algorithm (the space needed for the Q-mapping algorithm is insignificant with respect to the P-mapping algorithm) results in a total space complexity of $O(d_Td_Smn)+O(d_Td_S2^{\gamma})=O(d_Td_S(mn+2^{\gamma}))$. This completes the proof. ◀

C Figures



(a) T_1. **(b)** T_2. **(c)** T_3.

Figure S2 Three different PQ-trees. By the definition of frontier, $F(T_1)=ABCDEFG$; $F(T_2)=DCBAEGF$; $F(T_3)=ABDFEG$. T_2 can be obtained from T_1 by reversing the children of a Q-node (the left child of the root) and by reordering the children of a P-node (the right child of the root), so $T_2\equiv T_1$. T_3 can be obtained from T_1 by deleting one leaf and permuting the children of the right child of the root, so $T_1\succeq_1 T_3$. Now, $T_2\succeq_1 T_3$ can be inferred, because the $\equiv$ is an equivalence relation.


T
x_{11}
x_4
x_7
x_{10}
x_1 x_2 x_3 x_5 x_6 x_8 x_9
$\mathcal{M}:$ (x_5, ε) (x_6, ε)
reorder the children of x_4
smoothing: delete x_7
T'
x_{11}
x_4
x_{10}
x_3 x_1 x_2 x_8 x_9
S'
$S:$ σ_1 σ_2 σ_1 σ_2 σ_3 σ_4 σ_5 σ_6 σ_3
$\mathcal{M}:$ $(x_3, \sigma_1(3))$ $(\varepsilon, \sigma_2(4))$ $(x_1, \sigma_3(5))$ $(x_2, \sigma_4(6))$ $(x_8, \sigma_5(7))$ $(x_9, \sigma_6(8))$
$S_{\mathcal{M}}:$ x_3 x_1 x_2 x_8 x_9

(a) The derivation μ applied on T resulting in T': reorder the children of x_4, delete leaves according to $\mathcal{M}$ (delete x_5 and x_6) and perform smoothing (delete x_7, the parent node of x_5 and x_6). The root of T, x_{11}, is the node that μ derives, denoted $\mu.v$. Also, μ is a derivation of x_{11}. The nodes x_5, x_6 and x_7 are deleted under μ. The leaves x_1, x_2, x_3, x_8, x_9 are mapped under μ. The nodes x_4, x_{10}, x_{11} are kept under μ.

(b) The derivation μ on S' resulting in $S_{\mathcal{M}}$: apply substitutions and deletions according to $\mathcal{M}$. The substring $S' = S[3:8]$ is the string that μ derives. The character $S[4]$ is deleted under μ. The characters $S[3], S[5], S[6], S[7], S[8]$ are mapped under μ.

Figure S3 An illustration of the derivation μ from the PQ-tree T to the substring S' under the one-to-one mapping $\mathcal{M}$ ($\mu.o$) with $\mu.del_T = \mathsf{del}_T(\mathcal{M}) = 2$ deletions from the tree and $\mu.del_S = \mathsf{del}_S(\mathcal{M}) = 1$ deletions from the string. The start point of the derivation ($\mu.s$) is 3. The end point of the derivation ($\mu.e$) is 8. Notice that $S_{\mathcal{M}} = F(T')$ and $T \succeq_2 T'$ which means that $S_{\mathcal{M}} \in C_2(T)$.

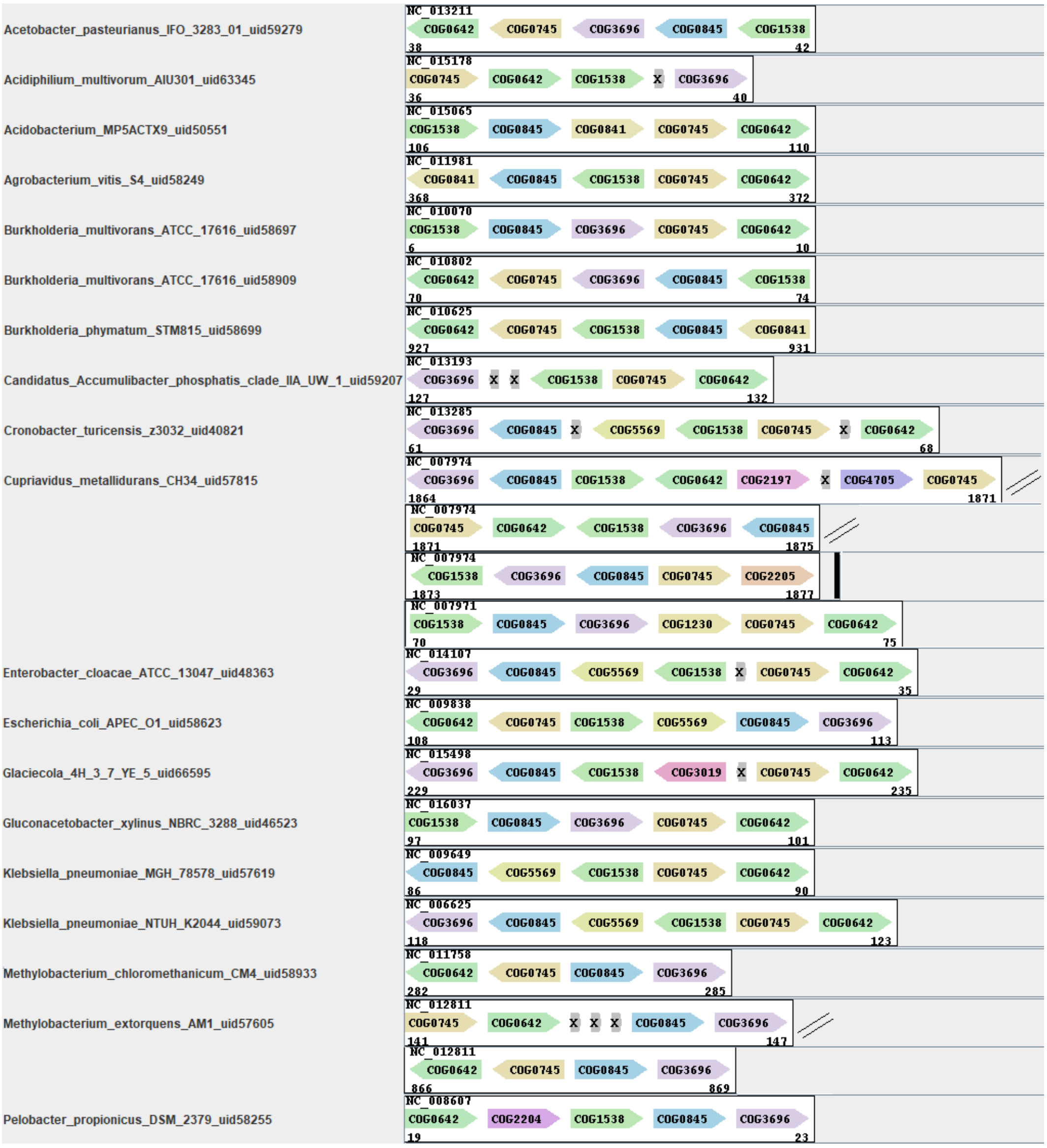
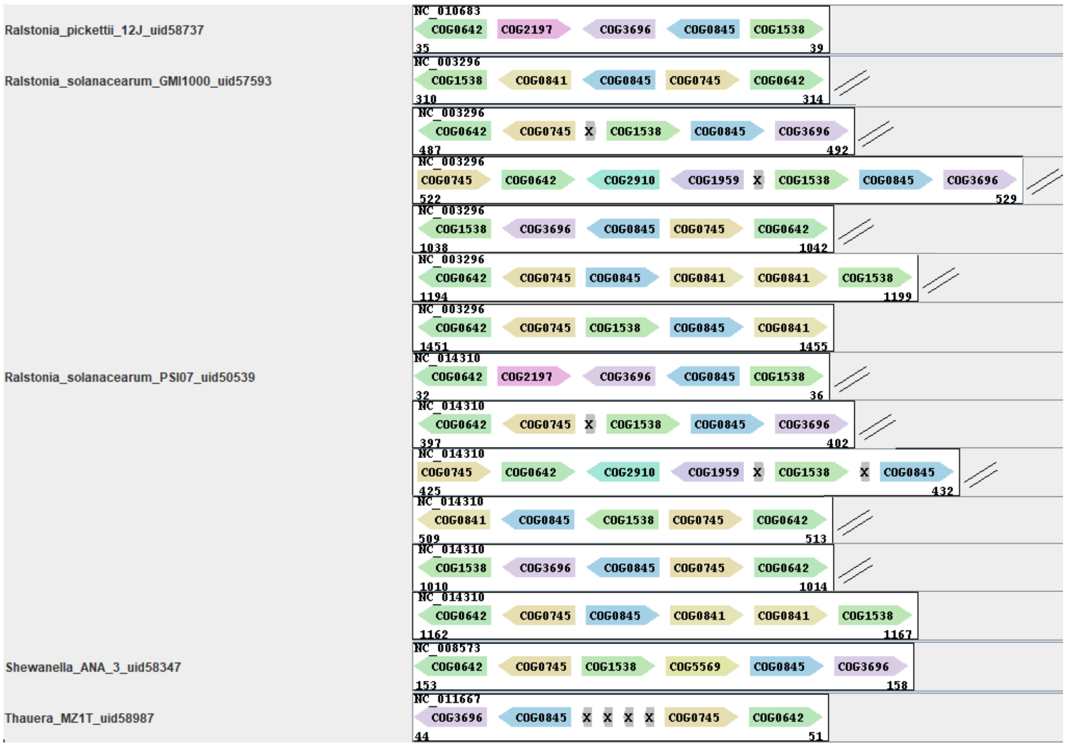

Acetobacter_pasteurianus_IFO_3283_01_uid59279
NC_013211
COG0642 COG0745 COG3696 COG0845 COG1538
38 42
Acidiphilium_multivorum_AIU301_uid63345
NC_015178
COG0745 COG0642 COG1538 X COG3696
36 40
Acidobacterium_MP5ACTX9_uid50551
NC_015065
COG1538 COG0845 COG0841 COG0745 COG0642
106 110
Agrobacterium_vitis_S4_uid58249
NC_011981
COG0841 COG0845 COG1538 COG0745 COG0642
368 372
Burkholderia_multivorans_ATCC_17616_uid58697
NC_010070
COG1538 COG0845 COG3696 COG0745 COG0642
6 10
Burkholderia_multivorans_ATCC_17616_uid58909
NC_010802
COG0642 COG0745 COG3696 COG0845 COG1538
70 74
Burkholderia_phymatum_STM815_uid58699
NC_010625
COG0642 COG0745 COG1538 COG0845 COG0841
927 931
Candidatus_Accumulibacter_phosphatis_clade_IIA_UW_1_uid59207
NC_013193
COG3696 X X COG1538 COG0745 COG0642
127 132
Cronobacter_turicensis_z3032_uid40821
NC_013285
COG3696 COG0845 X COG5569 COG1538 COG0745 X COG0642
61 68
Cupriavidus_metallidurans_CH34_uid57815
NC_007974
COG3696 COG0845 COG1538 COG0642 COG2197 X COG4705 COG0745
1864 1871
NC_007974
COG0745 COG0642 COG1538 COG3696 COG0845
1871 1875
NC_007974
COG1538 COG3696 COG0845 COG0745 COG2205
1873 1877
NC_007971
COG1538 COG0845 COG3696 COG1230 COG0745 COG0642
70 75
Enterobacter_cloacae_ATCC_13047_uid48363
NC_014107
COG3696 COG0845 COG5569 COG1538 X COG0745 COG0642
29 35
Escherichia_coli_APEC_O1_uid58623
NC_009838
COG0642 COG0745 COG1538 COG5569 COG0845 COG3696
108 113
Glaciecola_4H_3_7_YE_5_uid66595
NC_015498
COG3696 COG0845 COG1538 COG3019 X COG0745 COG0642
229 235
Gluconacetobacter_xylinus_NBRC_3288_uid46523
NC_016037
COG1538 COG0845 COG3696 COG0745 COG0642
97 101
Klebsiella_pneumoniae_MGH_78578_uid57619
NC_009649
COG0845 COG5569 COG1538 COG0745 COG0642
86 90
Klebsiella_pneumoniae_NTUH_K2044_uid59073
NC_006625
COG3696 COG0845 COG5569 COG1538 COG0745 COG0642
118 123
Methylobacterium_chloromethanicum_CM4_uid58933
NC_011758
COG0642 COG0745 COG0845 COG3696
282 285
Methylobacterium_extorquens_AM1_uid57605
NC_012811
COG0745 COG0642 X X X COG0845 COG3696
141 147
NC_012811
COG0642 COG0745 COG0845 COG3696
866 869
Pelobacter_propionicus_DSM_2379_uid58255
NC_008607
COG0642 COG2204 COG1538 COG0845 COG3696
19 23

Figure S4 This figure is continued in the next page.



(a)

COG0642 Signal transduction mechanisms | Signal transduction histidine kinase | BaeS
COG0745 Signal transduction mechanisms/Transcription | DNA-binding response regulator, OmpR family, contains REC and winged-helix (wHTH) domain | OmpR
COG3696 Inorganic ion transport and metabolism | Cu/Ag efflux pump CusA | CusA
COG0845 Cell wall/membrane/envelope biogenesis/Defense mechanisms | Multidrug efflux pump subunit AcrA (membrane-fusion protein) | AcrA
COG1538 Cell wall/membrane/envelope biogenesis | Outer membrane protein TolC | TolC
Inserted Genes:
COG3019 Function unknown | Uncharacterized conserved protein |
COG2197 Signal transduction mechanisms/Transcription | DNA-binding response regulator, NarL/FixJ family, contains REC and HTH domains | CitB
COG1230 Inorganic ion transport and metabolism | Co/Zn/Cd efflux system component | CzcD
COG0841 Defense mechanisms | Multidrug efflux pump subunit AcrB | AcrB
COG2205 Signal transduction mechanisms | K+-sensing histidine kinase KdpD | KdpD
COG2910 General function prediction only | Putative NADH-flavin reductase | YwnB
COG1959 Transcription | DNA-binding transcriptional regulator, IscR family | IscR
COG2204 Signal transduction mechanisms | DNA-binding transcriptional response regulator, NtrC family, contains REC, AAA-type ATPase, and a Fis-type DNA-binding domains | AtoC
COG5569 Inorganic ion transport and metabolism | Periplasmic Cu and Ag efflux protein CusF | CusF
COG4705 Function unknown | Uncharacterized membrane-anchored protein |

(b)

Figure S4 (Cont.) **(a)** The plasmid instances of the heavy metal efflux pump gene cluster discussed in Section 5.2. The COGs of the query gene cluster are: COG0642, COG0745, COG3639, COG0845, COG1538. The instances were identified using PQFinder and displayed using the graphical interface of the tool CSBFinder-S [36]. X indicates a gene with no COG annotation. The image was edited to display instances of the same genome in separate lines. **(b)** The functional description of the COGs shown in (a).

An Interpretable Classification Method for Predicting Drug Resistance in *M. Tuberculosis*

Hooman Zabeti
School of Computing Science, Simon Fraser University, Burnaby, Canada
hooman_zabeti@sfu.ca

Nick Dexter
Department of Mathematics, Simon Fraser University, Burnaby, Canada

Amir Hosein Safari
School of Computing Science, Simon Fraser University, Burnaby, Canada

Nafiseh Sedaghat
School of Computing Science, Simon Fraser University, Burnaby, Canada

Maxwell Libbrecht
School of Computing Science, Simon Fraser University, Burnaby, Canada

Leonid Chindelevitch
School of Computing Science, Simon Fraser University, Burnaby, Canada

Abstract

Motivation: The prediction of drug resistance and the identification of its mechanisms in bacteria such as *Mycobacterium tuberculosis*, the etiological agent of tuberculosis, is a challenging problem. Modern methods based on testing against a catalogue of previously identified mutations often yield poor predictive performance. On the other hand, machine learning techniques have demonstrated high predictive accuracy, but many of them lack interpretability to aid in identifying specific mutations which lead to resistance. We propose a novel technique, inspired by the group testing problem and Boolean compressed sensing, which yields highly accurate predictions and interpretable results at the same time.

Results: We develop a modified version of the Boolean compressed sensing problem for identifying drug resistance, and implement its formulation as an integer linear program. This allows us to characterize the predictive accuracy of the technique and select an appropriate metric to optimize. A simple adaptation of the problem also allows us to quantify the sensitivity-specificity trade-off of our model under different regimes. We test the predictive accuracy of our approach on a variety of commonly used antibiotics in treating tuberculosis and find that it has accuracy comparable to that of standard machine learning models and points to several genes with previously identified association to drug resistance.

2012 ACM Subject Classification Applied computing; Applied computing → Bioinformatics; Applied computing → Molecular sequence analysis

Keywords and phrases Drug resistance, whole-genome sequencing, interpretable machine learning, integer linear programming, rule-based learning

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.2

Supplementary Material `https://github.com/hoomanzabeti/TB_Resistance_RuleBasedClassifier`

Funding This project was funded by the Genome Canada grant BAC283. LC acknowledges additional funding from the CANSSI CRT and NSERC Discovery.

Acknowledgements The authors would like to thank Dr. Cedric Chauve, Dr. Ben Adcock and Matthew Nguyen for helpful discussions.

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti; Article No. 2; pp. 2:1–2:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Drug resistance is the phenomenon by which an infectious organism (also known as pathogen) develops resistance to one or more drugs that are commonly used in treatment [36]. In this paper we focus our attention on *Mycobacterium tuberculosis*, the etiological agent of tuberculosis, which is the largest infectious killer in the world today, responsible for over 10 million new cases and 2 million deaths every year [37].

The development of resistance to common drugs used in treatment is a serious public health threat, not only in low and middle-income countries, but also in high-income countries where it is particularly problematic in hospital settings [40]. It is estimated that, without the urgent development of novel antimicrobial drugs, the total mortality due to drug resistance will exceed 10 million people a year by 2050, a number exceeding the annual mortality due to cancer today [35].

Existing models for predicting drug resistance from whole-genome sequence (WGS) data broadly fall into two classes. The first, which we refer to as "catalogue methods," involves testing the WGS data of an isolate for the presence of point mutations (typically single-nucleotide polymorphisms, or SNPs) associated with known drug resistance. If one or more such mutations is identified, the isolate is declared to be resistant [46, 14, 4, 21, 15]. While these methods tend to be easy to understand and apply, they often suffer from poor predictive accuracy [43], especially in identifying novel drug resistance mechanisms or screening resistance to untested or rarely-used drugs.

The second class, which we will refer to as "machine learning methods", seeks to infer the drug resistance of an isolate by training complex models directly on WGS and drug susceptibility test (DST) data [48, 11, 2]. Such methods tend to result in highly accurate predictions at the cost of flexibility and interpretability - specifically, they typically do not provide any insights into the drug resistance mechanisms involved and often do not impose explicit limits on the predictive model's complexity. Learning approaches based on deep neural networks are one such example.

In this paper we propose a novel method, based on the group testing problem and Boolean compressed sensing (CS), for the prediction of drug resistance. Compressed sensing is a mathematical technique for sparse signal recovery from under-determined systems of linear equations [16], and has been successfully applied in many application areas including digital signal processing [13, 12], MRI imaging [26], radar detection [19], and computational uncertainty quantification [29, 9]. Under a sparsity assumption on the unknown signal vector, it has been shown that CS techniques enable recovery from far fewer measurements than required by the Nyquist-Shannon sampling theorem [5]. Boolean CS is a slight modification of the CS problem, replacing the matrix vector product with a Boolean OR operator [28], and has been successfully applied to areas such as group testing for infection [3, 1].

Our approach combines some of the flexibility and interpretability of catalogue methods with the accuracy of machine learning methods – specifically, this method is capable of recovering interpretable rules for predicting drug resistance that both result in a high classification accuracy as well as provide insights into the mechanisms of drug resistance. We show that our methods perform comparably to standard machine learning methods on *Mycobacterium tuberculosis* in terms of predicting first-line drug resistance, while accurately recovering many of the known mechanisms of drug resistance, and identifying some potentially novel ones.

2 Methods

Our proposed method is based on the rule-based classification technique introduced in [28], wherein group testing and Boolean CS are combined to determine subsets of infected individuals from large populations. In that setting the linear system encodes the infection status of the population through testing, and the solution, obtained from a suitable decoder, is a $\{0, 1\}$-valued vector representing the infection status of the individuals [6]. Since the infected group is assumed to be small, the solution vector is sparse and can be recovered using relatively few measurements with Boolean CS. The result of solving the Boolean CS problem can then be interpreted as a sparse set of rules for determining infections and used for classification on unseen data.

We present our methodology as follows. Section 2.1 introduces the group testing problem, and discusses how group testing can be combined with compressed sensing to deliver an interpretable predictive model. Section 2.2 introduces modifications to the standard setting to produce an accurate and flexible classifier, which can be tuned for specific evaluation metrics and tasks. Section 2.3 describes the tuning process for providing the desired trade-off between sensitivity and specificity in our model's predictions. Finally, Section 2.4 describes an approximation of the AUROC (area under receiver operating characteristic curve), a standard metric in machine learning, that is valid for evaluating the proposed approach.

2.1 Group testing and Boolean compressed sensing

We frame the problem of predicting drug resistance given sequence data as a group testing problem, originally introduced in [10]. This approach for detecting defective members of a set, was motivated by the need to screen large populations for syphilis while drafting citizens into military service for the United States during the World War II. The screening, performed by testing blood samples, was costly due to the low numbers of infected individuals. To make the screening more efficient, Dorfman suggested pooling blood samples into specific groups and testing the groups instead. A positive result for the group would imply the presence of at least one infected member. The problem then becomes to find the subset of individuals whose infected status would explain all of the positive results without invalidating any of the negative ones. By carefully selecting the groups, the total number of required tests m can be drastically reduced, i.e. if n is the population size, it is possible to achieve $m \ll n$.

Mathematically, a group testing problem with m tests can be described in terms of a Boolean matrix $A \in \{0, 1\}^{m \times n}$, where A_{ij} indicates the membership status of subject j in the i-th test group, and a Boolean vector $y \in \{0, 1\}^m$, where y_i represents the test result of the i-th group. If $w \in \{0, 1\}^n$ is a Boolean vector, with w_j representing the infection status of the j-th individual, then the result of all m tests will satisfy

$$y = A \vee w, \tag{1}$$

where $\vee$ is the Boolean inclusive OR operator, so that (1) can also be written

$$y_i = \bigvee_{j=1}^{n} A_{i,j} \wedge w_j \; \forall \; 1 \le i \le m.$$

If the vector w satisfying equation (1) is assumed to be sparse (i.e. there are few infected individuals), the problem of finding w is an instance of the sparse Boolean vector recovery problem:

$$\min \|w\|_0 \;\; \textbf{subject to} \;\; y = A \vee w, \tag{2}$$

where $\|w\|_0$ is the number of non-zero entries in the vector w.

Due to the non-convexity of the ℓ_0-norm and the nonlinearity of the Boolean matrix product, the combinatorial optimization problem (2) is well-known to be NP-hard, see, e.g., [16, Section 2.3] or [33]. In [27] a relaxation of (2) via linear programming is proposed, with the ℓ_0-norm replaced by the ℓ_1-norm (much like in basis pursuit for standard compressed sensing), and with the nonlinear Boolean matrix product also replaced with two closely related linear constraints. We recapitulate their equivalent 0-1 linear programming formulation here:

$$\begin{aligned} &\min \sum_{j=1}^{n} w_j \\ \text{s.t.}\quad & w \in \{0,1\}^n \\ & A_{\mathcal{P}} w \geq 1 \\ & A_{\mathcal{Z}} w = 0, \end{aligned} \tag{3}$$

where $\mathcal{P} = \{i : y_i = 1\}$ and $\mathcal{Z} = \{i : y_i = 0\}$ are the sets of groups that test positive and negative, respectively. However, this problem is also NP-hard, but can be made tractable for linear programming by relaxing the Boolean constraint on w in (3) to $0 \leq w_j \leq 1$ for all $j \in \{1, \dots, n\}$.

[28] extended this idea for interpretable rule-based classification, meanwhile proving recovery guarantees for the relaxed problem. Because the Boolean CS problem is based on Boolean algebra, the conditions on the Boolean measurement matrices A that guarantee exact recovery of K-sparse vectors via linear programming are quite different from those of standard CS. Specifically, these guarantees require the definition of K-disjunct matrices, i.e., matrices A for which all unions of their columns of size K do not contain any other columns of the original matrix. Constructions exist for matrices with $\mathcal{O}(K^2 \log(n))$ rows which satisfy this property. We also note that by introducing an *approximate disjunctness* property, allowing for matrices for which a fraction $(1-\varepsilon)$ of all $\binom{n}{K}$ possible K-subsets of the columns satisfy the disjunctness condition, it was shown in [30] that there exist constructions of measurement matrices A which allow for recovery from $\mathcal{O}(K^{3/2}\sqrt{\log(n/\varepsilon)})$ rows.

In the standard setting for uniform recovery results for CS, the measurement matrices A are subgaussian random matrices, i.e., having entries $A_{i,j}$ drawn independently according to a subgaussian distribution. Examples include $m \times n$ matrices consisting of Rademacher or Gaussian random variables, for which uniform recovery of K-sparse vectors via ℓ_1-minimization has been shown under the condition m is $\mathcal{O}(K \log(n/K))$, see, e.g. [16, Chapter 9] for more details. While subgaussian matrices have been shown to possess the most desirable recovery guarantees, they are not always applicable for every measurement scheme, in particular the one considered here.

In this work, we only consider the Boolean constrained problem, i.e. $w \in \{0,1\}^n$, though we adopt the slack variables and regularization proposed by [28] to trade off between the sparsity and the discrepancy with the test results of the relaxed problem. With these modifications in the Boolean constrained problem (3), our problem becomes:

$$\min \quad \sum_{j=1}^{n} w_j + \lambda \sum_{i=1}^{m} \xi_i \tag{4a}$$

$$\text{s.t.} \quad w \in \{0,1\}^n \tag{4b}$$

$$0 \leq \xi_i \leq 1, \quad i \in \mathcal{P} \tag{4c}$$

$$0 \leq \xi_i, \quad i \in \mathcal{Z} \tag{4d}$$

$$A_{\mathcal{P}} w + \xi_{\mathcal{P}} \geq 1 \tag{4e}$$

$$A_{\mathcal{Z}} w - \xi_{\mathcal{Z}} = 0, \tag{4f}$$

where $\lambda > 0$ is a regularization parameter. This Boolean constrained problem formulation can be solved via integer linear programming (ILP) techniques, see, e.g., [28].

2.1.1 Generalization to other contexts

The solution to the ILP (4) can be seen as an interpretable rule-based classifier in contexts beyond standard group testing. Given a rule for forming the matrix A, encoding binary attributes of a set of objects through multiple measurements or tests, and test data y, the general problem is to derive a Boolean disjunction that best classifies previously unseen objects from their features. In such a general setting, a context-specific technique for dichotomizing features may be needed [41]. However, in the case of drug resistance prediction, our features are the presence or absence of specific single-nucleotide polymorphisms (SNPs), and therefore no dichotomization is needed.

From now on, we assume that we have a binary labeled dataset $\mathcal{D}=\{(x_1,y_1),\ldots,(x_m,y_m)\}$, where the $x_i \in \mathcal{X} := \{0,1\}^n$ are n-dimensional binary feature vectors and the $y_i \in \{0,1\}$ are the binary labels. The feature matrix A is defined via $A_{i,j} = (x_i)_j$ (the j-th component of the i-th feature vector). If $\hat{w}$ is the solution of ILP (4) for this feature matrix and the label vector $y = (y_i)_{i=1}^m$, we define the classifier $\hat{c} : \mathcal{X} \to \{0,1\}$ as follows:

$$\hat{c}(x) = x \vee \hat{w}. \tag{5}$$

2.2 Our approach

The formulation of the ILP (4) is designed to provide a trade-off between the sparsity of a disjunctive rule and the total slack, a quantity that resembles (but does not equal) the training error. Unmodified, these conditions are not ideal for machine learning tasks: *i)* they do not allow for accurate expression of this error, and *ii)* they lack the ability to assign different weights to different components of the error. Such a weighting can play a large role in settings where the data is highly unbalanced, or when the cost of a false positive differs greatly from that of a false negative. We now describe an approach that provides more flexibility in the training process and performs better on specific tasks such as ours.

Recall that the regularization parameter λ in equation (4) provides control over the trade-off between the total slack and the sparsity of the solution. It is straightforward to generalize this term to provide useful information about the classifier's false positive and false negative rates. To obtain this information, we modify the ILP (4) in two ways.

For clarity, in the following section we assume that $\hat{c}$ is a binary classifier trained on a sample y with corresponding Boolean feature matrix A. In addition, unless otherwise stated, we refer to the misclassification of a training sample as a false negative if it has label 1 (is in $\mathcal{P}$), and as a false positive if it has label 0 (is in $\mathcal{Z}$). For instance, in the case of drug resistance, a false negative would mean that we incorrectly predict a drug-resistant isolate as sensitive, while a false positive would mean that we predict a drug-sensitive isolate as resistant.

First, note that in ILP (4), $\xi_{\mathcal{P}}$ corresponds to the training error of $\hat{c}$ on the positively labeled subset of the data, while $\xi_{\mathcal{Z}}$ does not correspond to its training error on the negatively labeled subset. This follows from the fact that A is a binary matrix and w is a binary vector, so $\xi_{\mathcal{P}}$ is also a binary vector, with

$$\sum_{i \in \mathcal{P}} \xi_i = 1^T \xi_{\mathcal{P}} = \text{FN}, \tag{6}$$

the number of false negatives. On the other hand, to obtain the number of false positives (FP) we need to modify the constraints (4d) and (4f) by setting

$$\xi_i \in \{0,1\}, \quad i \in \mathcal{Z} \tag{7}$$

and replacing $A_{\mathcal{Z}}w - \xi_{\mathcal{Z}} = 0$ with the inequalities:

$$A_{\mathcal{Z}}w - \xi_{\mathcal{Z}} \geq 0, \tag{8a}$$

$$\alpha_i \xi_i - A_i w \geq 0 \; \forall \; i \in \mathcal{Z}, \tag{8b}$$

where $\alpha_i = \sum_{j=1}^{n} A_{i,j}$ and A_i represent ith row of A. Note that the motivation behind this replacement is to count the number of non-zero elements of $A_{\mathcal{Z}}w$ by $\xi_{\mathcal{Z}}$. Therefore, we can observer that eq.(8a) ensure that $\xi_i = 0$ if $A_i w = 0$ and eq.(8b) ensures that $\xi_i = 1$ if $A_i w > 0$. However, eq.(8a) can be eliminated in those settings where the $\xi_{\mathcal{Z}}$ enter the objective function to be minimized with a positive coefficient. We will see similar situations in the following section.

After these modifications, we obtain

$$\sum_{i \in \mathcal{Z}} \xi_i = 1^T \xi_{\mathcal{Z}} = \text{FP}. \tag{9}$$

To provide the desired flexibility, we further split the regularization term into two terms corresponding to the positive class $\mathcal{P}$ and the negative class $\mathcal{Z}$:

$$\lambda_{\mathcal{P}} \sum_{i \in \mathcal{P}} \xi_i + \lambda_{\mathcal{Z}} \sum_{k \in \mathcal{Z}} \xi_k. \tag{10}$$

The general form of the new ILP is now as follows:

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} w_j + \lambda_{\mathcal{P}} \sum_{i \in \mathcal{P}} \xi_i + \lambda_{\mathcal{Z}} \sum_{k \in \mathcal{Z}} \xi_k \\ \text{s.t.} \quad & w \in \{0,1\}^n \\ & 0 \leq \xi_i \leq 1, \quad i \in \mathcal{P} \\ & \xi_i \in \{0,1\}, \quad i \in \mathcal{Z} \\ & A_{\mathcal{P}}w + \xi_{\mathcal{P}} \geq 1 \\ & \alpha_i \xi_i - A_i w \geq 0 \; \forall \; i \in \mathcal{Z} \end{aligned} \tag{11}$$

In this new formulation, $\lambda_{\mathcal{P}}$ and $\lambda_{\mathcal{Z}}$ control the trade-off between the false positives and the false negatives, and jointly influence the sparsity of the rule. This formulation can be further tailored to optimize specific evaluation metrics. In the following section we demonstrate this for sensitivity and specificity, as an example.

2.3 Optimizing sensitivity and specificity

Since the ILP formulation in (11) provides us with direct access to the two components of the training error, we may modify the classifier to optimize a specific evaluation metric. For instance, assume that we would like to train the classifier $\hat{c}$ to maximize the sensitivity at a given specificity threshold $\bar{t}$. First, recall that

$$\text{Specificity} = \frac{\text{TN}}{\text{TN+FP}} = 1 - \frac{\text{FP}}{\text{N}}, \tag{12}$$

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP+FN}} = 1 - \frac{\text{FN}}{\text{P}}. \tag{13}$$

From equation (10), equation (12) and the definition of $\mathcal{Z}$, we get the constraint

$$\bar{t} \leq 1 - \frac{1^T \xi_{\mathcal{Z}}}{|\mathcal{Z}|} \iff 1^T \xi_{\mathcal{Z}} \leq (1 - \bar{t})|\mathcal{Z}|. \tag{14}$$

Our objective is to maximize sensitivity, which is equivalent to minimizing $\sum_{i \in \mathcal{P}} \xi_i$ by equations (13) and (6). Hence, the ILP (11) can be modified as follows:

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} w_j + \lambda_{\mathcal{P}} \sum_{i \in \mathcal{P}} \xi_i \\ \text{s.t.} \quad & w \in \{0,1\}^n \\ & 0 \le \xi_i \le 1, \quad i \in \mathcal{P} \\ & \xi_i \in \{0,1\}, \quad i \in \mathcal{Z} \\ & A_{\mathcal{P}} w + \xi_{\mathcal{P}} \ge 1 \\ & \alpha_i \xi_i - A_i w \ge 0 \; \forall \, i \in \mathcal{Z} \\ & 1^T \xi_{\mathcal{Z}} \le (1 - \bar{t})|\mathcal{Z}|. \end{aligned} \tag{15}$$

The maximum specificity at given sensitivity can be found analogously.

2.4 Approximating the AUROC

In this section we compute an analog of the AUROC[1] of our classifier given a limit on rule size. Recall that the ROC is a plot demonstrating the performance of a score-producing classifier at different score thresholds, created by plotting the true positive rate (TPR) against the false positive rate (FPR). However, since the rule-based classifier produced by ILP (11) is a discrete classifier, it cannot produce a ROC curve in the usual way. To create a ROC curve for this classifier, we compute the true positive rate (TPR) for different values of the false positive rate (FPR). In addition, we set a limit on the rule size (sparsity) of the classifier.

More precisely, we create the ROC curve by incrementally changing the FPR and computing the optimum value of the TPR. To do so, we put varying upper bounds on the FPR and proceed analogously to the previous section. For instance, assume that we would like to get the best TPR value when the FPR is at most $\hat{t}$, where $0 \le \hat{t} \le 1$, meaning that

$$\text{FPR} = \frac{\text{FP}}{\text{N}} \le \hat{t}. \tag{16}$$

From equations (10), (16) and the definition of $\mathcal{Z}$ we get

$$\frac{1^T \xi_{\mathcal{Z}}}{|\mathcal{Z}|} \le \hat{t} \iff 1^T \xi_{\mathcal{Z}} \le \hat{t}|\mathcal{Z}|. \tag{17}$$

Assuming further that the limit on rule size is equal to $\hat{s}$, we have the following constraint:

$$1^T w \le \hat{s}. \tag{18}$$

[1] the Area Under the Receiver Operating Characteristic Curve

Therefore, the modified version of the ILP (11) suitable for computing an AUROC is:

$$\begin{aligned} \min \quad & \sum_{i \in \mathcal{P}} \xi_i \\ \text{s.t.} \quad & w \in \{0,1\}^n \\ & 0 \leq \xi_i \leq 1, \quad i \in \mathcal{P} \\ & \xi_i \in \{0,1\}, \quad i \in \mathcal{Z} \\ & A_{\mathcal{P}} w + \xi_{\mathcal{P}} \geq 1 \\ & \alpha_i \xi_i - A_i w \geq 0 \; \forall \, i \in \mathcal{Z} \\ & 1^T w \leq \hat{s} \\ & 1^T \xi_{\mathcal{Z}} \leq \hat{t} |\mathcal{Z}|. \end{aligned} \tag{19}$$

We utilize the CPLEX optimizer [20] to solve the ILP in (19).

3 Implementation

All the methods in this paper are implemented in the Python programming language. We use a Scikit-learn [38] implementation for the machine learning models and the CPLEX optimizer version 12.10.0 [20], together with its Python API, for our method.

3.1 Data

To obtain a dataset to train and evaluate our method on, we combine data from the Pathosystems Resource Integration Center (PATRIC)[47] and the Relational Sequencing TB Data Platform (ReSeqTB)[45]. This results in 8000 isolates together with their resistant/susceptible status (label) for seven drugs, including five first-line drugs (rifampicin, isoniazid, pyrazinamide, ethambutol, and streptomycin) and two second-line drugs (kanamycin and ofloxacin) [34, 8]. The short-read whole genome sequences of these 8000 isolates are downloaded from the European Nucleotide Archive [23] and the Sequence Read Archive [24]. The accession numbers used to obtain the data in our study were: ERP[000192, 006989, 008667, 010209, 013054, 000520], PRJEB[10385, 10950, 14199, 2358, 2794, 5162, 9680], PRJNA[183624, 235615, 296471], and SRP[018402, 051584, 061066].

In order to map the raw sequence data to the reference genome, we use a method similar to that used in previous work [7, 8]. We use the BWA software [25], specifically, the bwa-mem program. We then call the single-nucleotide polymorphisms (SNPs) of each isolate with two different pipelines, SAMtools [18] and GATK [39], and take the intersection of their calls to ensure reliability. The final dataset, which includes the position as well as the reference and alternative allele for each SNP [8], is used as the input to our classifier.

Starting from this input we create a binary feature matrix, where each row represents an isolate and each column indicates the presence or absence of a particular SNP. For each drug, we group all the SNPs with identical presence/absence patterns into a single column, since at most one SNP in a group would ever be selected to be part of a rule. The number of labeled and resistant isolates and of SNPs and SNP groups for each drug is stated in Table 1.

3.2 Train-Test split

To evaluate our classifier we use a stratified train-test split, where the training set contains 80% and the testing set contains 20% of data.

Table 1 Summary of number of isolates in our data.

Drug	Number of isolates	Number of resistant isolates	Number of SNPs	Number of SNP groups
Ethambutol	6,096	1,407	666,349	55,164
Isoniazid	7,734	3,445	666,349	65,090
Kanamycin	2,436	697	666,349	21,513
Ofloxacin	2,911	800	666,349	23,905
Pyrazinamide	3,858	754	666,349	33,942
Rifampicin	7,715	2,968	666,349	65,379
Streptomycin	5,125	2,104	666,349	45,037

3.3 AUROC comparison

The AUROC of our model was computed for two purposes: first, to investigate the effect of the classifier's sparsity (rule size) on its performance, and second, to compare this performance to that of other machine learning methods. We calculated the AUROC of classifiers with various limits on rule size, selected from $\{1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200\}$. For each rule size, we use the formulation in subsection 2.4, increasing the FPR upper bound from 0 to 1 in increments of 0.1. We then train a classifier by using the ILP (19), and compute the effective FPR and TPR. Lastly, we create the ROC curve by plotting the TPRs against the FPRs, and compute the AUROC.

To compare the performance of our model with other machine learning models, we also compute the AUROC of the Random Forest (RF) and ℓ_1-regularized Logistic Regression (LR) models. For these models, we first perform hyper-parameter tuning using grid search with three-fold cross validation, and then select the model with the highest AUROC.

3.4 Sensitivity at a fixed specificity

As another evaluation criteria we compute the sensitivity of our model at a desired specificity level (i.e. β% specificity). To do so, we use the ILP (15). In this formulation, the $\lambda_{\mathcal{P}}$ parameter can be tuned to provide the desired trade-off between the sparsity of the classifier (i.e., rule size) and the number of false negatives. However, in order to make a consistent comparison between the trained models for different drugs, we set a specific limit on rule size and use ILP (19) with the last constraint replaced by the last constraint of ILP (15), i.e. with (17) replaced with (14).

4 Results

Evaluating the performance of an interpretable predictive model can be challenging. While most evaluation methods focus on predictive accuracy, it is essential to assess the model's interpretability. Even though there is no consensus on the definition of interpretability, the "Predictive, Descriptive, Relevant" (PDR) framework introduced by [32] provides general insights into interpretable models, by emphasizing the balance between these characteristics. In this section, we use the PDR framework to evaluate our models in the following ways.

First, in Section 4.1, we assess our method's predictive accuracy by comparing it with RF and LR. At this step we do not have any specific restriction on the rule size, and we report the best AUROC that our model can achieve based on the settings in Section 3.3.

Second, in Section 4.2, we compare the AUROC produced by our method for different limits on rule size. This comparison between the method at different parameter values helps us evaluate its ability to produce a simple model (i.e. a model with a fairly small rule size) with a high AUROC. The simpler models are easier to understand for human users. In

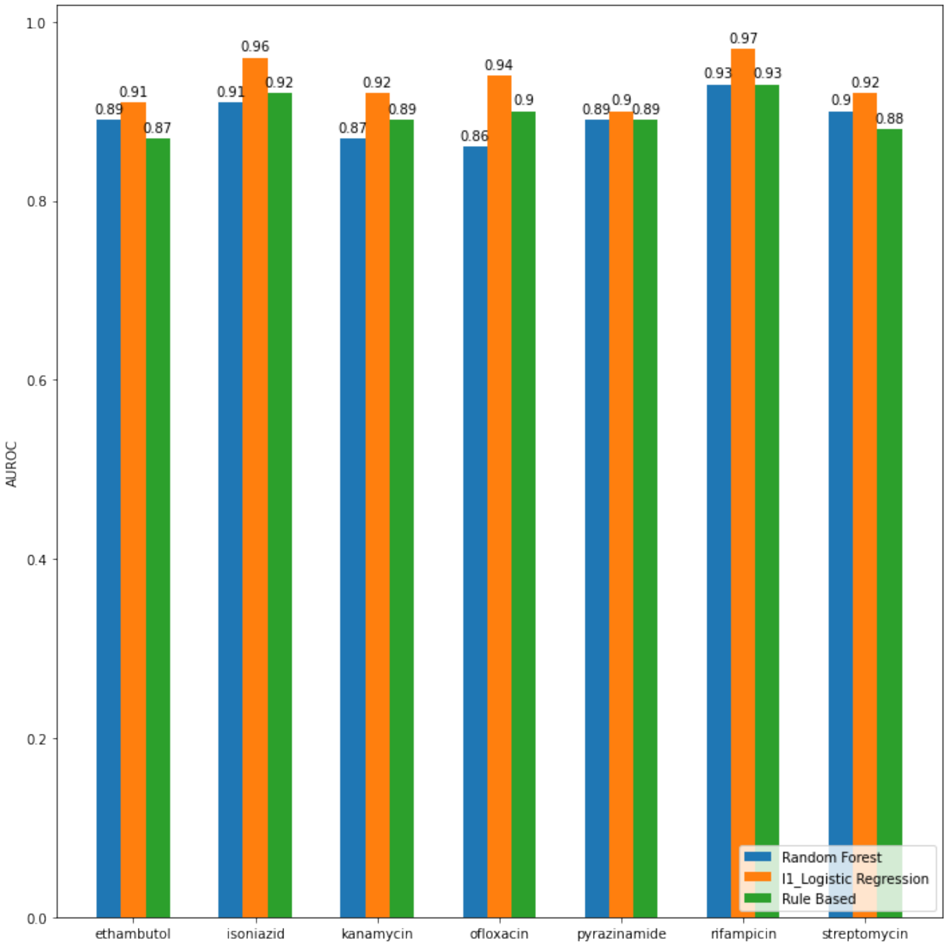


Figure 1 Comparison between the test AUROC of our rule-based model (with no limit imposed on the rule size), ℓ_1-regularized logistic regression and Random Forest.

this paper, we define the descriptiveness of a model by its simplicity (its rule size, i.e., the number of SNPs needed to define it). In addition, we evaluate our method's sensitivity by comparing it with LR and RF. To do so, we compute and compare the sensitivity of these three models at a specificity near 90%. More specifically, this comparison uses the specificity level achieved by the rule-based model that is closest to 90% (in practice, this is always between 88% and 92% for this dataset), since the rule-based model does not achieve every possible specificity level when given a limit on rule size. For this evaluation, we limit model complexity by setting a limit of 20 on the rule size.

Finally, in Section 4.3, we assess the relevance of the model produced by our method by observing the fraction of SNPs used by the model that are located in genes previously reported to be associated with drug resistance. Note that, unlike the approach in [48], we do not limit the genes *a priori* to those with known associations with drug resistance.

4.1 Our models produce competitive AUROCs

Figure 1 illustrates the results of comparing our model to LR and RF. In this figure, we can see that LR provides a higher AUROC for all 7 drugs, but our model produces slightly higher AUROCs than RF for 3 of the drugs, identical AUROCs for 2 other drugs and slightly lower ones for the remaining 2.

Table 2 Comparison between AUROCs of models produced by our method with different rule size limits. We observe that even small rule sizes produce models with a high AUROC.

Drug	Rule size $\leq$ 10	Rule size $\leq$ 20	Rule size $\leq$ 30	Rule size $\leq$ 40	Max AUROC
Ethambutol	0.86	0.86	0.85	**0.86**	0.87
Isoniazid	0.88	0.89	0.90	**0.91**	0.92
Kanamycin	0.88	0.89	**0.89**	0.88	0.89
Ofloxacin	0.90	0.87	**0.90**	0.88	0.90
Pyrazinamide	0.88	0.88	0.88	**0.89**	0.89
Rifampicin	0.90	0.92	0.92	**0.93**	0.93
Streptomycin	0.84	0.86	0.85	**0.87**	0.88

4.2 Our approach is able to produce simple models with high AUROC

Figure 2 demonstrates the change in AUROC as we increase the limit on the rule size. Our results show that as the limit on the rule size increases, we get higher AUROC on the training set. However, on the test set, we see that the AUROC increases more slowly after a rule size limit of 10, and eventually starts to decrease.

As shown in Figure 2 and Table 2, the AUROC does not increase significantly beyond a rule size limit of 10. Thus, our method is capable of producing models with a rule sizes small enough to keep the model simple yet keep the AUROC within 1% of the maximum.

Table 3 shows the same trend for the ℓ_1-regularized logistic regression. We see that, at the low rule-size limits (such as 10 and 20), our approach produces a comparable performance to that of ℓ_1-regularized logistic regression, while it is slightly worse for larger rule-size limits. At the same time, as we show in Figures 4a and 4b below, our approach results in the recovery of a lot more genes known to be associated with drug resistance than logistic regression.

4.3 Our model uses genes previously associated to drug resistance

Our results show that the models produced by our method contains many SNPs in genes previously associated with drug resistance in *Mycobacterium tuberculosis*. Due to the large size of SNP groups (SNPs in perfect linkage disequilibrium), the causality of specific SNPs remains difficult to determine. However, many of the genes known to be relevant to resistance mechanisms appear among the possible variants that are pointed to by the selected groups of duplicated SNPs.

In Figure 4a we show the number of SNPs within different classes of genes found by our approach with rule size $\leq$ 20 and specificity $\geq$ 90%, where each gene is classified according to whether it has a known association to drug resistance ("known") or not ("unknown"), with

Table 3 Comparison between AUROCs of models produced by ℓ_1-regularized logistic regression with different numbers of non-zero regression coefficients.

Drug	Non-zero coef. $\leq$ 10	Non-zero coef. $\leq$ 20	Non-zero coef. $\leq$ 30	Non-zero coef. $\leq$ 40	Max AUROC
Ethambutol	0.87	0.87	0.88	0.89	0.91
Isoniazid	0.90	0.91	0.92	0.93	0.96
Kanamycin	0.90	0.91	0.91	0.92	0.92
Ofloxacin	0.86	0.90	0.94	0.94	0.94
Pyrazinamide	0.81	0.87	0.89	0.89	0.90
Rifampicin	0.92	0.92	0.94	0.94	0.97
Streptomycin	0.88	0.88	0.89	0.90	0.92

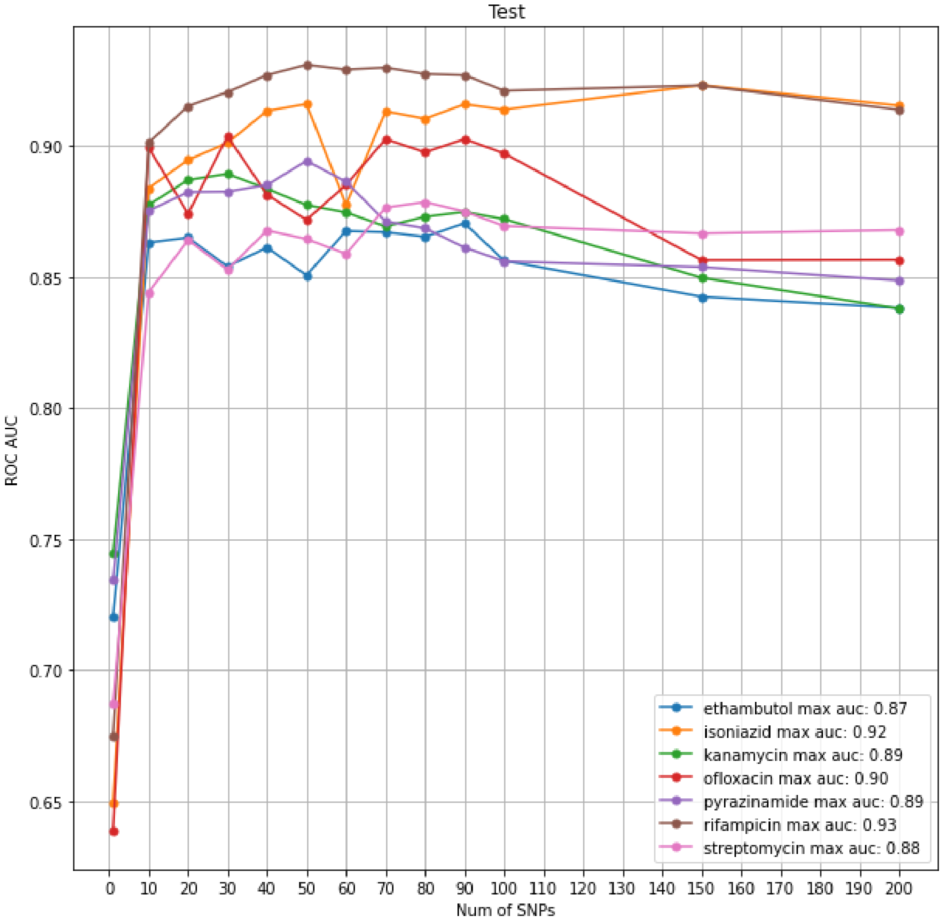


Figure 2 Test AUROC for models trained on each drug with various rule size limits. Beyond a certain rule size, which varies with the drug, the AUROC of the predictive model no longer improves.

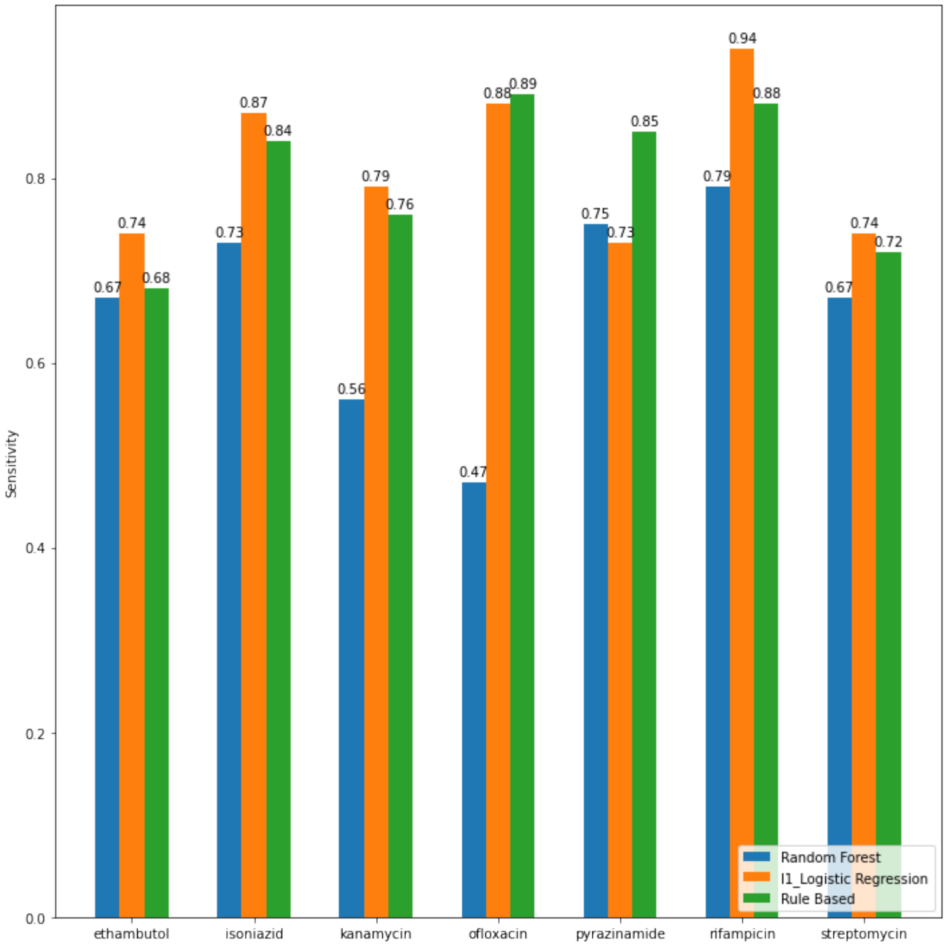
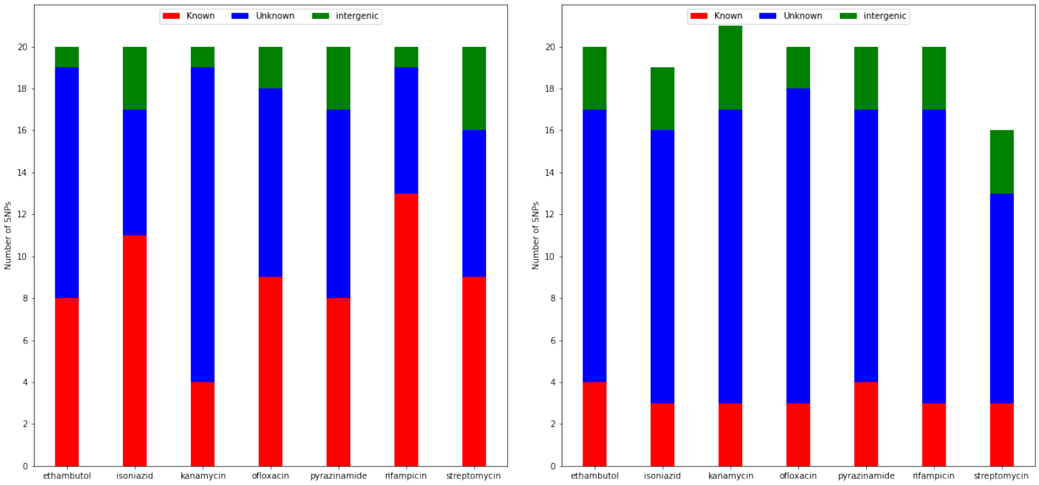


Figure 3 Comparison between the sensitivity of our rule-based method with the rule size limit set to 20, ℓ_1-Logistic regression and Random Forest at around 90% specificity on the testing data.



(a) **(b)**

Figure 4 (a) The number of SNPs in genes with known association to drug resistance, genes without such an association, and intergenic regions, in our models with at most 20 SNPs and a specificity of $\geq 90\%$. (b) The same numbers for ℓ_1-Logistic regression models with as close as possible to 20 non-zero regression coefficients.

an additional class for SNPs in intergenic regions. We show these numbers for ℓ_1-Logistic regression models with as close as possible to 20 non-zero regression coefficients in Figure 4b. A comparison between these figures suggests that when both approaches are restricted to a small number of features, our approach detects more relevant SNPs than ℓ_1-logistic regression. The list of "known" genes, selected through a data-driven and consensus-driven process by a panel of experts, is the one in [31], containing 183 out of over 4,000 *M. tuberculosis* genes. We note that in both cases, a group of SNPs in perfect linkage disequilibrium was coded as "known" if at least one of the SNPs was contained in a known gene, "intergenic" if none of them appeared in a gene, and as "unknown" otherwise.

4.4 Running time

We run our code on a cluster node with 2 CPU sockets, each with an 8-core 2.60 GHz Intel Xeon E5-2640 v3 with 32 threads. The training of a single model with fixed hyper-parameters takes between 1 and 8 minutes. This suggests that once a suitable value is chosen for the hyper-parameters, the optimization used to determine the optimal rule can be performed efficiently. Overall, producing the ROC curve for each drug takes between 3 and 18 hours, depending on the number of labeled isolates available for each drug.

5 Conclusion

In this paper, we introduced a new approach for creating rule-based classifiers. Our method utilizes the group testing problem and Boolean compressed sensing. It can produce interpretable, highly accurate, flexible classifiers which can be optimized for particular evaluation metrics.

We used our method to produce classifiers for predicting drug resistance in *Mycobacterium tuberculosis*. The classifiers' predictive accuracy was tested on a variety of antibiotics commonly used for treating tuberculosis, including five first-line and two second-line drugs.

We show that our method could produce classifiers with a high AUROC, slightly less than that of unrestricted ℓ_1-Logistic regression, and comparable to Random Forest, as well as ℓ_1-Logistic regression restricted to a comparably small number of selected features for interpretability. In addition, we show that our method is capable of producing accurate models with a rule size small enough to keep the model understandable for human users. Finally, we show that our approach can provide useful insights into its input data - in this case, it could help identify genes associated with drug resistance.

We note that the presence of SNPs with identical presence/absence patterns, which would be referred to as being in perfect linkage disequilibrium (LD) in genetics [42], is common in bacteria such as *Mycobacterium tuberculosis* whose evolution is primarily clonal [17]. For this reason, while the grouping of such SNPs together substantially greatly simplifies the computational task at hand, it is challenging to ascertain the exact representative of each group that should be selected to determine the drug resistance status of an isolate. Determining this representative would likely require larger sample sizes or a built-in prior knowledge of the functional effects of individual SNPs.

We also note that the genes we define as having a known association to drug resistance are not specific to the drug being tested, i.e. some of them may have been found to be associated with the resistance to a drug other than the one being predicted. This is to be expected, however, as the distinct resistance mechanisms are generally less numerous than antibiotics [44]. It will be interesting to see whether methods such as ours are able to detect specific, for instance, by testing it on data for newly developed antibiotics such as bedaquiline and delamanid [22].

Our goal in this paper was to introduce a novel method for producing interpretable models and explore its accuracy, descriptive ability, and relevance in detecting drug resistance in *Mycobacterium tuberculosis* isolates. In this study, the focus was mostly on the predictive accuracy, and we will explore the similarities and differences between our model and other interpretable techniques (both model-based and *post-hoc* ones) in future work.

References

1 Matthew Aldridge, Oliver Johnson, Jonathan Scarlett, et al. Group testing: an information theory perspective. *Foundations and Trends® in Communications and Information Theory*, 15(3-4):196–392, 2019.

2 Gustavo Arango-Argoty, Emily Garner, Amy Pruden, Lenwood S Heath, Peter Vikesland, and Liqing Zhang. DeepARG: a deep learning approach for predicting antibiotic resistance genes from metagenomic data. *Microbiome*, 6(1):1–15, 2018.

3 G. K. Atia and V. Saligrama. Boolean compressed sensing and noisy group testing. *IEEE Transactions on Information Theory*, 58(3):1880–1901, 2012.

4 P. Bradley, N. Gordon, T Walker, et al. Rapid antibiotic-resistance predictions from genome sequence data for Staphylococcus aureus and Mycobacterium tuberculosis. *Nature Communications*, 6, 2015.

5 E. J. Candes and M. B. Wakin. An introduction to compressive sampling. *IEEE Signal Processing Magazine*, 25(2):21–30, 2008.

6 Albert Cohen, Wolfgang Dahmen, and Ronald DeVore. Compressed sensing and best k-term approximation. *Journal of the American mathematical society*, 22(1):211–231, 2009.

7 Francesc Coll, Ruth McNerney, José Afonso Guerra-Assunção, Judith R. Glynn, João Perdigão, Miguel Viveiros, Isabel Portugal, Arnab Pain, Nigel Martin, and Taane G. Clark. A robust snp barcode for typing mycobacterium tuberculosis complex strains. *Nature Communications*, 2014. doi:10.1038/ncomms5812.

8 Wouter Deelder, Sofia Christakoudi, Jody Phelan, Ernest Diez Benavente, Susana Campino, Ruth McNerney, Luigi Palla, and Taane Gregory Clark. Machine learning predicts accurately Mycobacterium tuberculosis drug resistance from whole genome sequencing data. *Frontiers in Genetics*, 10:922, 2019.

9 Alireza Doostan and Houman Owhadi. A non-adapted sparse approximation of PDEs with stochastic inputs. *Journal of Computational Physics*, 230(8):3015–3034, 2011.

10 Robert Dorfman. The detection of defective members of large populations. *The Annals of Mathematical Statistics*, 14(4):436–440, 1943.

11 Sorin Drăghici and R Brian Potter. Predicting HIV drug resistance with neural networks. *Bioinformatics*, 19(1):98–107, 2003.

12 M. F. Duarte and Y. C. Eldar. Structured compressed sensing: From theory to applications. *IEEE Transactions on Signal Processing*, 59(9):4053–4085, 2011.

13 Y.C. Eldar and G. Kutyniok. *Compressed Sensing: Theory and Applications*. Cambridge University Press, 2012. URL: `https://books.google.ca/books?id=9ccLAQAAQBAJ`.

14 Coll F, McNerney R, Preston MD, et al. Rapid determination of anti-tuberculosis drug resistance from whole-genome sequences. *Genome Med.*, 7:51, 2015.

15 Silke Feuerriegel, Viola Schleusener, Patrick Beckert, Thomas A. Kohl, Paolo Miotto, Daniela M. Cirillo, Andrea M. Cabibbe, Stefan Niemann, and Kurt Fellenberg. PhyResSE: a web tool delineating Mycobacterium tuberculosis antibiotic resistance and lineage from whole-genome sequencing data. *Journal of Clinical Microbiology*, 53(6):1908–1914, 2015.

16 S. Foucart and H. Rauhut. *A Mathematical Introduction to Compressive Sensing*. Applied and Numerical Harmonic Analysis. Springer New York, 2013. URL: `https://books.google.ca/books?id=zb28BAAAQBAJ`.

17 Sebastien Gagneux. Ecology and evolution of Mycobacterium tuberculosis. *Nat Rev Microbiol*, 16:202–213, 2018.

18 Li H1, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R, and 1000 Genome Project Data Processing Subgroup. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–2079, 2009.

19 Matthew A Herman and Thomas Strohmer. High-resolution radar via compressed sensing. *IEEE transactions on signal processing*, 57(6):2275–2284, 2009.

20 IBM. IBM ILOG CPLEX Optimization Studio V12.10.0 documentation. `https://www.ibm.com/support/knowledgecenter/SSSA5P`, 2020.

21 H. Iwai, M. Kato-Miyazawa, T Kirikae, and T. Miyoshi-Akiyama. CASTB (the comprehensive analysis server for the Mycobacterium tuberculosis complex): A publicly accessible web server for epidemiological analyses, drug-resistance prediction and phylogenetic comparison of clinical isolates. *Tuberculosis*, pages 843–844, 2015.

22 Suha Kadura, Nicholas King, Maria Nakhoul, Hongya Zhu, Grant Theron, Claudio U Köser, and Maha Farhat. Systematic review of mutations associated with resistance to the new and repurposed Mycobacterium tuberculosis drugs bedaquiline, clofazimine, linezolid, delamanid and pretomanid. *Journal of Antimicrobial Chemotherapy*, May 2020. dkaa136.

23 Rasko Leinonen, Ruth Akhtar, Ewan Birney, Lawrence Bower, Ana Cerdeno-Tárraga, et al. The European Nucleotide Archive. *Nucleic Acids Research*, 39:D28–31, 2011.

24 Rasko Leinonen, Hideaki Sugawara, Martin Shumway, and International Nucleotide Sequence Database Collaboration. The sequence read archive. *Nucleic acids research*, 39(suppl_1):D19–D21, 2010.

25 H Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv*, 2013.

26 Michael Lustig, David Donoho, and John M Pauly. Sparse MRI: The application of compressed sensing for rapid MR imaging. *Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine*, 58(6):1182–1195, 2007.

27 D. Malioutov and M. Malyutov. Boolean compressed sensing: LP relaxation for group testing. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3305–3308, 2012.

28 Dmitry Malioutov and Kush Varshney. Exact rule learning via Boolean compressed sensing. In *International Conference on Machine Learning*, pages 765–773, 2013.

29 L Mathelin and KA Gallivan. A compressed sensing approach for partial differential equations with random input data. *Communications in computational physics*, 12(4):919–954, 2012.

30 Arya Mazumdar. On almost disjunct matrices for group testing. In Kun-Mao Chao, Tsan-sheng Hsu, and Der-Tsai Lee, editors, *Algorithms and Computation*, pages 649–658, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

31 Paolo Miotto, Belay Tessema, Elisa Tagliani, Leonid Chindelevitch, et al. A standardised method for interpreting the association between mutations and phenotypic drug-resistance in *Mycobacterium tuberculosis*. *European Respiratory Journal*, 50(6), 2017.

32 W James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. Interpretable machine learning: definitions, methods, and applications. *arXiv*, 2019.

33 Balas Kausik Natarajan. Sparse approximate solutions to linear systems. *SIAM journal on computing*, 24(2):227–234, 1995.

34 Tra-My Ngo and Yik-Ying Teo. Genomic prediction of tuberculosis drug-resistance: benchmarking existing databases and prediction algorithms. *BMC Bioinformatics*, 20(1):68, 2019.

35 Jim O'Neill. Antimicrobial resistance: Tackling a crisis for the health and wealth of nations. Technical report, Review on Antimicrobial Resistance, 2014. URL: `https://amr-review.org/Publications.html`.

36 World Health Organization. Antimicrobial resistance: global report on surveillance. Technical report, WHO, 2014.

37 World Health Organization. Global tuberculosis report 2019. Technical report, WHO, 2019.

38 F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

39 Ryan Poplin, Valentin Ruano-Rubio, Mark A. DePristo, Tim J. Fennell, Mauricio O. Carneiro, Geraldine A. Van der Auwera, David E. Kling, Laura D. Gauthier, Ami Levy-Moonshine, David Roazen, Khalid Shakir, Joel Thibault, Sheila Chandran, Chris Whelan, Monkol Lek, Stacey Gabriel, Mark J Daly, Ben Neale, Daniel G. MacArthur, and Eric Banks. Scaling accurate genetic variant discovery to tens of thousands of samples. *bioRxiv*, 2017.

40 Mario C Raviglione and Ian M Smith. XDR tuberculosis—implications for global public health. *New England Journal of Medicine*, 356(7):656–659, 2007.

41 Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144, 2016.

42 James Emmanuel San, Shakuntala Baichoo, Aquillah Kanzi, Yumna Moosa, Richard Lessells, Vagner Fonseca, John Mogaka, Robert Power, and Tulio de Oliveira. Current affairs of microbial genome-wide association studies: Approaches, bottlenecks and analytical pitfalls. *Frontiers in Microbiology*, 10:3119, 2020. URL: `https://www.frontiersin.org/article/10.3389/fmicb.2019.03119`.

43 V. Schleusener, C. Köser, P. Beckert, et al. Mycobacterium tuberculosis resistance prediction and lineage classification from genome sequencing: comparison of automated analysis tools. *Scientific Reports*, 7, 2017.

44 Almeida Da Silva, Pedro Eduardo, Palomino, and Juan Carlos. Molecular basis and mechanisms of drug resistance in Mycobacterium tuberculosis: classical and new drugs. *Journal of Antimicrobial Chemotherapy*, 66(7):1417–1430, May 2011.

45 Angela M Starks, Enrique Avilés, Daniela M Cirillo, Claudia M Denkinger, David L Dolinger, Claudia Emerson, Jim Gallarda, Debra Hanna, Peter S Kim, Richard Liwski, et al. Collaborative effort for a centralized worldwide tuberculosis relational sequencing data platform. *Clinical Infectious Diseases*, 61(suppl_3):S141–S146, 2015.

46 A Steiner, D Stucki, M Coscolla, S Borrell, and S Gagneux. KvarQ: targeted and direct variant calling from fastq reads of bacterial genomes. *BMC Genomics*, 15, 2014.

47 Alice R Wattam, David Abraham, Oral Dalay, Terry L Disz, Timothy Driscoll, Joseph L Gabbard, Joseph J Gillespie, Roger Gough, Deborah Hix, Ronald Kenyon, et al. PATRIC, the bacterial bioinformatics database and analysis resource. *Nucleic acids research*, 42(D1):D581–D591, 2014.

48 Yang Yang, Katherine E Niehaus, Timothy M Walker, Zamin Iqbal, A Sarah Walker, Daniel J Wilson, Tim EA Peto, Derrick W Crook, E Grace Smith, Tingting Zhu, et al. Machine learning for classifying tuberculosis drug-resistance from DNA sequencing data. *Bioinformatics*, 34(10):1666–1671, 2018.

Natural Family-Free Genomic Distance

Diego P. Rubert
Faculdade de Computação, Universidade Federal de Mato Grosso do Sul, Campo Grande, Brazil
diego@facom.ufms.br

Fábio V. Martinez
Faculdade de Computação, Universidade Federal de Mato Grosso do Sul, Campo Grande, Brazil
fhvm@facom.ufms.br

Marília D. V. Braga[1]
Faculty of Technology and Center for Biotechnology (CeBiTec), Bielefeld University, Germany
mbraga@cebitec.uni-bielefeld.de

Abstract

A classical problem in comparative genomics is to compute the rearrangement distance, that is the minimum number of large-scale rearrangements required to transform a given genome into another given genome. While the most traditional approaches in this area are *family-based*, i.e., require the classification of DNA fragments of both genomes into *families*, more recently an alternative model was proposed, which, instead of family classification, simply uses the *pairwise similarities* between DNA fragments of both genomes to compute their rearrangement distance. This model represents structural rearrangements by the generic *double cut and join* (DCJ) operation and is then called *family-free DCJ distance*. It computes the DCJ distance between the two genomes by searching for a matching of their genes based on the given pairwise similarities, therefore helping to find gene homologies. The drawback is that its computation is NP-hard. Another point is that the family-free DCJ distance must correspond to a maximal matching of the genes, due to the fact that unmatched genes are just ignored: maximizing the matching prevents the *free lunch* artifact of having empty or almost empty matchings giving the smaller distances.

In this paper, besides DCJ operations, we allow content-modifying operations of *insertions* and *deletions* of DNA segments and propose a new and more general family-free genomic distance. In our model we use the pairwise similarities to assign weights to both matched and unmatched genes, so that an optimal solution does not necessarily maximize the matching. Our model then results in a *natural family-free genomic distance*, that takes into consideration all given genes and has a search space composed of matchings of any size. We provide an efficient ILP formulation to solve it, by extending the previous formulations for computing family-based genomic distances from Shao et al. (*J. Comput. Biol.*, 2015) and Bohnenkämper et al. (*Proc. of RECOMB*, 2020). Our experiments show that the ILP can handle not only bacterial genomes, but also fungi and insects, or sets of chromosomes of mammals and plants. In a comparison study of six fruit fly genomes, we obtained accurate results.

2012 ACM Subject Classification Theory of computation → Linear programming

Keywords and phrases Comparative genomics, Genome rearrangement, DCJ-indel distance

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.3

Related Version `http://arxiv.org/abs/2007.03556`

Supplementary Material The ILP for computing the natural family-free DCJ-indel distance can be downloaded from our GitLab server at `https://gitlab.ub.uni-bielefeld.de/gi/gen-diff`.

Acknowledgements We thank the anonymous reviewers for their valuable comments.

[1] Corresponding author

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti; Article No. 3; pp. 3:1–3:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Genomes are subject to mutations or *rearrangements* in the course of evolution. A classical problem in comparative genomics is to compute the rearrangement *distance*, that is the minimum number of large-scale rearrangements required to transform a given genome into another given genome [21]. Typical large-scale rearrangements change the number of chromosomes, and/or the positions and orientations of DNA segments. Examples of such *structural* rearrangements are inversions, translocations, fusions and fissions. One might also need to consider rearrangements that modify the content of a genome, such as insertions and deletions (collectively called *indels*) of DNA segments.

In order to study the rearrangement distance, one usually adopts a high-level view of genomes, in which only "relevant" fragments of the DNA (e.g., genes) are taken into consideration. Furthermore, a pre-processing of the data is required, so that we can compare the content of the genomes. One popular method, adopted for more than 20 years, is to group the fragments in both genomes into *families*, so that two fragments in the same family are said to be equivalent. This setting is said to be *family-based.* Without duplications, that is, with the additional restriction that each family occurs at most once in each genome, many polynomial models have been proposed to compute the genomic distance [3, 6, 13, 24, 25]. However, when duplications are allowed the problem is more intricate and all approaches proposed so far are NP-hard, see for instance [2, 7, 8, 18, 22, 23].

The required pre-classification of DNA fragments into families is a drawback of the family-based approaches. Moreover, even with a careful pre-processing, it is not always possible to classify each fragment unambiguously into a single family. Due to these facts, an alternative to the family-based setting was proposed and consists in studying the rearrangement distance without prior family assignment. Instead of families, the *pairwise similarities* between fragments is directly used [5, 12]. By letting structural rearrangements be represented by the generic *double cut and join* (DCJ) operation [24], a first family-free genomic distance, called family-free DCJ distance, was already proposed [16]. Its computation helps to match occurrences of duplicated genes and find homologies, but unmatched genes are simply ignored.

In the family-based setting, the mentioned approaches that handle duplications either require the compared genomes to be *balanced* (that is, have the same number of occurrences of each family) [18, 23] or adopt some approach to match genes, ignoring unmatched genes [8, 22]. Recently, a new family-based approach was proposed, allowing each family to occur any number of times in each genome and integrating DCJ operations and indels in a *DCJ-indel* distance formula [4]. For its computation, that is NP-hard, an efficient ILP was proposed.

Here we adapt the approach mentioned above and give an ILP formulation to compute a new family-free DCJ-indel distance. In the family-based approach from [4] as well as in the family-free DCJ distance proposed in [16], the search space needs to be restricted to candidates that maximize the number of matched genes, in order to avoid the *free lunch* artifact of having empty or almost empty matchings giving the smaller distances [25]. In our formulation we use the pairwise similarities to assign weights to matched and unmatched genes, so that, for the first time, an optimal solution does not necessarily maximize the number of matched genes. For the fact that our model takes into consideration all given genes and has a search space composed of matchings of any size, we call it *natural family-free genomic distance.* Our simulated experiments show that our ILP can handle not only bacterial genomes, but also complete genomes of fungi and insects, or sets of chromosomes of mammals and plants. We use our implementation to generate pairwise distances and reconstruct the phylogeny of six species of fruit flies from the genus *Drosophila*, obtaining accurate results.

This paper is organized as follows. In Section 2 we give some basic definitions and previous results that are essential for the approach presented here. In Section 3 we define the new *natural* family-free DCJ-indel distance. In Section 4 we describe the optimization approach for computing the family-free DCJ-indel distance with the help of the *family-free relational diagram*. In Section 5 we present the ILP formulation and the experimental results. Finally, Section 6 concludes the text.

2 Preliminaries

We call *marker* an oriented DNA fragment. A *chromosome* is composed of markers and can be linear or circular. A marker m in a chromosome can be represented by the symbol m itself, if it is read in direct orientation, or the symbol $\overline{m}$, if it is read in reverse orientation. We concatenate all markers of a chromosome Z in a string z, which can be read in any of the two directions. If Z is circular, we can start to read it at any marker and the string z is flanked by parentheses. A set of chromosomes comprises a *genome*. As an example, let $A = \{\overline{6}178\overline{4}, 3\overline{5}2\}$ be a genome composed of two linear chromosomes. A genome can be transformed or *sorted* into another genome with the following types of mutations.

1. **DCJ operations modify the organization of a genome:** A *cut* performed on a genome A separates two adjacent markers of A. A *double-cut and join* or *DCJ* applied on a genome A is the operation that performs cuts in two different positions of A, creating four open ends, and joins these open ends in a different way [3, 24]. For example, let $A = \{\overline{6}178\overline{4}, 3\overline{5}2\}$, and consider a DCJ that cuts between markers 1 and 7 of its first chromosome and between markers 5 and 2 of its second chromosome, creating fragments $\overline{6}1\bullet$, $\bullet 78\overline{4}$, $3\overline{5}\bullet$ and $\bullet 2$ (where the symbols $\bullet$ represent the open ends). If we join the first with the fourth and the third with the second open end, we get $A' = \{\overline{6}12, 3\overline{5}78\overline{4}\}$, that is, the described DCJ operation is a translocation transforming A into A'. Indeed, a DCJ operation can correspond not only to a translocation but to several structural rearrangements, such as an inversion, a fusion or a fission.

2. **Indel operations modify the content of a genome:** We can modify the content of a genome with *insertions* and with *deletions* of blocks of contiguous markers, collectively called *indel* operations [6, 25]. As an example, consider the deletion of fragment 78 from chromosome $\overline{6}178\overline{4}$, resulting in chromosome $\overline{6}1\overline{4}$. In the model we consider, we do not allow that a marker is deleted and reinserted, nor inserted and then deleted. Furthermore, at most one chromosome can be entirely deleted or inserted at once.

Let A and B be two genomes and let $\mathcal{A}$ be the set of markers in genome A and $\mathcal{B}$ be the set of markers in genome B. We consider two distinct settings:

- In a *family-based setting* markers are grouped into families and each marker from a genome is represented by its family. Therefore, a marker from $\mathcal{A}$ can occur more than once in A, as well as a marker from $\mathcal{B}$ can occur more than once in B. Furthermore, genomes A and B may share a set of *common markers* $\mathcal{G} = \mathcal{A} \cap \mathcal{B}$. We also have sets $\mathcal{A}_\star = \mathcal{A} \setminus \mathcal{G}$ and $\mathcal{B}_\star = \mathcal{B} \setminus \mathcal{G}$ of markers that occur respectively only in A and only in B and are called *exclusive markers*. For example, consider genomes $A = \{\overline{3}143\overline{2}, 3\overline{5}2\}$ and $B = \{\overline{1}2\overline{1}3\overline{2}6\}$. In this case we have $\mathcal{A} = \{1, 2, 3, 4, 5\}$ and $\mathcal{B} = \{1, 2, 3, 6\}$. Consequently, $\mathcal{G} = \{1, 2, 3\}$, $\mathcal{A}_\star = \{4, 5\}$ and $\mathcal{B}_\star = \{6\}$.
- In a *family-free setting* the markers of A and B are all distinct and unique. In other words, each marker of $\mathcal{A}$ occurs exactly once in A, each marker of $\mathcal{B}$ occurs exactly once in B and $\mathcal{A} \cap \mathcal{B} = \emptyset$. Consider, for example, genomes $A = \{\overline{1}3\overline{4}2\}$ and $B = \{\overline{8}7\overline{5}, 9\overline{6}\}$.

2.1 Relational diagram and distance of family-based singular genomes

Let A and B be two genomes in a family-based setting and assume that both A and B are *singular*, that is, each marker from $\mathcal{G} = \mathcal{A} \cap \mathcal{B}$ occurs exactly once in each genome. We will now describe how the DCJ-indel distance can be computed in this case [6].

For a given marker m, denote its two extremities by m^t (tail) and m^h (head). Given two singular genomes A and B, the *relational diagram* $R(A, B)$ [4] has a set of vertices $V = V(A) \cup V(B)$, where $V(A)$ is the set of extremities of markers from A and $V(B)$ is the set of extremities of markers from B. There are three types of edges in $R(A, B)$:

- *Adjacency edges*: for each pair of marker extremities γ_1 and γ_2 that are adjacent in a chromosome of any of the two genomes, we have the adjacency edge $\gamma_1\gamma_2$. Denote by E^A_{adj} and by E^B_{adj} the adjacency edges in A and in B, respectively. Marker extremities located at chromosome ends are called *telomeres* and are not connected to any adjacency edge.
- *Extremity edges*, whose set is denoted by E_γ: for each common marker $m \in \mathcal{G}$, we have two extremity edges, one connecting the vertex m^h from $V(A)$ to the vertex m^h from $V(B)$ and the other connecting the vertex m^t from $V(A)$ to the vertex m^t from $V(B)$.
- *Indel edges*: for each occurrence of an exclusive marker $m \in \mathcal{A}_\star \cup \mathcal{B}_\star$, we have the indel edge $m^t m^h$. Denote by E^A_{id} and by E^B_{id} the indel edges in A and in B.

Each vertex has degree one or two: it is connected either to an extremity edge or to an indel edge, and to at most one adjacency edge, therefore $R(A, B)$ is a simple collection of cycles and paths. A path that has one endpoint in genome A and the other in genome B is called an *AB-path*. In the same way, both endpoints of an *AA-path* are in A and both endpoints of a *BB-path* are in B. A cycle contains either zero or an even number of extremity edges. When a cycle has at least two extremity edges, it is called an *AB-cycle*. Moreover, a path (respectively cycle) of $R(A, B)$ composed exclusively of indel and adjacency edges in one of the two genomes corresponds to a whole linear (respectively circular) chromosome and is called a *linear* (respectively *circular*) *singleton* in that genome. Actually, linear singletons are particular cases of *AA*- or *BB*-paths. The numbers of telomeres and of *AB*-paths in $R(A, B)$ are even. An example of a relational diagram is given in Figure 1.

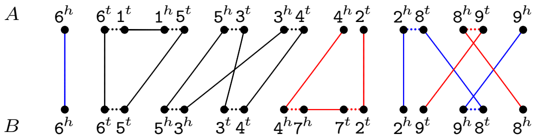


Figure 1 For genomes $A = \{\overline{6}1534, 289\}$ and $B = \{\overline{6}5\overline{3}4\overline{7}2, 98\}$, the relational diagram contains two cycles, two *AB*-paths (represented in blue), one *AA*-path and one *BB*-path (both represented in red). Short dotted horizontal edges are adjacency edges, long horizontal edges are indel edges, top-down edges are extremity edges.

DCJ distance of canonical genomes. When singular genomes A and B have no exclusive markers, that is, $\mathcal{A}_\star = \mathcal{B}_\star = \emptyset$, they are said to be *canonical*. In this case A can be sorted into B with DCJ operations only and their DCJ distance d_{DCJ} can be computed as follows [3]:

$$\text{d}_{\text{DCJ}}(A, B) \;=\; |\mathcal{G}| - c - \frac{i}{2}\,,$$

where c is the number of *AB*-cycles and i is the number of *AB*-paths in $R(A, B)$.

Runs and indel-potential. When singular genomes A and B have exclusive markers, it is possible to optimally select DCJ operations that group exclusive markers together for minimizing indels [6], as follows.

Given two genomes A and B and a component C of $R(A, B)$, a *run* [6] is a maximal subpath of C, in which the first and the last edges are indel edges, and all indel edges belong to the same genome. It can be an $\mathcal{A}$-run when its indel edges are in genome A, or a $\mathcal{B}$-run when its indel edges are in genome B. We denote by $\Lambda(C)$ the number of runs in component C. If $\Lambda(C) \geq 1$ the component C is said to be *indel-enclosing*, otherwise $\Lambda(C) = 0$ and C is said to be *indel-free*. The *indel-potential* of a component C, denoted by $\lambda(C)$, is the optimal number of indels obtained after "sorting" C separately and can be directly computed from $\Lambda(C)$ [6]:

$$\lambda(C) = \begin{cases} 0\,, & \text{if } \Lambda(C) = 0 \quad (C \text{ is indel-free}); \\ \left\lceil \frac{\Lambda(C)+1}{2} \right\rceil\,, & \text{if } \Lambda(C) \geq 1 \quad (C \text{ is indel-enclosing}). \end{cases}$$

With the indel-potential, an upper bound for the DCJ-indel distance $\mathrm{d}_{\mathrm{DCJ}}^{\mathrm{ID}}$ was established [6]:

$$\mathrm{d}_{\mathrm{DCJ}}^{\mathrm{ID}}(A, B) \;\leq\; |\mathcal{G}| - c - \frac{i}{2} \;+ \sum_{C \in R(A,B)} \lambda(C) \tag{1}$$

DCJ-indel distance of singular circular genomes. For singular circular genomes, the graph $R(A, B)$ is composed of cycles only. In this case the upper bound given by Equation (1) is tight and leads to a simplified formula [6]:

$$\mathrm{d}_{\mathrm{DCJ}}^{\mathrm{ID}}(A, B) \;=\; |\mathcal{G}| - c \;+ \sum_{C \in R(A,B)} \lambda(C)\,.$$

DCJ-indel distance of singular linear genomes. For singular linear genomes, the upper bound given by Equation (1) is achieved when the components of $R(A, B)$ are sorted separately. However, it can be decreased by *recombinations*, that are DCJ operations that act on two distinct paths of $R(A, B)$. Such path recombinations are said to be *deducting*. The total number of types of deducting recombinations is relatively small. By exhaustively exploring the space of recombination types, it is possible to identify groups of chained recombinations [6], so that the sources of each group are the original paths of the graph. In other words, a path that is a resultant of a group is never a source of another group. This results in a greedy approach (detailed in [6]) that optimally finds the value $\delta \geq 0$ to be deducted. We then have the following exact formula [6]:

$$\mathrm{d}_{\mathrm{DCJ}}^{\mathrm{ID}}(A, B) \;=\; |\mathcal{G}| - c - \frac{i}{2} \;+ \sum_{C \in R(A,B)} \lambda(C) \;\; - \delta\,.$$

3 The family-free setting

As already stated, in the family-free setting, each marker in each genome is represented by a distinct symbol, thus $\mathcal{A} \cap \mathcal{B} = \emptyset$. Observe that the cardinalities $|\mathcal{A}|$ and $|\mathcal{B}|$ may be distinct.

3.1 Marker similarity graph for the family-free setting

Given a threshold $0 \leq x \leq 1$, we can represent the similarities between the markers of genome A and the markers of genome B in the so called *marker similarity graph* [5], denoted by

$\mathcal{S}_x(A,B)$. This is a weighted bipartite graph whose partitions $\mathcal{A}$ and $\mathcal{B}$ are the sets of markers in genomes A and B, respectively. Furthermore, for each pair of markers $a \in \mathcal{A}$ and $b \in \mathcal{B}$, denote by $\sigma(a,b)$ their *normalized similarity*, a value that ranges in the interval $[0,1]$. If $\sigma(a,b) \geq x$ there is an edge e connecting a and b in $\mathcal{S}_x(A,B)$ whose weight is $\sigma(e) := \sigma(a,b)$. An example is given in Figure 2.

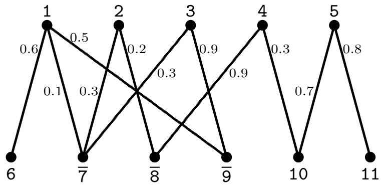


Figure 2 Graph $\mathcal{S}_{0.1}(A,B)$ for the two genomes $A = \{1\ 2\ 3\ 4\ 5\}$ and $B = \{6\ \overline{7}\ \overline{8}\ \overline{9}\ 10\ 11\}$.

Mapped genomes. Let A and B be two genomes with marker similarity graph $\mathcal{S}_x(A,B)$ and let $M = \{e_1, e_2, \ldots, e_n\}$ be a matching in $\mathcal{S}_x(A,B)$. Since the endpoints of each edge $e_i = (a,b)$ in M are not saturated by any other edge of M, we can unambiguously define the function $s(a,M) = s(b,M) = i$. We then define the set of *M-saturated mapped markers* $\mathcal{G}(M) = \{s(g,M) \colon g \text{ is } M\text{-saturated } \} = \{1,2,\ldots,n\}$.

Let $\tilde{n}_A$ be the number of unsaturated markers in $\mathcal{A}$ and $\tilde{n}_B$ be the number of unsaturated markers in $\mathcal{B}$. We extend the function s, so that it maps each unsaturated marker $a' \in \mathcal{A}$ to one value in $\{n+1, n+2, \ldots, n+\tilde{n}_A\}$ and each unsaturated marker $b' \in \mathcal{B}$ to one value in $\{n+\tilde{n}_A+1, n+\tilde{n}_A+2, \ldots, n+\tilde{n}_A+\tilde{n}_B\}$. The sets of *$M$-unsaturated mapped markers* are:

- $\mathcal{A}_\star(M) = \{s(a',M) \colon a' \in \mathcal{A} \text{ is } M\text{-unsaturated } \} = \{n+1, n+2, \ldots, n+\tilde{n}_A\}$ and
- $\mathcal{B}_\star(M) = \{s(b',M) \colon b' \in \mathcal{B} \text{ is } M\text{-unsaturated } \} = \{n+\tilde{n}_A+1, n+\tilde{n}_A+2, \ldots, n+\tilde{n}_A+\tilde{n}_B\}$.

The *mapped genomes* A^M and B^M are then obtained by renaming each marker $a \in \mathcal{A}$ to $s(a,M)$ and each marker $b \in \mathcal{B}$ to $s(b,M)$, preserving all orientations.

Established distances of mapped genomes. Let the relational graph $R(A^M,B^M)$ have c_M AB-cycles and i_M AB-paths. By simply ignoring the exclusive markers of $\mathcal{A}_\star(M)$ and $\mathcal{B}_\star(M)$, we can compute the DCJ distance:

$$\mathrm{d}_{\mathrm{DCJ}}(A^M,B^M) = |M| - c_M - \frac{i_M}{2}\,.$$

Taking into consideration the weight of the matching M defined as $w(M) = \sum_{e \in M} \sigma(e)$, we can also compute the weighted DCJ distance $\mathrm{wd}_{\mathrm{DCJ}}(A^M,B^M)$ [16]:

$$\mathrm{wd}_{\mathrm{DCJ}}(A^M,B^M) = \mathrm{d}_{\mathrm{DCJ}}(A^M,B^M) + |M| - w(M)\,.$$

Observe that, when all edges of M have the maximum weight 1, we have $w(M) = |M|$ and $\mathrm{wd}_{\mathrm{DCJ}}(A^M,B^M) = \mathrm{d}_{\mathrm{DCJ}}(A^M,B^M)$.

Finally, taking into consideration the exclusive markers of $\mathcal{A}_\star(M)$ and $\mathcal{B}_\star(M)$, but not the weight $w(M)$, we can compute the DCJ-indel distance of mapped genomes A^M and B^M:

$$\mathrm{d}_{\mathrm{DCJ}}^{\mathrm{ID}}(A^M,B^M) = |M| - c_M - \frac{i_M}{2} + \sum_{C \in R(A^M,B^M)} \lambda(C) \quad - \delta_M\,,$$

where δ_M is the deduction given by path recombinations in $R(A^M,B^M)$.

3.2 The family-free DCJ-indel distance

Let A^M and B^M be the mapped genomes for a given matching M of $\mathcal{S}_x(A,B)$. The *weighted relational diagram* of A^M and B^M, denoted by $WR(A^M, B^M)$, is obtained by constructing the relational diagram of A^M and B^M and adding weights to the indel edges as follows. For each mapped M-unsaturated marker $m \in \mathcal{A}_\star(M) \cup \mathcal{B}_\star(M)$, the indel edge $m^h m^t$ receives a weight $w(m^h m^t) = \max\{\sigma(uv) | uv \in \mathcal{S}_x(A,B) \text{ and } u{=}s^{-1}(m,M)\}$, that is the maximum similarity among the edges incident to the marker $u = s^{-1}(m,M)$ in $\mathcal{S}_x(A,B)$. We denote by $\widetilde{M} = E^A_{\text{id}} \cup E^B_{\text{id}}$ the set of indel edges, here also called the *complement* of M. The weight of $\widetilde{M}$ is $w(\widetilde{M}) = \sum_{e \in \widetilde{M}} w(e)$. Examples of diagrams of mapped genomes are shown in Figure 3.

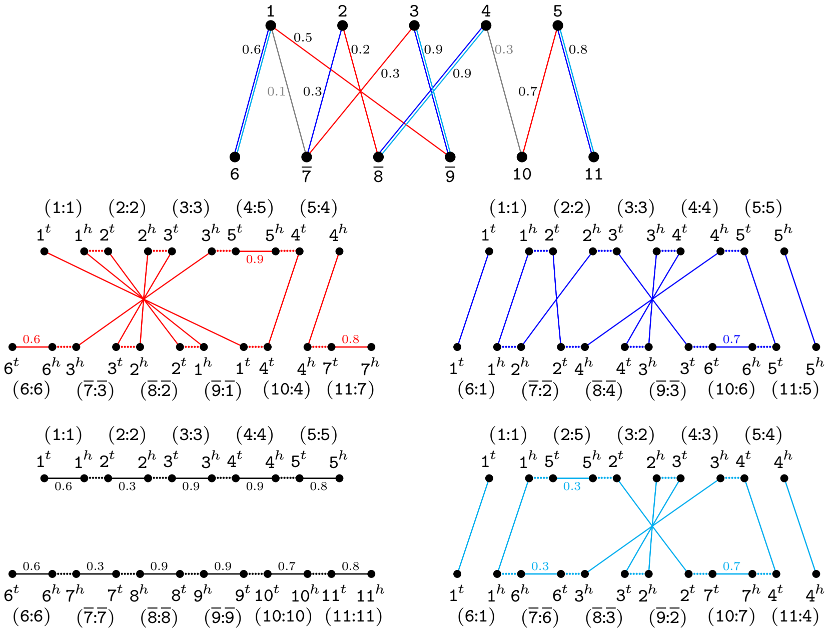

1
2
3
4
5
0.6
0.5
0.2
0.9
0.3
0.9
0.1
0.3
0.3
0.7
0.8
6
$\overline{7}$
$\overline{8}$
$\overline{9}$
10
11
(1:1) (2:2) (3:3) (4:5) (5:4)
1^t 1^h 2^t 2^h 3^t 3^h 5^t 5^h 4^t 4^h
0.9
0.6
0.8
6^t 6^h 3^h 3^t 2^h 2^t 1^h 1^t 4^t 4^h 7^t 7^h
(6:6) ($\overline{7}$:$\overline{3}$) ($\overline{8}$:$\overline{2}$) ($\overline{9}$:$\overline{1}$) (10:4) (11:7)
(1:1) (2:2) (3:3) (4:4) (5:5)
1^t 1^h 2^t 2^h 3^t 3^h 4^t 4^h 5^t 5^h
0.7
1^t 1^h 2^h 2^t 4^h 4^t 3^h 3^t 6^t 6^h 5^t 5^h
(6:1) ($\overline{7}$:$\overline{2}$) ($\overline{8}$:$\overline{4}$) ($\overline{9}$:$\overline{3}$) (10:6) (11:5)
(1:1) (2:2) (3:3) (4:4) (5:5)
1^t 1^h 2^t 2^h 3^t 3^h 4^t 4^h 5^t 5^h
0.6 0.3 0.9 0.9 0.8
0.6 0.3 0.9 0.9 0.7 0.8
6^t 6^h 7^h 7^t 8^h 8^t 9^h 9^t 10^t 10^h 11^t 11^h
(6:6) ($\overline{7}$:$\overline{7}$) ($\overline{8}$:$\overline{8}$) ($\overline{9}$:$\overline{9}$) (10:10) (11:11)
(1:1) (2:5) (3:2) (4:3) (5:4)
1^t 1^h 5^t 5^h 2^t 2^h 3^t 3^h 4^t 4^h
0.3
0.3
0.7
1^t 1^h 6^h 6^t 3^h 3^t 2^h 2^t 7^t 7^h 4^t 4^h
(6:1) ($\overline{7}$:$\overline{6}$) ($\overline{8}$:3) ($\overline{9}$:$\overline{2}$) (10:7) (11:4)

Figure 3 Considering the same genomes $A = \{1\,2\,3\,4\,5\}$ and $B = \{6\,\overline{7}\,\overline{8}\,\overline{9}\,10\,11\}$ as in Figure 2, let M_1 (red) and M_2 (blue) be two distinct maximal matchings in $\mathcal{S}_{0.1}(A,B)$. We also represent the non-maximal matching M_3 (cyan) that is a subset of M_2. In the middle part we show diagrams $WR(A^{M_1}, B^{M_1})$ and $WR(A^{M_2}, B^{M_2})$, both with two *AB*-paths and two *AB*-cycles. In the lower part we show diagrams $WR(A^{M_\emptyset}, B^{M_\emptyset})$, corresponding to the trivial empty matching $M_\emptyset$ and with two linear singletons (one *AA*-path and one *BB*-path), and $WR(A^{M_3}, B^{M_3})$, with two *AB*-paths and two *AB*-cycles. The labeling $(\mathtt{X}:\mathtt{Y})$ indicates that $\mathtt{Y} = s(\mathtt{X}, M_i)$.

In the computation of the weighted DCJ-indel distance of mapped genomes A^M and B^M, denoted by $\text{wd}^{\text{ID}}_{\text{DCJ}}(A^M, B^M)$, we should take into consideration the exclusive markers of $\mathcal{A}_\star(M)$ and $\mathcal{B}_\star(M)$, and the weights $w(M)$ and $w(\widetilde{M})$. An important condition is that $\text{wd}^{\text{ID}}_{\text{DCJ}}(A^M, B^M)$ must be equal to $\text{d}^{\text{ID}}_{\text{DCJ}}(A^M, B^M)$ if $w(M) = |M|$ and $w(\widetilde{M}) = 0$. We can achieve this by extending the formula for computing $\text{wd}_{\text{DCJ}}(A^M, B^M)$ as follows:

$$\begin{aligned}
\text{wd}^{\text{ID}}_{\text{DCJ}}(A^M, B^M) &= \text{wd}_{\text{DCJ}}(A^M, B^M) + \sum_{C \in WR(A^M, B^M)} \lambda(C) \quad - \delta_M + w(\widetilde{M}) \\
&= \text{d}_{\text{DCJ}}(A^M, B^M) + |M| - w(M) + \sum_{C \in WR(A^M, B^M)} \lambda(C) \quad - \delta_M + w(\widetilde{M}) \\
&= \text{d}^{\text{ID}}_{\text{DCJ}}(A^M, B^M) + |M| - w(M) + w(\widetilde{M})\,.
\end{aligned}$$

Let us now examine the behaviour of the formula above for the examples given in Figure 3. Matching M_1 is maximal and gives the distance $\mathrm{wd}_{\mathrm{DCJ}}^{\mathrm{ID}}(A^{M_1}, B^{M_1}) = 8.6$. Matching M_2 is also maximal and gives the distance $\mathrm{wd}_{\mathrm{DCJ}}^{\mathrm{ID}}(A^{M_2}, B^{M_2}) = 5.2$. The empty matching $M_\emptyset$ gives the distance $\mathrm{wd}_{\mathrm{DCJ}}^{\mathrm{ID}}(A^{M_\emptyset}, B^{M_\emptyset}) = 9.7$, that is the biggest. And the non-maximal matching $M_3 \subset M_2$ gives the distance $\mathrm{wd}_{\mathrm{DCJ}}^{\mathrm{ID}}(A^{M_3}, B^{M_3}) = 5.1$, that is the smallest.

Given that $\mathbb{M}$ is the set of all distinct matchings in $\mathcal{S}_x(A, B)$, the family-free DCJ-indel distance is defined as follows:

$$\mathrm{ffd}_{\mathrm{DCJ}}^{\mathrm{ID}}(A, B, \mathcal{S}_x) = \min_{M \in \mathbb{M}} \{\mathrm{wd}_{\mathrm{DCJ}}^{\mathrm{ID}}(A^M, B^M)\} \,.$$

Complexity. If two family-based genomes contain the same number of occurrences of each marker, they are said to be *balanced*. The problem of computing the DCJ distance of balanced genomes (BG-DCJ) is NP-hard [23]. Since the computation of $\mathrm{ffd}_{\mathrm{DCJ}}^{\mathrm{ID}}$ can be used to solve BG-DCJ, it is also NP-hard. The details of the reduction can be found in [19].

4 Family-free relational diagram

An efficient way to solve the family-free DCJ-indel distance is to develop an ILP that searches for its solution in a general graph, that represents all possible diagrams corresponding to all candidate matchings, in a similar way as the approaches given in [4, 16, 23]. Given two genomes A and B and their marker similarity graph $\mathcal{S}_x(A, B)$, the structure that integrates the properties of all diagrams of mapped genomes is the *family-free relational diagram* $\mathit{FFR}(A, B, \mathcal{S}_x)$, that has a set $V(A)$ with a vertex for each of the two extremities of each marker of genome A and a set $V(B)$ with a vertex for each of the two extremities of each marker of genome B.

Again, sets E_{adj}^A and E_{adj}^B contain adjacency edges connecting adjacent extremities of markers in A and in B. But here the set E_γ contains, for each edge $ab \in \mathcal{S}_x(A, B)$, an extremity edge connecting a^t to b^t, and an extremity edge connecting a^h to b^h. To both edges a^tb^t and a^hb^h, that are called *siblings*, we assign the same weight, that corresponds to the similarity of the edge ab in $\mathcal{S}_x(A, B)$: $w(a^tb^t) = w(a^hb^h) = \sigma(ab)$. Furthermore, for each marker m there is an indel edge connecting the vertices m^h and m^t. The indel edge m^hm^t receives a weight $w(m^hm^t) = \max\{\sigma(mv) | mv \in \mathcal{S}_x(A, B)\}$, that is, it is the maximum similarity among the edges incident to the marker m in $\mathcal{S}_x(A, B)$. We denote by E_{id}^A the set of indel edges of markers in genome A and by E_{id}^B the set of indel edges of markers in genome B. An example of a family-free relational diagram is given in Figure 4.

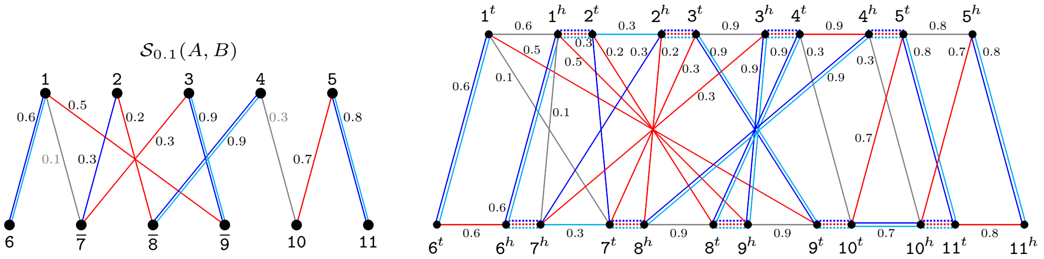


Figure 4 Given genomes $A = \{1\,2\,3\,4\,5\}$ and $B = \{6\,\overline{7}\,\overline{8}\,\overline{9}\,10\,11\}$, in the left part we represent the marker similarity graph $\mathcal{S}_{0.1}(A, B)$ and in the right part the family-free relational diagram $\mathit{FFR}(A, B, \mathcal{S}_{0.1})$. We represent in multiple colors the edges that correspond to multiple matchings.

4.1 Consistent decompositions

The diagram $\mathit{FFR}(A, B, \mathcal{S}_x)$ may contain vertices of degree larger than two. A *decomposition* of $\mathit{FFR}(A, B, \mathcal{S}_x)$ is a collection of vertex-disjoint *components*, that can be cycles and/or paths, covering all vertices of $\mathit{FFR}(A, B, \mathcal{S}_x)$. There can be multiple ways of selecting a decomposition, and we need to find one that allows to identify a matching of $\mathcal{S}_x(A, B)$. A set $S \subseteq E_\gamma$ is a *sibling-set* if it is exclusively composed of pairs of siblings and does not contain any pair of incident edges. Thus, a sibling-set S of $\mathit{FFR}(A, B, \mathcal{S}_x)$ corresponds to a matching of $\mathcal{S}_x(A, B)$. In other words, there is a clear bijection between matchings of $\mathcal{S}_x(A, B)$ and sibling-sets of $\mathit{FFR}(A, B, \mathcal{S}_x)$ and we denote by M_S the matching corresponding to the sibling-set S.

The set of edges $D[S]$ *induced* by a sibling-set S is said to be a *consistent decomposition* of $\mathit{FFR}(A, B, \mathcal{S}_x)$ and can be obtained as follows. In the beginning, $D[S]$ is the union of S with the sets of adjacency edges E^A_{adj} and E^B_{adj}. We then need to determine the *complement* of the sibling-set S, denoted by $\widetilde{S}$, that is composed of the indel-edges of $\mathit{FFR}(A, B, \mathcal{S}_x)$ that must be added to $D[S]$: for each indel edge e, if its two endpoints have degree one or zero in $D[S]$, then e is added to both $\widetilde{S}$ and $D[S]$. (Note that $\widetilde{S} = \widetilde{M_S}$, while $|S| = 2|M_S|$ and $w(S) = 2w(M_S)$.) The consistent decomposition $D[S]$ covers all vertices of $\mathit{FFR}(A, B, \mathcal{S}_x)$ and is composed of cycles and paths, allowing us to compute the values

$$\mathrm{d}^{\text{ID}}_{\text{DCJ}}(D[S]) = \frac{|S|}{2} - c_D - \frac{i_D}{2} + \sum_{C \in D[S]} \lambda(C) - \delta_D \quad \text{and}$$

$$\mathrm{wd}^{\text{ID}}_{\text{DCJ}}(D[S]) = \mathrm{d}^{\text{ID}}_{\text{DCJ}}(D[S]) + \frac{|S|}{2} - \frac{w(S)}{2} + w(\widetilde{S})\,,$$

where c_D and i_D are the numbers of *AB*-cycles and *AB*-paths in $D[S]$, respectively, and δ_D is the optimal deduction of recombinations of paths from $D[S]$.

Given that $\mathbb{S}$ is the sets of all sibling-sets of $\mathit{FFR}(A, B, \mathcal{S}_x)$, we compute the family-free DCJ-indel distance of A and B with the following equation:

$$\mathrm{ffd}^{\text{ID}}_{\text{DCJ}}(A, B, \mathcal{S}_x) = \min_{S \in \mathbb{S}} \{\mathrm{wd}^{\text{ID}}_{\text{DCJ}}(D[S])\}\,.$$

4.2 Capping

Telomeres produce some difficulties for the decomposition of $\mathit{FFR}(A, B, \mathcal{S}_x)$, and a known technique to overcome this problem is called *capping* [13]. It consists of modifying the diagram by adding *artificial* markers, also called *caps*, whose extremities should be properly connected to the telomeres of the linear chromosomes of A and B. Therefore, usually the capping depends on the numbers κ_A and κ_B, that are, respectively, the total numbers of linear chromosomes in genomes A and B.

Family-based singular genomes. First we recall the capping of family-based singular genomes. Here the caps must circularize all linear chromosomes, so that their relational diagram is composed of cycles only, but, if the capping is optimal, the DCJ-indel distance is preserved.

An optimal capping that transforms singular linear genomes A and B into singular circular genomes can be obtained after identifying the recombination groups [6]. The DCJ-indel distance is preserved by properly linking the components of each identified recombination group into a single cycle [4]. Such a capping may require some artificial adjacencies between caps. The following result is very useful.

▶ **Theorem 1** (from [4]). *We can obtain an optimal capping of singular genomes A and B with exactly $p_* = \max\{\kappa_A, \kappa_B\}$ caps and $|\kappa_A - \kappa_B|$ artificial adjacencies between caps.*

Capped family-free relational diagram. We transform $FFR(A, B, \mathcal{S}_x)$ into the *capped family-free relational diagram* $FFR_\circ(A, B, \mathcal{S}_x)$ as follows. Again, let $p_* = \max\{\kappa_A, \kappa_B\}$. The diagram $FFR_\circ(A, B, \mathcal{S}_x)$ is obtained by adding to $FFR(A, B, \mathcal{S}_x)$ $4p_*$ new vertices, named $\circ_A^1, \circ_A^2, \ldots, \circ_A^{2p_*}$ and $\circ_B^1, \circ_B^2, \ldots, \circ_B^{2p_*}$, each one representing a *cap extremity.* Each of the $2\kappa_A$ telomeres of A is connected by an adjacency edge to a distinct cap extremity among $\circ_A^1, \circ_A^2, \ldots, \circ_A^{2\kappa_A}$. Similarly, each of the $2\kappa_B$ telomeres of B is connected by an adjacency edge to a distinct cap extremity among $\circ_B^1, \circ_B^2, \ldots, \circ_B^{2\kappa_B}$. Moreover, if $\kappa_A < \kappa_B$, for $i = 2\kappa_A + 1, 2\kappa_A + 3, \ldots, 2\kappa_B - 1$, connect $\circ_A^i$ to $\circ_A^{i+1}$ by an *artificial adjacency edge.* Otherwise, if $\kappa_B < \kappa_A$, for $j = 2\kappa_B + 1, 2\kappa_B + 3, \ldots, 2\kappa_A - 1$, connect $\circ_B^j$ to $\circ_B^{j+1}$ by an artificial adjacency edge. All these new adjacency edges and artificial adjacency edges are added to E_{adj}^A and E_{adj}^B, respectively. We also connect each $\circ_A^i$, $1 \leq i \leq 2p_*$, by a *cap extremity edge* to each $\circ_B^j$, $1 \leq j \leq 2p_*$, and denote by $E_\circ$ the set of cap extremity edges.

A set $P \subseteq E_\circ$ is a *capping-set* if it does not contain any pair of incident edges and is maximal. Since each cap extremity of A is connected to each cap extremity of B, the size of any (maximal) capping-set is $2p_*$. A consistent decomposition $Q[S, P]$ of $FFR_\circ(A, B, \mathcal{S}_x)$ is induced by a sibling-set $S \subseteq E_\gamma$ and a (maximal) capping-set $P \subseteq E_\circ$ and is composed of vertex disjoint cycles that cover all vertices of $FFR_\circ(A, B, \mathcal{S}_x)$.

▶ **Theorem 2.** *Let $\mathbb{P}_{MAX}$ be the set of all distinct (maximal) capping-sets from $FFR_\circ(A, B, \mathcal{S}_x)$. For each sibling-set S of $FFR(A, B, \mathcal{S}_x)$ and $FFR_\circ(A, B, \mathcal{S}_x)$, we have*

$$\mathrm{d}_{\mathrm{DCJ}}^{\mathrm{ID}}(D[S]) = \min_{P \in \mathbb{P}_{MAX}} \{\mathrm{d}_{\mathrm{DCJ}}^{\mathrm{ID}}(Q[S, P])\}\,, \text{ and}$$

$$\mathrm{wd}_{DCJ}^{ID}(D[S]) = \min_{P \in \mathbb{P}_{MAX}} \{\mathrm{wd}_{DCJ}^{ID}(Q[S, P])\}\,.$$

Proof. Each capping-set corresponds to exactly p_* caps. In addition, all adjacencies, including the $|\kappa_A - \kappa_B|$ artificial adjacencies between cap extremities, are part of each consistent decomposition. Recall that each sibling-set S of $FFR_\circ(A, B, \mathcal{S}_x)$ corresponds to a matching M_S of $\mathcal{S}_x(A, B)$. The set of consistent decompositions include all possible distinct consistent decompositions induced by S together with one distinct element of $\mathbb{P}_{\mathrm{MAX}}$. Theorem 1 states that the pair of matched genomes A^{M_S} and B^{M_S} can be optimally capped with p_* caps and $|\kappa_A - \kappa_B|$ artificial adjacencies. Therefore, it is clear that $\mathrm{d}_{\mathrm{DCJ}}^{\mathrm{ID}}(D[S]) = \min_{P \in \mathbb{P}_{\mathrm{MAX}}}\{\mathrm{d}_{\mathrm{DCJ}}^{\mathrm{ID}}(Q[S, P])\}$. Since the capping does not change the sizes of the sibling-sets and their weights and complements, it is also clear that $\mathrm{wd}_{\mathrm{DCJ}}^{\mathrm{ID}}(D[S]) = \min_{P \in \mathbb{P}_{\mathrm{MAX}}}\{\mathrm{wd}_{\mathrm{DCJ}}^{\mathrm{ID}}(Q[S, P])\}$. ◀

Alternative formula for computing the indel-potential of cycles. The consistent decompositions of $FFR_\circ(A, B, \mathcal{S}_x)$ are composed exclusively of cycles, and the number of runs $\Lambda(C)$ of a cycle C is always in $\{0, 1, 2, 4, 6, \ldots\}$. Therefore, the formula to compute the indel-potential of a cycle C can be simplified to

$$\lambda(C) = \begin{cases} \Lambda(C)\,, & \text{if } \Lambda(C) \in \{0, 1\} \\ 1 + \frac{\Lambda(C)}{2}\,, & \text{if } \Lambda(C) \in \{2, 4, 6, \ldots\} \end{cases}$$

that can still be redesigned to a form that can be easier implemented in the ILP [4]. First, let a *transition* in a cycle C be an indel-free segment of C that is between a run in one genome and a run in the other genome and denote by $\aleph(C)$ the number of transitions in C. Observe that, if C is indel-free, then obviously $\aleph(C) = 0$. If C has a single run, then we also have $\aleph(C) = 0$. On the other hand, if C has at least 2 runs, then $\aleph(C) = \Lambda(C)$. The new formula is split into two parts. The first part is the function $r(C)$, defined as $r(C) = 1$ if

$\Lambda(C) \geq 1$, otherwise $r(C) = 0$, that simply tests whether C is indel-enclosing or indel-free. The second part depends on the number of transitions $\aleph(C)$, and the complete formula stands as follows [4]:

$$\lambda(C) = r(C) + \frac{\aleph(C)}{2}\,.$$

Distance formula. Note that the number of indel-enclosing components is $\sum_{C \in Q[S,P]} r(C) = c_Q^r + s_Q$, where c_Q^r and s_Q are the number of indel-enclosing AB-cycles and the number of circular singletons in $Q[S,P]$, respectively. Furthermore, the number of indel-free AB-cycles of $Q[S,P]$ is $c_Q^{\tilde{r}} = c_Q - c_Q^r$. We can now compute the values

$$\begin{aligned}
\mathrm{d}_{\mathrm{DCJ}}^{\mathrm{ID}}(Q[S,P]) &= p_* + \frac{|S|}{2} - c_Q + \sum_{C \in Q[S,P]} \lambda(C) \\
&= p_* + \frac{|S|}{2} - c_Q + \sum_{C \in Q[S,P]} \left(r(C) + \frac{\aleph(C)}{2} \right) \\
&= p_* + \frac{|S|}{2} - c_Q^{\tilde{r}} + s_Q + \sum_{C \in Q[S,P]} \frac{\aleph(C)}{2}\,, \quad \text{and}
\end{aligned}$$

$$\begin{aligned}
\mathrm{wd}_{\mathrm{DCJ}}^{\mathrm{ID}}(Q[S,P]) &= \mathrm{d}_{\mathrm{DCJ}}^{\mathrm{ID}}(Q[S,P]) + \frac{|S|}{2} - \frac{w(S)}{2} + w(\widetilde{S}) \\
&= p_* + |S| - c_Q^{\tilde{r}} + s_Q + \sum_{C \in Q[S,P]} \frac{\aleph(C)}{2} - \frac{w(S)}{2} + w(\widetilde{S})\,. \qquad (2)
\end{aligned}$$

Given that $\mathbb{S}$ and $\mathbb{P}_{\mathrm{MAX}}$ are, respectively, the sets of all sibling-sets and all maximal capping-sets of $\mathit{FFR}_\circ(A, B, \mathcal{S}_x)$, the final version of our optimization problem is

$$\mathrm{ffd}_{\mathrm{DCJ}}^{\mathrm{ID}}(A, B, \mathcal{S}_x) = \min_{S \in \mathbb{S}, P \in \mathbb{P}_{\mathrm{MAX}}} \left\{ \mathrm{wd}_{\mathrm{DCJ}}^{\mathrm{ID}}(Q[S,P]) \right\}.$$

5 ILP formulation to compute the family-free DCJ-indel distance

Our formulation is an adaptation of the ILP for computing the DCJ-indel distance of family-based natural genomes, by Bohnenkämper et al. [4], that is itself an extension of the ILP for computing the DCJ distance of family-based balanced genomes, by Shao et al. [23]. The main differences between our approach and the approach from [4] are the underlying graphs and the objective functions. The general idea is searching for a sibling-set, that, together with a maximal capping-set, gives an optimal consistent cycle decomposition of the capped diagram $\mathit{FFR}_\circ(A, B, \mathcal{S}_x) = (V, E)$, where the set of edges comprises all disjoint sets of distinct types: $E = E_\gamma \cup E_\circ \cup E_{\mathrm{adj}}^A \cup E_{\mathrm{adj}}^B \cup E_{\mathrm{id}}^A \cup E_{\mathrm{id}}^B$. While in the ILP from [4] the search space is restricted to maximal sibling-sets, in the family-free DCJ-indel distance the search space includes all sibling-sets, of any size.

In Algorithm 1 we give the formulation for computing $\mathrm{ffd}_{\mathrm{DCJ}}^{\mathrm{ID}}(A, B, \mathcal{S}_x)$, distributed in three main parts. Counting indel-free cycles in the decomposition makes up the first part, depicted in constraints (C.01)–(C.06), variables and domains (D.01)–(D.03). The second part is for counting transitions, described in constraints (C.07)–(C.10), variables and domains (D.04)–(D.05). The last part describes how to count the number of circular singletons, with constraint (C.11), variable and domain (D.06). The objective function of our ILP minimizes the size of the sibling-set, with sum over variables x_e, the number of circular singletons, calculated by the sum over variables s_k, half the overall number of transitions in indel-enclosing *AB*-cycles, calculated by the sum over variables t_e, and the weight of all indel edges in the decomposition, given by the sum over their weights $w_e x_e$ for all $e \in E_{\text{id}}$, while maximizing both the number of indel-free cycles, counted by the sum over variables z_i, and half of the weights of the extremity edges in the decomposition, given by the sum over their weights $w_e x_e$ for all edges $e \in E_\gamma$. The minimization is not affected by constant p_*, that is included in the objective function to keep the correspondence to Equation (2).

Algorithm 1 ILP for computing the family-free DCJ-indel distance.

$$
\text{min} \quad p_* + \sum_{e\in E_\gamma} x_e - \sum_{1\le i\le |V|} z_i + \sum_{k\in K} s_k + \frac{1}{2}\sum_{e\in E} t_e - \frac{1}{2}\sum_{e\in E_\gamma} w_e x_e + \sum_{e\in E_{\text{id}}} w_e x_e
$$

$$
\begin{array}{lrcllr}
\text{s. t.} & x_e & = & 1 & \forall\ e \in E^A_{\text{adj}} \cup E^B_{\text{adj}} & \text{(C.01)} \\
 & \displaystyle\sum_{uv\in E} x_{uv} & = & 2 & \forall\ u \in V & \text{(C.02)} \\
 & x_e & = & x_d & \forall\ e,d \in E_\gamma,\ e,d \text{ are siblings} & \text{(C.03)} \\
 & \begin{array}{r} y_i \\ y_j \end{array} & \begin{array}{c} \le \\ \le \end{array} & \left.\begin{array}{l} y_j + i(1 - x_{v_i v_j}) \\ y_i + j(1 - x_{v_i v_j}) \end{array}\right\} & \forall\ v_i v_j \in E & \text{(C.04)} \\
 & \begin{array}{r} y_i \\ y_j \end{array} & \begin{array}{c} \le \\ \le \end{array} & \left.\begin{array}{l} i(1 - x_{v_i v_j}) \\ j(1 - x_{v_i v_j}) \end{array}\right\} & \forall\ v_i v_j \in E^A_{\text{id}} \cup E^B_{\text{id}} & \text{(C.05)} \\
 & i z_i & \le & y_i & \forall\ 1 \le i \le |V| & \text{(C.06)} \\
 & \begin{array}{r} r_v \\ r_{v'} \end{array} & \begin{array}{c} \le \\ \ge \end{array} & \left.\begin{array}{l} 1 - x_{uv} \\ x_{u'v'} \end{array}\right\} & \begin{array}{l} \forall\ uv \in E^A_{\text{id}} \\ \forall\ u'v' \in E^B_{\text{id}} \end{array} & \text{(C.07)} \\
 & \begin{array}{r} t_{uv} \\ t_{uv} \end{array} & \begin{array}{c} \ge \\ \ge \end{array} & \left.\begin{array}{l} r_v - r_u - (1 - x_{uv}) \\ r_u - r_v - (1 - x_{uv}) \end{array}\right\} & \forall\ uv \in E & \text{(C.08)} \\
 & \displaystyle\sum_{d\in E^A_{\text{id}}\,,\ d\cap e\neq\emptyset} x_d - t_e & \ge & 0 & \forall\ e \in E^A_{\text{adj}} & \text{(C.09)} \\
 & t_e & = & 0 & \forall\ e \in E \setminus E^A_{\text{adj}} & \text{(C.10)} \\
 & \displaystyle\sum_{e\in E^k_{\text{id}}} x_e - |k| & \le & s_k & \forall\ k \in K & \text{(C.11)} \\
\text{and} & x_e & \in & \{0,1\} & \forall\ e \in E & \text{(D.01)} \\
 & 0 & \le & y_i \ \le\ i & \forall\ 1 \le i \le |V| & \text{(D.02)} \\
 & z_i & \in & \{0,1\} & \forall\ 1 \le i \le |V| & \text{(D.03)} \\
 & r_v & \in & \{0,1\} & \forall\ v \in V & \text{(D.04)} \\
 & t_e & \in & \{0,1\} & \forall\ e \in E & \text{(D.05)} \\
 & s_k & \in & \{0,1\} & \forall\ k \in K & \text{(D.06)} \\
 & p_* & = & \max\{\kappa_A, \kappa_B\} & & \text{(D.07)}
\end{array}
$$

Comparison to related models. Since the pre-requisites of a family-free setting differ substantially from those of a family-based setting, we could not compare our approach to the one from [4]. We intend to perform such a comparison in a future work, for example by using pairwise similarities to cluster the genes into families. Comparing our approach to the original family-free DCJ distance was also not possible, because the ILP provided in [16] is only suitable for unichromosomal genomes. Again, we intend to perform such a comparison in a future work, after we implement an ILP that is able to compute the family-free DCJ distance of multichromosomal genomes.

Unweighted version. In the present work, for comparison purposes, we also implemented a simpler version of the family-free DCJ-indel distance, that simply ignores all weights. This version is called *unweighted* family-free DCJ-indel distance, and consists of finding a sibling-set in $\mathit{FFR}_\circ(A, B, \mathcal{S}_x)$ that minimizes $\mathrm{d}_{\text{DCJ}}^{\text{ID}}(D[S, P])$. But here it is important to observe that smaller sibling-sets, that simply discard blocks of contiguous markers, tend to give the smaller distances. Considering the similarity graph $\mathcal{S}_{0.1}(A, B)$ of Figure 3, the trivial empty matching gives the distance $\mathrm{d}_{\text{DCJ}}^{\text{ID}}(A^{M_\emptyset}, B^{M_\emptyset}) = 2$ (deletion of the chromosome of A followed by the insertion of the chromosome of B). For the other matchings we have $\mathrm{d}_{\text{DCJ}}^{\text{ID}}(A^{M_1}, B^{M_1}) = 4$ and $\mathrm{d}_{\text{DCJ}}^{\text{ID}}(A^{M_2}, B^{M_2}) = \mathrm{d}_{\text{DCJ}}^{\text{ID}}(A^{M_3}, B^{M_3}) = 3$. We then restrict the search space to *maximal* sibling-sets only, avoiding that blocks of markers are discarded. However, this could also enforce weak connections. In the example shown in Figure 3, both maximal matchings M_1 and M_2 have weak edges with weights 0.2 and 0.3. Matching M_3 has only edges with weight at least 0.6, but it would be ignored for being non-maximal. Enforcing weak connections can be prevented by removing them from the similarity graph, that is, by assigning a higher value to the cutting threshold x. Given that $\mathbb{S}_{\text{MAX}}$ and $\mathbb{P}_{\text{MAX}}$ are, respectively, the sets of all maximal sibling-sets and all maximal capping-sets of $\mathit{FFR}_\circ(A, B, \mathcal{S}_x)$, the unweighted version of the problem is then:

$$\mathrm{unwffd}_{\text{DCJ}}^{\text{ID}}(A, B, \mathcal{S}_x) = \min_{S \in \mathbb{S}_{\text{MAX}}, P \in \mathbb{P}_{\text{MAX}}} \left\{ \mathrm{d}_{\text{DCJ}}^{\text{ID}}(Q[S, P]) \right\}.$$

For computing the unweighted $\mathrm{unwffd}_{\text{DCJ}}^{\text{ID}}(A, B, \mathcal{S}_x)$ we need to slightly modify the ILP described in Algorithm 1. The details can be found in [19].

Implementation. The ILPs for computing both the family-free DCJ-indel distance and its unweighted version were implemented and can be downloaded from our GitLab server at `https://gitlab.ub.uni-bielefeld.de/gi/gen-diff`.

Data analysis. For all pairwise comparisons, we obtained gene similarities using the FFGC pipeline[2] [11], with the following parameters: (i) 1 for the minimum number of genomes for which each gene must share some similarity in, (ii) 0.1 for the stringency threshold, (iii) 1 for the BLAST e-value, and (iv) default values for the remaining parameters. As an ILP solver, for all experiments we ran CPLEX with 8 2.67GHz cores.

Cutting threshold. Differently from the unweighted version, that requires a cutting threshold of about $x = 0.5$ to give accurate results, the weighted $\mathrm{ffd}_{\text{DCJ}}^{\text{ID}}$ was designed to be computed with all given pairwise similarities, i.e., with the cutting threshold $x = 0$, that leads to a "complete" family-free relational diagram. Such a diagram would be too large to be handled in practice, therefore, if $x = 0$, we consider only the similarities that are strictly greater than 0. Nevertheless, for bigger instances the diagram with similarities close to 0 might still be too large to be solved in reasonable time. Hence, for some instances it may be necessary to do a small increase of the cutting threshold. Our experiments in real data (described in Section 5.2) show that small similarities have a minor impact on the computed distance, therefore, by adopting a small cutting threshold x up to 0.3, it is possible to reduce the diagram and solve bigger instances, still with good accuracy.

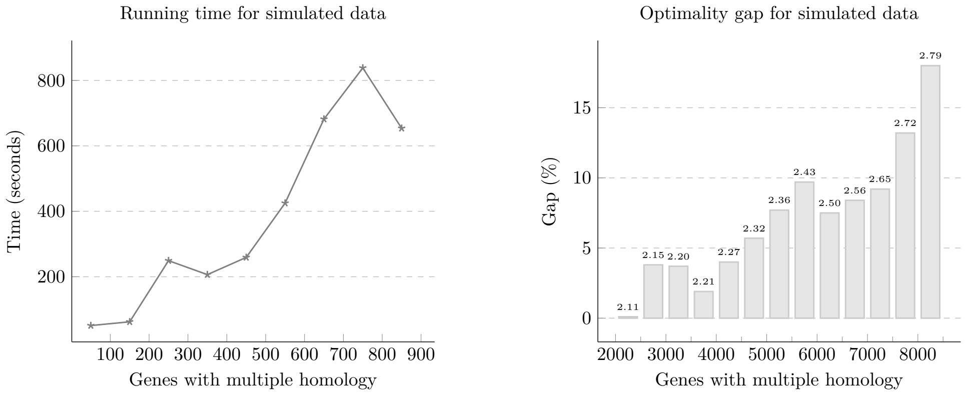
[2] `https://bibiserv.cebitec.uni-bielefeld.de/ffgc`

5.1 Performance evaluation

We generated simulated genomes using Artificial Life Simulator (ALF) [10] in order to benchmark our algorithm for computing the family-free DCJ-indel distance. We simulated and compared 190 pairs of genomes with different duplication rates, keeping all other parameters fixed (e.g. rearrangement, indel and mutation rates). The extant genomes have around 10,000 genes. We obtained gene similarities between simulated genomes using FFGC. For each genome pair, a threshold of $x = 0.1$ resulted in up to 8,400 genes with multiple homology relations (i.e. vertices with degree > 1 in $\mathcal{S}_{0.1}(A, B)$) and from 2 to 2.8 relations on average for those genes. In addition, each pair is about 3,000 rearrangement events away from each other. The complete parameter sets used for running ALF, together with additional information on simulated genomes, can be found in Appendix A.

For computing the family-free DCJ-indel distances, we ran CPLEX with maximum CPU time of 1 hour. Results were grouped depending on the number of genes with multiple homology relations in the respective genome pairs. Figure 5 summarizes the performance of our weighted family-free DCJ-indel distance formulation. The running times escalate quickly as the number of genes with multiple homologies increase (Figure 5a, grouped in intervals of 100), reaching the time limit after 2,000 of them (Figure 5b, grouped in intervals of 500). The optimality gap is the relative gap between the best solution found and the upper bound found by the solver, calculated by $(\frac{\text{upper bound}}{\text{best solution}} - 1) \times 100$, and appears to grow, for our simulated data, linearly in the number of genes with multiple homologies (Figure 5b).

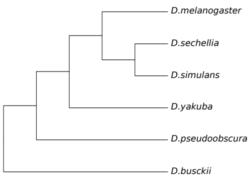
The solution time and the optimality gap of our algorithm clearly depends less on genome sizes and more on the multiplicity of homology relations. In our experiments, we were able to find in 1 hour optimal or near-optimal solutions for genomes with 10,000 genes and up to 4,000 genes with 2.2 homology relations on average. Our formulation should be able to handle, for instance, the complete genomes of bacteria, fungi and insects, or even sets of chromosomes of mammal and plant genomes.



(a) **(b)**

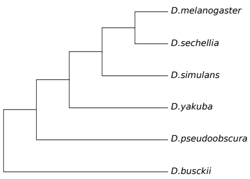
Figure 5 Results of the weighted family-free DCJ-indel distance given by the solver, (a) shows the average running time for instances grouped by the number of vertices with degree > 1 in $\mathcal{S}_{0.1}(A, B)$ (in intervals of 100, those greater than 900 are not shown), and (b) for groups of instances that did not finish within the time limit of 1 hour, the average optimality gap and the average number of homology relations for those genes with multiple homologies (in intervals of 500).

ILP	SET	x
$\mathrm{ffd}^{\mathrm{ID}}_{\mathrm{DCJ}}$	Gen	0.3
$\mathrm{ffd}^{\mathrm{ID}}_{\mathrm{DCJ}}$	Xchr	0.0
		0.1
		0.2
		0.3
$\mathrm{unwffd}^{\mathrm{ID}}_{\mathrm{DCJ}}$	Xchr	0.5



(a)

ILP	SET	x
$\mathrm{unwffd}^{\mathrm{ID}}_{\mathrm{DCJ}}$	Gen	0.3
$\mathrm{unwffd}^{\mathrm{ID}}_{\mathrm{DCJ}}$	Xchr	0.0
		0.3



(b)

Figure 6 Based on distance matrices calculated by our ILPs for the pairwise comparisons of complete genomes (Gen) or only X chromosomes (Xchr) of *Drosophila*, we built phylogenetic trees computed by the Neighbor-Joining method [14, 20]. The output of this algorithm is an unrooted tree, and we assumed the most distant species *D. busckii* as the outgroup for rooting the trees. All comparisons converged to exactly two trees, and next to each tree we give a list of comparisons that produced that tree. The tree in (a) agrees with the reference shown in Figure 7 (Appendix B), while the tree in (b) diverges from the reference in a single branch.

5.2 Real data analysis

We evaluated the potential of our approach by comparing genomes of fruit flies from the genus *Drosophila* [1, 9, 17, 26], including the following species: *D. busckii*, *D. melanogaster*, *D. pseudoobscura*, *D. sechellia*, *D. simulans* and *D. yakuba*. A reference phylogenetic tree of these species is shown in Figure 7, in Appendix B, where we also give the sources of the DNA sequences for each analyzed genome, and additional information on the experiments. Each genome has approximately 150Mb, with about 13,000 genes distributed in 5–6 chromosomes. We obtained gene similarities using FFGC and performed two separate experiments, whose computed distances were used to build phylogenetic trees using Neighbor-Joining [14, 20].

Pairwise comparison of complete genomes. In this experiment, genomes in each comparison comprise together $\sim$ 13,000 genes with multiple homologies (11.2 on average), some of them having about 90 relations considering similarities that are strictly greater than $x = 0$. Since these instances were too large, we set the threshold to $x = 0.3$. We then ran CPLEX with maximum CPU time of 3 hours. All $\mathrm{ffd}^{\mathrm{ID}}_{\mathrm{DCJ}}$ computations finished within the time limit, most of them in less than 10 minutes, whereas the unweighted $\mathrm{unwffd}^{\mathrm{ID}}_{\mathrm{DCJ}}$ computations, in spite of having a search space of maximal sibling-sets, that is much smaller, surprisingly took from 1 to 3 hours. We conjecture that this is due to a large number of co-optimal solutions in the unweighted version, while in $\mathrm{ffd}^{\mathrm{ID}}_{\mathrm{DCJ}}$ the co-optimality is considerably minimized by weights, which helps the solver to converge faster. While the tree given by $\mathrm{ffd}^{\mathrm{ID}}_{\mathrm{DCJ}}$, shown in Figure 6a, agrees with the reference tree, the tree given by $\mathrm{unwffd}^{\mathrm{ID}}_{\mathrm{DCJ}}$, shown in Figure 6b, diverges from the reference in a single branch. Details of the results are given in Appendix B.1.

Pairwise comparison of X chromosomes. We also did an experiment with smaller instances, composed of pairwise comparisons of X chromosomes only, so that we could evaluate the impact of the cutting threshold on the accuracy of the approach. In this experiment, considering similarities that are strictly greater than $x = 0$, each pair comprises 1,000–2,000 genes with multiple homologies (5 on average) with as many as 30 relations.

We computed $\mathrm{ffd}^{\mathrm{ID}}_{\mathrm{DCJ}}$ with cutting thresholds $x = 0$, $x = 0.1$, $x = 0.2$ and $x = 0.3$, always obtaining the accurate phylogenetic tree from Figure 6a. These results suggest that a small cutting threshold allows to reduce the size of the instances, without having a big impact in the accuracy of $\mathrm{ffd}^{\mathrm{ID}}_{\mathrm{DCJ}}$.

In addition, we computed $\text{unwffd}_{\text{DCJ}}^{\text{ID}}$ with cutting thresholds $x = 0$ and $x = 0.3$, both resulting in the slightly inaccurate tree from Figure 6b, and $x = 0.5$, that also resulted in the accurate tree from Figure 6a. As expected, in the unweighted formulation the cutting threshold plays a major role in the accuracy of the calculated distances.

The analyses were done with maximum CPU time of 1 hour. The comparisons finished within a few seconds for most of instances, except for $\text{unwffd}_{\text{DCJ}}^{\text{ID}}$ with threshold $x = 0$, for which the majority of the pairwise comparisons reached the time limit – with an optimality gap of less than 3.5% though (see Appendix B.2).

Length of indel segments. As a generalization of the singular DCJ-indel model [6], the basic idea behind our approach is that runs can be merged and accumulated with DCJ operations. This is a more parsimonious alternative to the trivial approach of inserting or deleting exclusive markers individually. However, it raises the question of whether the indels then tend to be very long, and whether this makes biological sense. Considering that it is possible to distribute the runs so that each indel is composed of 1-2 runs, we can say that the lengths of the runs play a major role in defining the length of indel segments. In the particular analysis of *Drosophila* complete genomes, we have an average run length of 5.1, while the maximum run length is 121. We conjecture that the long runs are mostly composed of genes that are part of a contiguous segment from the beginning, and are not really accumulated by DCJ operations. In a future work we intend to have a closer look into the long runs, so that we can characterize their structures and verify this conjecture.

6 Conclusions and discussion

In this work we proposed a new genomic distance, for the first time integrating DCJ and indel operations in a family-free setting. In this setting the whole analysis requires less pre-processing and no classification of the data, since it can be performed based on the pairwise similarities of markers in both genomes. Based on the positions and orientations of markers in both genomes we build the *family-free relational diagram.* We then assign weights to the edges of the diagram, according to the given pairwise similarities. A *sibling-set* of edges corresponds to a set of matched markers in both genomes. Our approach transfers weights from the edges to matched and unmatched markers, so that, again for the first time, an optimal solution does not necessarily need to maximize the number of matched markers. Instead, the search space of our approach allows solutions composed of any number of matched markers. The computation of our new family-free DCJ-indel distance is NP-hard and we provide an efficient ILP formulation to solve it.

The experiments on simulated data show that our ILP can handle not only bacterial genomes, but also complete genomes of fungi and insects, or sets of chromosomes of mammals and plants. We performed a comparison study of six fruit fly genomes, using the obtained distances to reconstruct the phylogenetic tree of the six species, obtaining accurate results. This study was a first validation of the quality of our method and a more rigorous evaluation will be performed in a future work. In particular, we intend to analyze the reasons behind insertions and deletions of long segments and verify the quality of the obtained gene matchings, by comparing them to the annotated orthologies given by public databases. Furthermore, as already mentioned, we plan to compare our ILP to the one given in [4], once we manage to cluster the genes into families, and also to implement an ILP that is able to compute the family-free DCJ distance described in [16] for multichromosomal genomes, so that we can compare it to our ILP.

References

1 Mark D. Adams, Susan E. Celniker, Robert A. Holt, *et al.* The genome sequence of *Drosophila melanogaster*. *Science*, 287:2185–2195, 2000. doi:10.1126/science.287.5461.2185.

2 Sébastien Angibaud, Guillaume Fertin, Irena Rusu, Annelyse Thévenin, and Stéphane Vialette. On the approximability of comparing genomes with duplicates. *Journal of Graph Algorithms and Applications*, 13(1):19–53, 2009. doi:10.7155/jgaa.00175.

3 Anne Bergeron, Julia Mixtacki, and Jens Stoye. A unifying view of genome rearrangements. In *Proc. of WABI*, volume 4175 of *Lecture Notes in Bioinformatics*, pages 163–173, 2006. doi:10.1007/11851561_16.

4 Leonard Bohnenkämper, Marília D. V. Braga, Daniel Doerr, and Jens Stoye. Computing the rearrangement distance of natural genomes. In *Proc. of RECOMB*, volume 12074 of *Lecture Notes in Bioinformatics*, pages 3–18, 2020. doi:10.1007/978-3-030-45257-5_1.

5 Marília D. V. Braga, Cedric Chauve, Daniel Doerr, Katharina Jahn, Jens Stoye, Annelyse Thévenin, and Roland Wittler. The potential of family-free genome comparison. In C. Chauve, N. El-Mabrouk, and E. Tannier, editors, *Models and Algorithms for Genome Evolution*, chapter 13, pages 287–307. Springer, 2013. doi:10.1007/978-1-4471-5298-9_13.

6 Marília D. V. Braga, Eyla Willing, and Jens Stoye. Double cut and join with insertions and deletions. *Journal of Computational Biology*, 18(9):1167–1184, 2011. doi:10.1089/cmb.2011.0118.

7 David Bryant. The complexity of calculating exemplar distances. In David Sankoff and Joseph H. Nadeau, editors, *Comparative Genomics*, pages 207–211. Springer, 2000. doi:10.1007/978-94-011-4309-7_19.

8 Laurent Bulteau and Minghui Jiang. Inapproximability of (1,2)-exemplar distance. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 10(6):1384–1390, 2013. doi:10.1109/TCBB.2012.144.

9 Andrew G. Clark, Michael B. Eisen, Douglas R. Smith, *et al.* Evolution of genes and genomes on the *Drosophila* phylogeny. *Nature*, 450:203–218, 2007. doi:10.1038/nature06341.

10 Daniel A. Dalquen, Maria Anisimova, Gaston H. Gonnet, and Christophe Dessimoz. ALF – a simulation framework for genome evolution. *Mol Biol Evol*, 29(4):1115, 2012. doi:10.1093/molbev/msr268.

11 Daniel Doerr, Pedro Feijão, and Jens Stoye. Family-free genome comparison. In João C. Setubal, Jens Stoye, and Peter F. Stadler, editors, *Comparative Genomics: Methods and Protocols*, pages 331–342. Springer, 2018. doi:10.1007/978-1-4939-7463-4_12.

12 Daniel Doerr, Annelyse Thévenin, and Jens Stoye. Gene family assignment-free comparative genomics. *BMC Bioinformatics*, 13(Suppl 19):S3, 2012. doi:10.1186/1471-2105-13-S19-S3.

13 Sridhar Hannenhalli and Pavel A. Pevzner. Transforming men into mice (polynomial algorithm for genomic distance problem). In *Proc. of FOCS*, pages 581–592, 1995. doi:10.1109/SFCS.1995.492588.

14 Sudhir Kumar, Glen Stecher, Michael Li, Christina Knyaz, and Koichiro Tamura. MEGA X: molecular evolutionary genetics analysis across computing platforms. *Molecular Biology and Evolution*, 35(6):1547–1549, 2018. doi:10.1093/molbev/msy096.

15 Sudhir Kumar, Glen Stecher, Michael Suleski, and S. Blair Hedges. Timetree: a resource for timelines, timetrees, and divergence times. *Molecular Biology and Evolution*, 34(7):1812–1819, 2017. doi:10.1093/molbev/msx116.

16 Fábio V. Martinez, Pedro Feijão, Marília D. V. Braga, and Jens Stoye. On the family-free DCJ distance and similarity. *Algorithms for Molecular Biology*, 13(10), 2015. doi:10.1186/s13015-015-0041-9.

17 Stephen Richards, Yue Liu, Brian R. Bettencourt *et al.* Comparative genome sequencing of *Drosophila pseudoobscura*: Chromosomal, gene, and cis-element evolution. *Genome Research*, 15:1–18, 2005. doi:10.1101/gr.3059305.

18 Diego P. Rubert, Pedro Feijão, Marília D. V. Braga, Jens Stoye, and Fábio V. Martinez. Approximating the DCJ distance of balanced genomes in linear time. *Algorithms for Molecular Biology*, 12(3), 2017. doi:10.1186/s13015-017-0095-y.

19 Diego P. Rubert, Fábio V. Martinez, and Marília D. V. Braga. Natural family-free genomic distance. arXiv:2007.03556, 2020.

20 Naruya Saitou and Masatoshi Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987. doi:10.1093/oxfordjournals.molbev.a040454.

21 David Sankoff. Edit distance for genome comparison based on non-local operations. In *Proc. of CPM*, volume 644 of *Lecture Notes in Computer Science*, pages 121–135, 1992. doi:10.1007/3-540-56024-6_10.

22 David Sankoff. Genome rearrangement with gene families. *Bioinformatics*, 15(11):909–917, 1999. doi:10.1093/bioinformatics/15.11.909.

23 Mingfu Shao, Yu Lin, and Bernard Moret. An exact algorithm to compute the double-cut-and-join distance for genomes with duplicate genes. *Journal of Computational Biology*, 22(5):425–435, 2015. doi:10.1089/cmb.2014.0096.

24 Sophia Yancopoulos, Oliver Attie, and Richard Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346, 2005. doi:10.1093/bioinformatics/bti535.

25 Sophia Yancopoulos and Richard Friedberg. DCJ path formulation for genome transformations which include insertions, deletions, and duplications. *Journal of Computational Biology*, 16(10):1311–1338, 2009. doi:10.1089/cmb.2009.0092.

26 Qi Zhou and Doris Bachtrog. Ancestral chromatin configuration constrains chromatin evolution on differentiating sex chromosomes in *Drosophila*. *PLoS Genetics*, 11(6), 2015. doi:10.1371/journal.pgen.1005331.

A Generation of simulated data

Here we describe the process and the parameters used in Artificial Life Simulator (ALF) [10] for generating our simulated data. Each one of the 190 instances generated consists of a pair of simulated genomes. We used the default values for parameters not mentioned. PAM units were used as time scale for simulation, starting with a randomly generated root genome with 10,000 genes, whose lengths where drawn from a Gamma distribution with $k = 2.4019$ and $\theta = 133.8063$ (minimum length 100). We used a custom evolutionary tree defining an speciation event after 25 time units, resulting in two leaf species, which evolved for additional 25 time units. The WAG substitution model was used together with Zipfian indels in DNA sequences with rate 0.0002 (maximum length 50). Such rate varies among sites according to a Gamma distribution with shape 1 and 10 classes. In addition, we set the rate of invariable sites to 0.001. Inversions and translocations of up to 30 genes were allowed at a rate of 0.0025. Finally, for generating instances comprising genes with different numbers of homologies, we varied the gene duplication and the gene loss rates between 1×10^{-5} and 2×10^{-3}.

B Analysis of *Drosophila* genomes

We downloaded the genomes of 6 species of *Drosophila* [1, 9, 17, 26] from NCBI[3]. In our experiments we used the assemblies listed in Figure 7, with their respective gene annotations.

[3] https://www.ncbi.nlm.nih.gov

Species	NCBI Assembly	Species	NCBI Assembly
Drosophila busckii	ASM1175060v1	*Drosophila sechellia*	ASM438219v1
Drosophila melanogaster	Release 6 plus ISO1 MT	*Drosophila simulans*	ASM75419v2
Drosophila pseudoobscura	UCI_Dpse_MV25	*Drosophila yakuba*	dyak_caf1

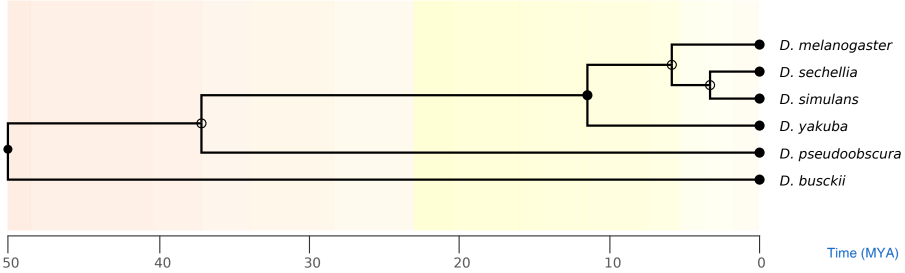


Figure 7 List of genomes used in our experiments and a reference phylogenetic tree of the respective species of *Drosophila* given by TimeTree [15], a public knowledge-base for information on the tree-of-life and its evolutionary timescale.

As already mentioned, we obtained pairwise similarities between genes of *Drosophila* genomes using the FFGC pipeline[4] [11] with the following parameters: (i) 1 for the minimum number of genomes for which each gene must share some similarity in, (ii) 0.1 for the stringency threshold, (iii) 1 for the BLAST e-value, and (iv) default values for the remaining parameters.

In the following subsections, in-depth information is provided on the results for experiments using complete genomes and X chromosomes of the listed *Drosophila* species.

B.1 Complete genomes

The first tables in this section detail the results of the comparison of complete genomes in terms of the BLAST alignment performed for all genes, and the corresponding similarity graphs for each genome pair without cutting threshold. This data was generated using the FFCG pipeline with the parameters described above. Unplaced scaffolds were discarded, decreasing the number of genes from $\sim$ 15,000 to $\sim$ 13,000. Table 1 outlines the number of gene pairs in each similarity range for each pair of genomes. Table 2 shows the number of genes with no homology relations (which induce trivial selections of indel edges in the relational diagram), the number of genes with exactly one homology relation and the number of genes with multiple homologies (which pose a significant challenge to the solver). The computed distances and elapsed time (or gap in % when the solver reaches the time limit) in the pairwise comparisons with cutting threshold 0.3 are shown in Tables 3 and 4. The solver was set to stop after finding a solution with optimality gap smaller than 0.5% or after 3 hours.

[4] `https://bibiserv.cebitec.uni-bielefeld.de/ffgc`

Table 1 Distribution of similarities between genes (and percentage) in pairwise comparisons of complete genomes.

species	similarity	*pseudoobscura*	*sechellia*	*simulans*	*yakuba*	*busckii*
melanogaster	(0.0-0.2)	53648 (60.09%)	33409 (48.69%)	34803 (49.15%)	38143 (51.71%)	53733 (65.42%)
	[0.2-0.4)	19034 (21.32%)	17822 (25.97%)	18566 (26.22%)	18748 (25.42%)	16129 (19.64%)
	[0.4-0.6)	6036 (6.76%)	3896 (5.68%)	4019 (5.68%)	4195 (5.69%)	5207 (6.34%)
	[0.6-0.8)	4993 (5.59%)	1826 (2.66%)	1909 (2.70%)	3010 (4.08%)	4300 (5.23%)
	[0.8-1.0]	5570 (6.24%)	11666 (17.00%)	11513 (16.26%)	9663 (13.10%)	2772 (3.37%)
		89281 (100%)	68619 (100%)	70810 (100%)	73759 (100%)	82141 (100%)
pseudoobscura	(0.0-0.2)		53777 (62.13%)	54221 (61.83%)	54147 (61.96%)	54104 (65.78%)
	[0.2-0.4)		18169 (20.99%)	18724 (21.35%)	18645 (21.34%)	15940 (19.38%)
	[0.4-0.6)		5466 (6.32%)	5601 (6.39%)	5595 (6.40%)	5183 (6.30%)
	[0.6-0.8)		4838 (5.59%)	4895 (5.58%)	4797 (5.49%)	4223 (5.13%)
	[0.8-1.0]		4303 (4.97%)	4255 (4.85%)	4202 (4.81%)	2798 (3.40%)
			86553 (100%)	87696 (100%)	87386 (100%)	82248 (100%)
sechellia	(0.0-0.2)			34227 (49.87%)	38169 (52.98%)	53105 (66.03%)
	[0.2-0.4)			17325 (25.25%)	17430 (24.19%)	15521 (19.30%)
	[0.4-0.6)			3721 (5.42%)	4075 (5.66%)	5003 (6.22%)
	[0.6-0.8)			1277 (1.86%)	2987 (4.15%)	4175 (5.19%)
	[0.8-1.0]			12077 (17.60%)	9379 (13.02%)	2626 (3.26%)
				68627 (100%)	72040 (100%)	80430 (100%)
simulans	(0.0-0.2)				39218 (52.89%)	54066 (66.32%)
	[0.2-0.4)				18288 (24.66%)	15648 (19.20%)
	[0.4-0.6)				4287 (5.78%)	5115 (6.27%)
	[0.6-0.8)				2960 (3.99%)	4103 (5.03%)
	[0.8-1.0]				9395 (12.67%)	2589 (3.18%)
					74148 (100%)	81521 (100%)
yakuba	(0.0-0.2)					54022 (66.32%)
	[0.2-0.4)					15767 (19.36%)
	[0.4-0.6)					5027 (6.17%)
	[0.6-0.8)					4105 (5.04%)
	[0.8-1.0]					2540 (3.12%)
						81461 (100%)

Table 2 Association between genes in pairwise comparisons of complete genomes, considering pairwise similarities strictly greater than 0. The tables show the number of genes with zero, one and multiple homology relations, respectively. For all of them, the element stored in line i and column j represents the number of genes of the species i in the pairwise comparison of genomes i and j.

Number of unassociated genes

species		*melanog*	*pseudoob*	*sechellia*	*simulans*	*yakuba*	*busckii*
	#genes	13049	13399	13037	13023	12835	11371
melanogaster	13049	—	570	213	277	352	1183
pseudoobscura	13399	565	—	583	694	710	1211
sechellia	13037	189	620	—	263	393	1189
simulans	13023	335	779	345	—	484	1358
yakuba	12835	306	666	323	327	—	1225
busckii	11371	304	354	321	380	400	—

Number of genes uniquely associated

species		*melanog*	*pseudoob*	*sechellia*	*simulans*	*yakuba*	*busckii*
	#genes	13049	13399	13037	13023	12835	11371
melanogaster	13049	—	5439	6624	6533	6361	5107
pseudoobscura	13399	5775	—	5746	5704	5707	5205
sechellia	13037	6650	5487	—	6656	6307	5099
simulans	13023	6516	5394	6594	—	6237	4985
yakuba	12835	6288	5358	6242	6251	—	4982
busckii	11371	4797	4654	4749	4730	4725	—

Number of genes associated to at least two other genes

species		*melanog*	*pseudoob*	*sechellia*	*simulans*	*yakuba*	*busckii*
	#genes	13049	13399	13037	13023	12835	11371
melanogaster	13049	—	7040	6212	6239	6336	6759
pseudoobscura	13399	7059	—	7070	7001	6982	6983
sechellia	13037	6198	6930	—	6118	6337	6749
simulans	13023	6172	6850	6084	—	6302	6680
yakuba	12835	6241	6811	6270	6257	—	6628
busckii	11371	6270	6363	6301	6261	6246	—

Table 3 Computed $\mathrm{ffd}^{\mathrm{ID}}_{\mathrm{DCJ}}$ and elapsed time (or gap in %) in pairwise comparisons of complete genomes, with cutting threshold $x = 0.3$. The time limit for execution of the ILP solver is 10800s.

species	*pseudoobscura*	*sechellia*	*simulans*	*yakuba*	*busckii*
melanogaster	7373.7 (0.76%)	1925.5 (4431.78s)	2094.7 (109.60s)	3193.2 (201.49s)	7764.6 (540.19s)
pseudoobscura		7326.0 (163.12s)	7355.5 (764.24s)	7351.2 (5782.73s)	7784.0 (290.12s)
sechellia			1661.0 (103.33s)	3259.0 (146.88s)	7710.4 (415.23s)
simulans				3306.0 (216.77s)	7699.9 (115.54s)
yakuba					7667.4 (153.36s)

Table 4 Computed $\mathrm{unwffd}^{\mathrm{ID}}_{\mathrm{DCJ}}$ and elapsed time (or gap in %) in pairwise comparisons of complete genomes, with cutting threshold $x = 0.3$. The time limit for execution of the ILP solver is 10800s.

species	*pseudoobscura*	*sechellia*	*simulans*	*yakuba*	*busckii*
melanogaster	4084 (2.67%)	708 (0.96%)	933 (0.62%)	1269 (1.35%)	4791 (0.95%)
pseudoobscura		4088 (1.50%)	4176 (1.47%)	4142 (1.22%)	4797 (1.20%)
sechellia			905 (2812.89s)	1341 (1.10%)	4817 (0.98%)
simulans				1478 (1.44%)	4866 (0.84%)
yakuba					4820 (1.00%)

B.2 X chromosomes

Similarity values in pairwise comparisons are given in Table 5. Results for $\text{unwffd}^{\text{ID}}_{\text{DCJ}}$ and for $\text{ffd}^{\text{ID}}_{\text{DCJ}}$ are shown in Tables 6 and 7 (CPLEX was set to stop after finding a solution with optimality gap smaller than 0.1% or after 1 hour). The number of genes with 0, 1 and multiple homologies are given in Table 8.

Table 5 Distribution of similarities (and percentage) in pairwise comparisons of X chromosomes.

species	similarity	*pseudoobscura*	*sechellia*	*simulans*	*yakuba*	*busckii*
melanogaster	(0.0-0.2)	4710 (63.70%)	829 (22.38%)	897 (25.63%)	987 (28.81%)	2072 (48.64%)
	[0.2-0.4)	980 (13.25%)	576 (15.55%)	536 (15.31%)	528 (15.41%)	738 (17.23%)
	[0.4-0.6)	541 (7.32%)	352 (9.50%)	256 (7.31%)	242 (7.06%)	475 (11.15%)
	[0.6-0.8)	605 (8.18%)	271 (7.32%)	256 (7.31%)	412 (12.03%)	584 (13.71%)
	[0.8-1.0]	558 (7.55%)	1676 (45.25%)	1555 (44.43%)	1257 (36.69%)	391 (9.18%)
		7394 (100%)	3704 (100%)	3500 (100%)	3426 (100%)	4260 (100%)
pseudoobscura	(0.0-0.2)		4849 (64.60%)	4703 (65.15%)	4588 (64.64%)	5021 (66.59%)
	[0.2-0.4)		962 (12.82%)	907 (12.56%)	898 (12.65%)	953 (12.64%)
	[0.4-0.6)		539 (7.18%)	498 (6.90%)	495 (6.97%)	563 (7.47%)
	[0.6-0.8)		600 (7.99%)	585 (8.10%)	574 (8.09%)	584 (7.75%)
	[0.8-1.0]		556 (7.41%)	526 (7.29%)	543 (7.65%)	419 (5.56%)
			7506 (100%)	7219 (100%)	7098 (100%)	7540 (100%)
sechellia	(0.0-0.2)			773 (22.62%)	961 (28.16%)	2014 (47.90%)
	[0.2-0.4)			521 (15.24%)	532 (15.59%)	741 (17.62%)
	[0.4-0.6)			191 (5.59%)	266 (7.79%)	486 (11.56%)
	[0.6-0.8)			139 (4.07%)	423 (12.39%)	574 (13.65%)
	[0.8-1.0]			1795 (52.50%)	1231 (36.07%)	390 (9.27%)
				3419 (100%)	3413 (100%)	4205 (100%)
simulans	(0.0-0.2)				1038 (30.77%)	2069 (49.95%)
	[0.2-0.4)				506 (15.00%)	697 (16.83%)
	[0.4-0.6)				254 (7.53%)	448 (10.82%)
	[0.6-0.8)				403 (11.95%)	556 (13.42%)
	[0.8-1.0]				1172 (34.75%)	372 (8.98%)
					3373 (100%)	4142 (100%)
yakuba	(0.0-0.2)					2110 (50.62%)
	[0.2-0.4)					668 (16.03%)
	[0.4-0.6)					456 (10.94%)
	[0.6-0.8)					561 (13.46%)
	[0.8-1.0]					373 (8.95%)
						4168 (100%)

Table 6 Computed $\text{unwffd}^{\text{ID}}_{\text{DCJ}}$ and elapsed time (or gap in %) in pairwise comparisons of X chromosomes, with cutting thresholds $x = 0.0$, $x = 0.3$ and $x = 0.5$. The time limit is 3600s.

species	x	*pseudoobscura*	*sechellia*	*simulans*	*yakuba*	*busckii*
melanogaster	0.0	720 (2.80%)	132 (0.38%)	178 (30.93s)	218 (4.99s)	832 (3.49%)
	0.3	829 (1.70s)	160 (9.31s)	218 (0.98s)	293 (0.66s)	972 (0.90s)
	0.5	940 (0.46s)	228 (0.52s)	298 (0.40s)	397 (0.34s)	1003 (0.27s)
pseudoobscura	0.0		743 (3.13%)	743 (2.14%)	724 (1.40%)	912 (3.76%)
	0.3		836 (1.06s)	849 (1.03s)	837 (0.96s)	980 (1.06s)
	0.5		929 (0.44s)	938 (0.43s)	908 (0.43s)	1015 (0.39s)
sechellia	0.0			171 (45.95s)	236 (6.41s)	850 (2.19%)
	0.3			194 (1.06s)	301 (0.74s)	982 (2.11s)
	0.5			244 (0.44s)	423 (0.40s)	1014 (0.28s)
simulans	0.0				265 (18.14s)	863 (2.35%)
	0.3				336 (0.63s)	994 (0.87s)
	0.5				453 (0.33s)	1005 (0.25s)
yakuba	0.0					830 (1.73%)
	0.3					972 (0.72s)
	0.5					992 (0.24s)

Table 7 Computed $\mathrm{ffd}_{\mathrm{DCJ}}^{\mathrm{ID}}$ and elapsed time (or gap in %) in pairwise comparisons of X chromosomes, with cutting thresholds ranging between $x = 0.0$ and $x = 0.3$. The time limit is 3600s.

species	x	*pseudoobscura*	*sechellia*	*simulans*	*yakuba*	*busckii*
melanogaster	0.0	1390.3 (255.22s)	407.3 (0.32%)	432.4 (9.60s)	587.4 (4.21s)	1362.7 (109.59s)
	0.1	1370.1 (30.01s)	408.3 (0.34%)	433.8 (10.12s)	590.0 (4.08s)	1363.9 (11.49s)
	0.2	1326.0 (6.28s)	412.7 (174.06s)	436.6 (5.25s)	601.0 (2.17s)	1344.7 (5.12s)
	0.3	1296.0 (4.13s)	416.7 (24.47s)	445.4 (3.55s)	609.3 (1.64s)	1321.8 (2.87s)
pseudoobscura	0.0		1417.1 (258.89s)	1375.6 (281.95s)	1361.7 (94.68s)	1515.7 (368.78s)
	0.1		1394.1 (36.51s)	1355.1 (45.6s)	1337.0 (17.74s)	1491.7 (33.27s)
	0.2		1344.1 (3.64s)	1309.7 (329.25s)	1299.5 (3.34s)	1433.3 (5.61s)
	0.3		1308.0 (5.56s)	1278.0 (3.73s)	1262.3 (3.69s)	1374.1 (4.69s)
sechellia	0.0			352.5 (5.90s)	626.8 (4.70s)	1378.2 (74.01s)
	0.1			352.8 (5.83s)	630.4 (3.92s)	1377.1 (23.36s)
	0.2			351.9 (3.56s)	635.3 (3.08s)	1354.2 (5.38s)
	0.3			355.0 (2.55s)	641.3 (1.92s)	1328.3 (4.18s)
simulans	0.0				617.8 (7.78s)	1344.0 (80.84s)
	0.1				621.3 (5.27s)	1342.7 (29.58s)
	0.2				626.2 (2.04s)	1316.7 (5.50s)
	0.3				637.8 (1.99s)	1295.5 (3.25s)
yakuba	0.0					1325.5 (69.40s)
	0.1					1323.7 (24.32s)
	0.2					1304.9 (6.27s)
	0.3					1280.8 (3.73s)

Table 8 Association between genes in pairwise comparisons of the corresponding X chromosomes, considering pairwise similarities strictly greater than 0. For the three tables, the element stored in line i and column j represents the number of genes of the species i in the pairwise comparison of genomes i and j. The X chromosome of *D. pseudoobscura* was fused with another chromosome during evolution [17], therefore it presents a larger number of unassociated genes when compared to the other species.

Number of unassociated genes

species		*melanog*	*pseudoob*	*sechellia*	*simulans*	*yakuba*	*busckii*
	#genes	2043	4770	2107	2007	1956	1953
melanogaster	2043	—	152	23	84	100	221
pseudoobscura	4770	2076	—	2025	2127	2113	2110
sechellia	2107	57	174	—	102	133	257
simulans	2007	84	187	74	—	130	272
yakuba	1956	80	153	80	107	—	227
busckii	1953	167	124	173	217	201	—

Number of genes uniquely associated

species		*melanog*	*pseudoob*	*sechellia*	*simulans*	*yakuba*	*busckii*
	#genes	2043	4770	2107	2007	1956	1953
melanogaster	2043	—	1052	1440	1402	1428	1191
pseudoobscura	4770	1613	—	1651	1579	1646	1565
sechellia	2107	1479	1084	—	1454	1449	1224
simulans	2007	1382	1024	1392	—	1352	1126
yakuba	1956	1352	1017	1353	1331	—	1123
busckii	1953	1155	977	1154	1125	1147	—

Number of genes associated to at least two other genes

species		*melanog*	*pseudoob*	*sechellia*	*simulans*	*yakuba*	*busckii*
	#genes	2043	4770	2107	2007	1956	1953
melanogaster	2043	—	839	580	557	515	631
pseudoobscura	4770	1081	—	1094	1064	1011	1095
sechellia	2107	571	849	—	551	525	626
simulans	2007	541	796	541	—	525	609
yakuba	1956	524	786	523	518	—	606
busckii	1953	631	852	626	611	605	—

Fast Lightweight Accurate Xenograft Sorting

Jens Zentgraf
Bioinformatics, Computer Science XI, TU Dortmund University, Germany
`http://www.rahmannlab.de/people/zentgraf`
Jens.Zentgraf@tu-dortmund.de

Sven Rahmann
Genome Informatics, Institute of Human Genetics, University of Duisburg-Essen, Essen, Germany
`http://www.rahmannlab.de/people/rahmann`
Sven.Rahmann@uni-due.de

Abstract

Motivation: With an increasing number of patient-derived xenograft (PDX) models being created and subsequently sequenced to study tumor heterogeneity and to guide therapy decisions, there is a similarly increasing need for methods to separate reads originating from the graft (human) tumor and reads originating from the host species' (mouse) surrounding tissue. Two kinds of methods are in use: On the one hand, alignment-based tools require that reads are mapped and aligned (by an external mapper/aligner) to the host and graft genomes separately first; the tool itself then processes the resulting alignments and quality metrics (typically BAM files) to assign each read or read pair. On the other hand, alignment-free tools work directly on the raw read data (typically FASTQ files). Recent studies compare different approaches and tools, with varying results.
Results: We show that alignment-free methods for xenograft sorting are superior concerning CPU time usage and equivalent in accuracy. We improve upon the state of the art by presenting a fast lightweight approach based on three-way bucketed quotiented Cuckoo hashing. Our hash table requires memory comparable to an FM index typically used for read alignment and less than other alignment-free approaches. It allows extremely fast lookups and uses less CPU time than other alignment-free methods and alignment-based methods at similar accuracy.

2012 ACM Subject Classification Applied computing → Molecular sequence analysis; Applied computing → Bioinformatics; Theory of computation → Bloom filters and hashing; Theory of computation → Data structures design and analysis

Keywords and phrases xenograft sorting, alignment-free method, Cuckoo hashing, k-mer

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.4

Related Version This work is on bioRxiv. `https://www.biorxiv.org/content/10.1101/2020.05.14.095604v2`

Supplementary Material Our software *xengsort* is available under the MIT license at `http://gitlab.com/genomeinformatics/xengsort`. It is written in numba-compiled Python and comes with Snakemake workflows for hash table construction and dataset processing.

Funding *Sven Rahmann*: DFG SFB 876 subproject C1; Mercator Research Center Ruhr (MERCUR), project Pe-2013-0012.

1 Introduction

To learn about tumor heterogeneity and tumor progression under realistic *in vivo* conditions, but without putting human life at risk, one can implant human tumor tissue into a mouse and study its evolution. This is called a (patient-derived) xenograft (PDX). Over time, several samples of the (graft / human) tumor and surrounding (host / mouse) tissue are taken and subjected to exome or whole genome sequencing in order to monitor the changing genomic features of the tumor. This information can be used to predict the response to

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti; Article No. 4; pp. 4:1–4:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Tools for xenograft sorting and read filtering with key properties. See text for definition of operations; Lang.: programming language.

Tool	Ref.	Input	Operations	Lang.
XenofilteR	[13]	aligned BAM	filter	R
Xenosplit	[9]	aligned BAM	filter, count	Python
Bamcmp	[12]	aligned BAM	partial sort	C++
Disambiguate	[1]	aligned BAM	partial sort	Python or C++
BBsplit	[4]	raw FASTQ	partial sort	Java
xenome	[7]	raw FASTQ	count, sort	C++
xengsort	(this)	raw FASTQ	count, sort	Python + numba

different chemotherapy alternatives and to monitor treatment success or failure. A key step in such analyses is *xenograft sorting*, i.e., separating the human tumor reads from the mouse reads. A recent study [10] showed that if such a step is omitted, several mouse reads would be aligned to certain regions of the human genome (HAMA: human-aligned mouse allele) and induce false positive variant calls for the tumor; this especially concerns certain oncogenes.

Several tools have been developed for xenograft sorting, motivated by different goals and using different approaches; a summary appears below. Here we improve upon the existing approaches in several ways: By using carefully engineered k-mer hash tables, our approach is both *faster* and needs *less memory* than existing tools. By designing a new decision function, we also obtain *fewer unclassified reads* and in some cases even *higher classification accuracy*. Since we use a comprehensive reference of the genome and transcriptome, we are able to uniformly deal with genome, exome, and transcriptome data of xenografts.

Concerning related work, we distinguish alignment-based methods that work on already aligned reads (BAM files), versus alignment-free methods that directly work on short subsequences (k-mers) of the raw reads (FASTQ files). On the other hand, we do not distinguish between the type of data that the tools have been applied to (transcripts, or genomic DNA), because this does not depend so much on the tool but rather on the reference sequences used (genome, exome, set of transcripts, etc.).

Alignment-based methods scan existing alignments in BAM files and test whether each read maps better to the graft or to the host genome. Differences result from different parameter settings used for the alignment tool (often `bwa` or `bowtie2`) and from the way “better alignment” is defined by each of these tools. Alignment-free methods use a large lookup table to associate species information with each k-mer.

In Table 1, we list properties of existing tools and of *xengsort*, our implementation of the method we describe in this article. These tools support different operations: Operation “count” outputs proportions of reads belonging to each category (host, graft, etc.); operation “sort” sorts reads or alignments into different files according to origin, ideally into five categories: host, graft, both, neither, ambiguous; a “partial sort” only has three categories: host, graft, both/other; operation “filter” writes only an output file with graft reads or alignments. The sort operation is more general than the filter or partial sort operation and allows full flexibility in downstream processing. When available, the count operation is faster than counting the output of the sort operation, because it avoids the overhead of creating new BAM or FASTQ files.

XenofilteR, *Xenosplit*, *Bamcmp* and *Disambiguate* all work on aligned BAM files. This means that the reads must be mapped and aligned with a supported read mapper first (typically, ’bwa mem’) and the resulting BAM file must be sorted in a specific way required by the tool. The tool is typically a script that reads and compares the mapping scores and qualities in the two BAM files containing host and graft alignments. In principle, all of these tools do the same thing; large differences result rather from different alignment parameters than the tool itself. We therefore picked *XenofilteR* as a representative of this family, also because it performed well in a recent comparison [10].

BBsplit (part of BBTools) is special in the sense that it performs the read mapping itself, against multiple references simultaneously, based on k-mer seeds. Unfortunately, only up to approximately 1.9 billion k-mers can be indexed because of Java's array indexing limitations (up to 2^{31} elements) and a table load limit of 0.9; so BBsplit was not usable for our human-mouse index that contains approximately $4.5 \cdot 10^9 > 2^{32}$ k-mers.

The tool *xenome* [7] is similar to our approach: It is based on a large hash table of k-mers and sorts the reads into several categories (host, graft, both, neither, ambiguous). A read is classified based on its k-mer content according to relatively strict rules. We found the threading code of *xenome* to be buggy, such that the pure counting mode resulted in a deadlock and produced no output. The sorting mode produced the complete output but then did not terminate either.

Recent studies [5, 8, 10] have compared the computational efficiency of several methods, as well as the classification accuracy of these methods and the effects on subsequent variant calling after running vs. not running xenograft sorting. The results were contradictory, with some studies reporting that alignment-based tools are more efficient than alignment-free tools, and different tools achieving highest accuracy. Our interpretation of the results of [10] is that each of the existing approaches is able to sort with good accuracy and the main difference is in computational efficiency. Results about efficiency have to be interpreted with care because sometimes the time for alignment is included and sometimes not.

2 Methods

2.1 Overview

By considering all available host and graft reference sequences (both transcripts and genomic sequences of mouse and human), we build a large key-value store that allows us to look up the species of origin (host, graft or both) of each DNA/RNA k-mer that occurs in either species. A sequenced dataset (a collection of single-end or paired-end FASTQ files) is then processed by iterating over reads or read pairs, looking up the species of origin of each k-mer in a read (host, graft, both or none) and classifying the read based on a decision rule.

Our implementation of the key-value store as a three-way bucketed Cuckoo hashtable makes k-mer lookup faster than in other methods; the associated value can often be retrieved with a single random memory access. A high load factor of the hash table, combined with the technique of quotienting, ensures a low memory footprint, without resorting to approximate membership data structures, such as Bloom filters.

2.2 Key-value stores of canonical k-mers

We partition the reference genome (plus alternative alleles and unplaced contigs) and transcriptome into short substrings of a given length k (so-called k-mers); we evaluated $k \in \{23, 25, 27\}$. For each k-mer ("key") in any of the reference sequences, we store whether it occurs exclusively in the host reference, exclusively in the graft reference, or in both, represented by "values" 1, 2, 3, respectively. For the host- and graft-exclusive k-mers, we also store whether a closely similar k-mer (at Hamming distance 1) occurs in the other species (add value 4); such a k-mer is then called a *weak* (host or graft) k-mer. This idea extends

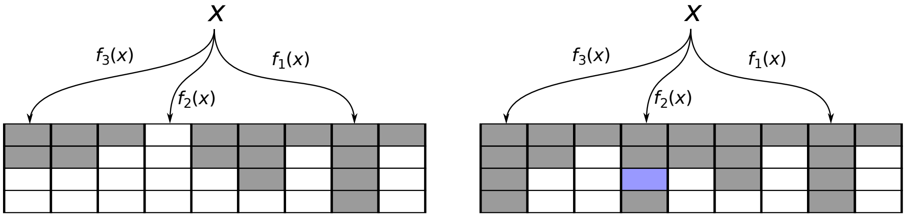


Figure 1 Illustration of (3,4) Cuckoo hashing with 3 hash functions and buckets of size 4. **Left:** Each key-value pair can be stored at one of up to 12 locations in 3 buckets. For key x, the bucket given by $f_1(x)$ is full, so bucket $f_2(x)$ is attempted, where a free slot is found. **Right:** If all hb slots are full, key x is placed into one of these slots at random (blue), and the currently present key-value pair is evicted and re-inserted into an alternative slot.

the k-mer classification of *xenome* [7], where a k-mer can be host, graft, both, or marginal, the latter category comprising both our weak host and weak graft k-mers. So we store, for each k-mer, a value from the 5-element set "host" (1), "graft" (2), "both" (3), "weak host" (5), "weak graft" (6). This value is stored using 3 bits. While a more compact base-5 representation is possible (e.g., storing 3 values with $125 < 128 = 2^7$ combinations in 7 bits instead of in 9 bits), we decided to use slightly more memory for higher speed.

To be precise, we do not work on k-mers directly, but on their canonical integer representations (*canonical codes*), such that a k-mer and its reverse complement map to the same number. We use a simple base-4 numeric encoding A $\mapsto$ 0, C $\mapsto$ 1, G $\mapsto$ 2, T/U $\mapsto$ 3, e.g. reading the 4-mer AGCG as $(0212)_4 = 38$ and its reverse complement CGCT as $(1213)_4 = 103$. The *canonical code* is then the *maximum* of these two numbers; here the canonical code of both AGCG and CGCT is thus 103. (In *xenome*, canonical k-mer codes are implemented with a more complex but still deterministic function of the two base-4 encodings; in other tools, it is often the minimum of the two encodings.) For odd k, there are exactly $c(k) := 4^k/2$ different canonical k-mer codes, so each can be stored in $2k - 1$ bits in principle. However, implementing a *fast* bijection of the set of canonical codes (which is a subset of size $c(k)$ of $\{0 \mathinner{..} (4^k - 1)\}$) to $\{0 \mathinner{..} (c(k) - 1)\}$ seems difficult, so we use $2k$ bits to store the canonical code directly, which allows faster access. We do use quotienting, described below, to reduce the size; yet in principle, one additional bit could be saved.

2.3 Multi-way bucketed quotiented Cuckoo hashing

We use multi-way bucketed Cuckoo hash table as the data structure for the k-mer key-value store. Let C be the set of canonical codes of k-mers; as explained above, we take $C = \{0 \mathinner{..} (4^k - 1)\}$, even though only half of the codes are used (for odd k). Let P be the set of locations (buckets) in the hash table and p their number; we set $P := \{0 \mathinner{..} (p - 1)\}$. Each key can be stored at up to h different locations (buckets) in the table. The possible buckets for a code are computed by h different hash functions $f_1, f_2, \dots, f_h : C \to P$. Each bucket can store up to a certain number b of key-value pairs. So there is space for $N := pb$ key-value pairs in the table overall, and each pair can be stored at one of hb locations in h buckets. Together with an insertion strategy as described below, this framework is referred to as (h, b) Cuckoo hashing. Classical Cuckoo hashing uses $h = 2$ and $b = 1$; for this work, we use $h = 3$ and $b = 4$. A visualization is provided in Figure 1. Using several hash functions and larger buckets increases the load limit; using $h = 3$ and $b = 4$ allows a load factor of over 99.9% [17, Table 1], while classical Cuckoo hashing only allows to fill 50% of the table.

Search and insert. Searching for a key-value pair works as follows. Given key (canonical code) x, first $f_1(x)$ is computed, and this bucket is searched for key x and the associated value. If it is not found, buckets $f_2(x)$ and then $f_3(x)$ are searched similarly. Each bucket access is a random memory lookup and most likely triggers a cache miss. We can ensure that each bucket is contained within a single cache line (by using additional padding bits if necessary). Then, the number of cache misses is limited to $h = 3$ for one search operation.

Because we fill the table well below the load limit (at 88% of 99.9%), we are able to store most key-value pairs in the bucket indicated by the *first* hash function f_1, and only incur a single cache miss when looking for them. Unsuccessful searches (for k-mers that are not present in either host or graft genome) will always need h memory accesses. However, one optimization is possible: If, say, the first bucket $f_1(x)$ contains an empty slot, we do not need to search further, because the insertion procedure produced a tight layout, in the sense that if a single element could be moved to an "earlier" bucket, it would have been done.

Insertion of a key-value pair works as follows. First, the key is searched as described above. If it is found, the value is updated with the new value. For example, if an existing host k-mer is to be inserted again as a graft k-mer, the value is updated to "both". If the key is not found, we check whether any of the buckets $f_1(x), f_2(x), f_3(x)$ contains a free slot. If this is the case, x and its value are inserted there. If all buckets are full, a random slot among the hb slots is picked, and the key-value pair stored there is evicted (like a cuckoo removes eggs from other birds' nests) to make room for x and its value. Then an alternative location for the evicted element is searched. This process may continue for several iterations and is called a "random walk" through the table. If the walk becomes too long (longer than 5000 steps, say), we declare that the table is too full, and construction fails and has to be restarted with a larger table or different random seed.

We require that the final size (number of buckets p) of the hash table is known in advance, so we can pre-allocate it. The genome length is a good (over-)estimate of the number of distinct k-mers and can be used. We recently presented a practical algorithm [18] to optimize the assignment of k-mers to buckets (i.e., their hash function choices) such that the average search cost of present k-mers is minimized to the provable optimum. This optimization takes significant additional time and requires large additional data structures; so we took the opportunity here to evaluate whether it significantly improves lookup times in comparison to a table filled by the above random walk strategy.

Bijective hash functions and quotienting. In principle, we need to store the $2k$ bits for the canonical k-mer code x and the 3 bits for the value at each slot. However, by using hash functions of the form $f(x) := g(x) \bmod p$, where p is the number of buckets and g is a *bijective* (randomized) transformation on the full key set $\{0 \,..\, (4^k - 1)\}$, we can encode part of x in $f(x)$: Note that from $f(x)$ and $q(x) := g(x)//p$ (integer division), we can recover $g(x) = p \cdot q(x) + f(x)$, and since g is bijective, we can recover x itself. This means that we only need to store $q(x)$, not x itself in bucket $f(x)$, which only takes $\lceil 2k - \log_2 p \rceil$ instead of $2k$ bits. However, since we have h alternative hash functions, we also need to store *which* hash function we used, using 2 bits for $h = 3$ (0 indicating that the slot is empty). This technique is known as *quotienting*. It gives higher savings for smaller buckets (for constant $N = pb$, smaller b means larger p), but on the other hand the load limit is smaller for small b. We find $b = 4$ to be a good compromise, allowing table loads of 99.9%.

For the bijective part $g(x)$, we use affine functions of the form

$$g_{a,b}(x) := [a \cdot (\mathrm{rot}_k(x) \text{ xor } b)] \bmod 4^k,$$

where rot_k performs a cyclic rotation of k bits (half the width of x), moving the "random" inner bits to outer positions and the less random outer bits (due to the max operation when taking canonical codes) inside, b is a $2k$-bit offset, and a is an odd multiplier. Picking a "random" hash function means picking random values for a and b.

▶ **Lemma 1.** *For any $2k$-bit number b and any odd $2k$-bit number a, the function $g_{a,b}$ is a bijection on $K := \{0 \mathinner{..} (4^k - 1)\}$.*

Proof. Let $y = g_{a,b}(x)$. By definition, the range of $g_{a,b}$ on K is a subset of K. Because $|K|$ is a power of 2 and a is odd, the greatest common divisor of $|K|$ and a is 1, and so there exists a unique multiplicative inverse a' of a modulo $4^k = |K|$, such that $aa' = 1 \pmod{4^k}$. The other operations (xor b, rot_k) are inverses of themselves; so we recover $x = \mathrm{rot}_k([(a' \cdot y) \bmod 4^k] \text{ xor } b)$. ◀

In summary, each stored canonical k-mer needs $2 + 3 + \lceil 2k - \log_2 p \rceil$ bits to remember the hash function choice and to store the value (species), and the quotient, respectively. For $k = 25$ and $p = 1\,276\,595\,745$ buckets, this amounts to 25 bits per k-mer, or 100 bits for each bucket of 4 k-mers. To ensure cache line aligned pages, we could insert 28 padding bits to grow the bucket size to 128 bits; however, we chose less memory for a small speed decrease, and let some buckets cross cache line boundaries.

2.4 Annotating weak k-mers

A k-mer that occurs only in the host (graft) reference, but has a Hamming-distance-1 neighbor in the graft (host) reference, is called a *weak* host (graft) k-mer. So for a weak k-mer, a single nucleotide variation could flip its assigned species, while a k-mer that is not weak is more robust in this sense. A similar concept exists in *xenome*; however, weak host and graft k-mers are combined into "marginal" k-mers, and their origin is not stored. After the hash table has been constructed with all k-mers and their values "host", "graft" or "both", we mark weak k-mers by modifying the value, setting an additional "weak" bit. In principle, we could scan over the k-mers and query all $3k$ neighbors of each k-mer, but this is inefficient.

Instead, we extract from the hash table a complete list L of k-mers and their reverse complements (not canonical codes; approx. $9 \cdot 10^9$ entries for $4.5 \cdot 10^9$ distinct k-mers), together with their current values. To save memory, this list is created and processed in 16 chunks according to the first two nucleotides of the k-mer, thus needing approx. 4.5 GB of additional memory temporarily. Since we use odd $k = 2\ell + 1$, we can partition a k-mer into its ℓ-prefix, its middle base and its ℓ-suffix. We make use of the following observation.

▶ **Observation 1.** *For $k = 2\ell + 1$, two k-mers x, y with Hamming distance 1 differ either in their ℓ-suffix, in the ℓ-suffix of their reverse complement or in their middle base.*

We thus partition the sorted list into blocks of constant $(\ell + 1)$-prefixes. Different blocks are processed independently in parallel threads. The ℓ-suffixes of all pairs of k-mers in such a block are queried with a fast bit-vector test for Hamming distance 1. If a pair is found and the k-mers occur in different species, the "weak bit" (value 4) is set. It remains to find pairs of k-mers that differ only in their middle base. We conceptually partition the list into blocks of constant ℓ-prefixes and use that such pairs must occur consecutively in a block and agree in the ℓ-suffix. So these pairs can be identified within a single linear scan. In the end, updated values are transferred to the values of the canonical k-mers in the hash table.

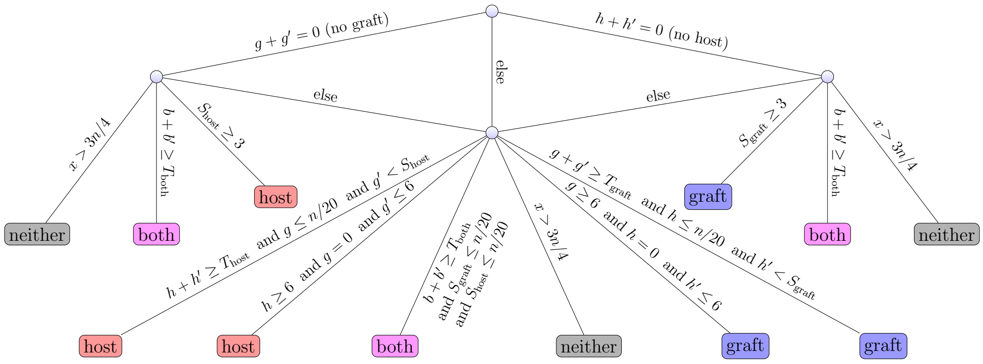

$g + g' = 0$ (no graft)
else
$h + h' = 0$ (no host)
$x > 3n/4$
$b + b' \geq T_{\text{both}}$
$S_{\text{host}} \geq 3$
else
else
$S_{\text{graft}} \geq 3$
$b + b' \geq T_{\text{both}}$
$x > 3n/4$
neither
both
host
graft
both
neither
$h + h' \geq T_{\text{host}}$ and $g \leq n/20$ and $g' < S_{\text{host}}$
$h \geq 6$ and $g = 0$ and $g' \leq 6$
$b + b' \geq T_{\text{both}}$ and $S_{\text{graft}} \leq n/20$ and $S_{\text{host}} \leq n/20$
$x > 3n/4$
$g \geq 6$ and $h = 0$ and $h' \leq 6$
$g + g' \geq T_{\text{graft}}$ and $h \leq n/20$ and $h' < S_{\text{graft}}$
host
host
both
neither
graft
graft

Figure 2 Decision rule tree for classifying a DNA fragment from k-mer statistics $(h, h', g, g', b, x; n)$, meaning number of k-mers of type "host" (h), "weak host" (h'), "graft" (g), "weak graft" (g'), "both" (b), and number of k-mers not present in the key-value store (x), respectively; n is the total number of (valid) k-mers in the fragment. We also use weighted scores $S_{\text{host}} := h + \lfloor h'/2 \rfloor$ and $S_{\text{graft}} := g + \lfloor g'/2 \rfloor$ and thresholds $T_{\text{host}} := \lfloor n/4 \rfloor, T_{\text{graft}} := \lfloor n/4 \rfloor$ and $T_{\text{both}} := \lfloor n/4 \rfloor$. A fragment is thus classified as "host", "graft", "both", "neither", or "ambiguous". Category "ambiguous" is chosen if no other rule applies and no "else" rule is present in a node.

2.5 Reference sequences

To build the k-mer hash table from genomic and transcribed sequences from human and mouse, we obtained the "toplevel DNA" genome FASTA files, which include both the primary assembly, unplaced contigs and alternative alleles, and the "all cDNA" files, which contain the known transcripts, from the ensembl FTP site, release 98.

As the alternative alleles of the human and mouse toplevel references contain mostly Ns to keep positional alignment of alternative alleles to the consensus reference, they decompress to huge FASTA files (over 60 GB for human, over 12 GB for mouse). Therefore we condensed the toplevel reference sequences by replacing runs of more than 25 Ns by 25 Ns. This does not change the k-mer content, as k-mers containing even a single N are ignored. It does provide an efficiency boost to alignment-based tools because read mappers build an index of every position in the genome and typically replace runs of Ns by random sequence.

2.6 Fragment classification

Given a sequenced fragment (single read or read pair), we query each k-mer of the fragment about its origins; k-mers with undetermined bases are ignored. Our implementation reads large chunks (several MB) of FASTQ files and distributes read classification over several threads (we found that 8 threads saturate the I/O).

We collect k-mer statistics for each fragment (adding the numbers of both reads for a read pair): Let n be the number of (valid) k-mers in the fragment. Let h be the number of host k-mers and h' the number of weak host k-mers, and analogously define g and g' for the graft species. Further, let b be the number of k-mers occuring in both species, and let x be the number of k-mers that were not found in the key-value store. Based on the vector $(h, h', g, g', b, x; n)$, we use a tree of hierarchical rules to classify the fragment into one of five categories: "host", "graft", "both", "neither" and "ambiguous". Categories "host" and "graft" are for reads that can be clearly assigned to one of the species. Category "both" is for reads that match equally well to both references. Category "neither" is for reads that contain many k-mers that cannot be found in the key-value store; these could point

Table 2 Properties of the k-mer index for different values of k (wk: weak). Underlying reference sequences are given in Section 2.5.

k-**mers**	$k = 23$	(%)	$k = 25$	(%)	$k = 27$	(%)
total	4 396 323 491	(100)	4 496 607 845	(100)	4 576 953 994	(100)
host	1 924 087 512	(43.8)	2 050 845 757	(45.6)	2 105 520 461	(46.0)
graft	2 173 923 063	(49.4)	2 323 880 612	(51.7)	2 395 147 724	(52.3)
both	18 701 862	(0.4)	12 579 160	(0.3)	9 627 252	(0.2)
wk host	132 469 231	(3.0)	52 063 110	(1.2)	32 445 717	(0.7)
wk graft	147 141 823	(3.4)	57 239 206	(1.3)	34 212 840	(0.7)

to technical problems (primer dimers) or contamination of the sample with other species. Finally, category "ambiguous" is for reads that provide conflicting information. Such reads should not usually be seen; they could result from PCR hybrids between host and graft during library preparation. The precise rules are shown in Figure 2. Category "ambiguous" is chosen if no "else" rule exists and no other rule applies in any given node.

Quick mode. Inspired by a similar acceleration in the *kallisto* software [3] for transcript expression quantification, we additionally implemented a "quick mode" that initially looks only at the type of the third and third-last k-mer in every read. If the two (for single-end reads) or four (for paired-end reads) types agree (e.g. all are "graft"), the fragment is classified on this sampled evidence alone. This results in quicker processing of large FASTQ files, but only considers a small sample of the available information.

3 Results

We evaluate our alignment-free xenograft sorting approach and its implementation *xengsort* for the common case of human-tumor-in-mouse xenografts, by using mouse datasets, human datasets, xenograft datasets and datasets from other species, and compare against an existing tool with the same purpose, *xenome* from the *gossamer* suite [7], and against a representative of alignment-based filtering tools, *XenofilteR* [13]. The hardware used for the benchmarks was one server with two AMD Epyc 7452 CPUs (with 32 cores and 64 threads each), 1024 GB DDR4-2666 memory and one 12 TB HDD with 7200 rpm and 256 MB cache.

3.1 Hash table construction

Table size and uniqueness of k-mers. We evaluated $k \in \{23, 25, 27\}$ and then decided to use $k = 25$ because it offers a good compromise between species specificity and memory requirements. Table 2 shows several index properties. In particular, moving from $k = 25$ to $k = 27$, the small decrease in k-mers that map to both genomes and in weak k-mers did not justify the additional memory requirements. In addition, longer k-mers lead to lower error tolerance against sequencing errors, as each error affects up to k of the k-mers in a read.

Construction time and memory. Table 3 shows time and memory requirements for building the k-mer hash table or FM index for *bwa* (for *XenofilteR*). The main difference is that the BWA index is a succinct representation of the suffix array of the references and not a k-mer hash table. Our hash table construction is not paralellized; hence CPU times and wall clock times agree and are less than one hour. The hash construction of *xenome* is paralellized; we gave it 8 threads (but 9 were sometimes used); yet it does about 20 times the CPU work and takes three times as long as *xengsort*, even when using multiple threads.

Table 3 Index construction: CPU times and wall clock times in minutes and memory in Gigabytes using different tools and different k-mer sizes for *xengsort*. "Build" times refer to collecting and hashing the k-mers according to species, but without marking weak k-mers. "Mark" times refer to marking weak k-mers. "Total" times are the sum of build and mark times, plus additional I/O times. "CPU" times measure total CPU work load (as reported by the time command as user time), and "wall" times refer to actually passed time. Final size ("mem final") is measured by index size on disk (GB). Memory peak ("mem peak") is the highest memory usage during construction (GB).

		build	**build**	**mark**	**mark**	**total**	**total**	**mem**	**mem**
tool	k	**CPU**	**wall**	**CPU**	**wall**	**CPU**	**wall**	**final**	**peak**
xengsort	23	50	50	591	176	641	226	12.8	17.3
xengsort	25	53	53	437	158	490	211	15.9	20.4
xengsort	27	51	51	495	214	546	265	17.3	21.8
xenome	25	992	151	2338	356	3626	552	31.2	57.1
XenofilteR	–	528	658	–	–	528	658	13.0	22.0

Marking weak or marginal k-mers is paralellized in both approaches; wall clock times are measured using 8 threads. Again, *xengsort* finds the weak k-mers faster, both in terms of total CPU work and wall clock time.

The indexing method of bwa is not comparable, as it builds a complete suffix array (FM index) that is independent of k and does not include marking weak k-mers. Here the CPU time is lower than the wall clock time, which indicates an I/O starved process.

We note that *xenome* uses a large amount of memory during hash table construction (it was given up to 64 GB). It works with less if restricted, but at the expense of longer running times. BWA indexing also needs significant additional memory during construction. The additional memory required by *xengsort* results from the additional sorted k-mer list required for detecting weak k-mers. Overall, our construction is fast (even though serial only) and uses a reasonable amount of memory.

Load factor and hash choice distribution. As explained in Section 2.3, 3-way Cuckoo hash tables support very high loads (fill ratios) over 99.9%. However, such loads come at the expense of distributing all k-mers almost evenly across hash function choices. For faster lookup, it is beneficial to leave part of the hash table empty. We used a load factor of 88% and thus find 76.7% of the k-mers at their first bucket choice, 15.5% at their second choice and only 7.8% at their third choice, yielding an average of 1.31 lookups for a present k-mer.

Applying assignment optimization [18], which takes an additional 5 hours (serial CPU time, not parallelized) and needs over 80 GB of RAM, we achieve a slightly better average of 1.17 lookups for a present k-mer.

3.2 Classification results

We applied our method *xengsort*, *xenome* and *XenofilteR* to several datasets with reads of known origin (except possible contamination issues or technical artefacts), that however present certain particular challenges. A summary of running times for all datasets appears in Table 4.

Human-captured mouse exomes. A recent comparative study [10] made five mouse exomes accessible, which were captured with a human-exome capture kit and hence presents mouse reads that are biased towards high similarity with human reads. The mouse strains were

Table 4 Dataset sizes (number of fragments; M: millions) and CPU times in minutes spent on different datasets, measured with the "time" command (user time) when running with 8 threads (*xenome*, *xengsort*, *bwa-mem*, BAM sorting, except for *XenofilteR* (XfR), which is single-threaded). N/A: not applicable; tool could not be run on this dataset.

dataset / tool	size	XfR +	*bwa* +	sort	*xenome*	*xengsort*
mouse exomes	307 M	310 +	8291 +	179	1823	368
human matepair	1258 M	N/A +	222939 +	940	9845	2463
chicken genome	251 M	76 +	6976 +	118	1273	592
leukemia RNA	1760 M	778 +	22111 +	521	5188	1680
PDX RNA	9742 M	16043 +	278329 +	5862	59692	13555

Table 5 Detailed classification results on five human-captured mouse exomes from different mouse strains (2× A/J, 1× BALB/c, 2× C57BL/6). Running times are reported both in CPU minutes [Cm], measuring CPU work, and wall clock minutes [Wm], measuring actual time spent. Times for *XenofilteR* (XfR) do not include alignment or BAM sorting time. Classification results report the number and percentage (in brackets) of fragments classified as mouse (correct), both human and mouse (likely correct), human (incorrect), ambiguous (no statement) and neither (likely incorrect). *XenofilteR* (XfR) only extracts human fragments and does not classify the remainder; so only the number of fragments classified as human are reported.

A/J-1	*xengsort*		*xenome*		XfR	
time	70 Cm	14 Wm	371 Cm	45 Wm	56 Cm	56 Wm
	fragmets	(%)	fragmets	(%)	fragmets	(%)
mouse	46 648 014	(78.03)	45 759 814	(76.54)		
both	120 808	(0.20)	65 269	(0.11)		
human	12 813 583	(21.43)	12 500 844	(20.91)	6 315 955	(10.56)
ambgs.	58 449	(0.10)	1 383 547	(2.31)		
neither	143 775	(0.24)	75 155	(0.13)		

A/J-2	*xengsort*		*xenome*		XfR	
time	70 Cm	15 Wm	416 Cm	50 Wm	67 Cm	67 Cm
	fragmets	(%)	fragmets	(%)	fragmets	(%)
mouse	60 255 189	(95.57)	59 135 489	(93.80)		
both	151 396	(0.24)	89 089	(0.14)		
human	2 301 384	(3.65)	2 271 131	(3.60)	1 718 545	(2.73)
ambgs.	57 827	(0.09)	1 340 814	(2.13)		
neither	279 556	(0.44)	208 829	(0.33)		

BALB/c	*xengsort*		*xenome*		XfR	
time	68 Cm	15 Wm	392 Cm	45 Wm	61 Cm	61 Wm
mouse	62 235 960	(98.99)	61 274 277	(97.46)		
both	118 541	(0.19)	68 949	(0.11)		
human	342 908	(0.55)	348 154	(0.55)	285 556	(0.45)
ambgs.	45 063	(0.07)	1 098 036	(1.65)		
neither	127 035	(0.20)	80 091	(0.13)		

C57BL/6-1	*xengsort*		*xenome*		XfR	
time	72 Wm	14 Wm	359 Wm	44 Wm	58 Cm	58 Wm
mouse	57 993 361	(98.93)	57 522 446	(98.13)		
both	118 984	(0.20)	74 325	(0.13)		
human	375 716	(0.64)	376 653	(0.64)	290 894	(0.50)
ambgs.	27 731	(0.05)	571 542	(0.98)		
neither	103 895	(0.18)	74 721	(0.13)		

C57BL/6-2	*xengsort*		*xenome*		XfR	
time	67 Cm	15 Wm	422 Cm	51 Wm	62 Cm	62 Wm
mouse	62 384 448	(99.00)	61 941 783	(98.30)		
both	107 019	(0.17)	66 163	(0.10)		
human	189 536	(0.30)	208 149	(0.33)	132 535	(0.21)
ambgs.	27 142	(0.04)	562 659	(0.89)		
neither	304 677	(0.48)	234 068	(0.37)		

A/J (two mice), BALB/c (one mouse), and C57BL6 (two mice); they were sequenced on the Illumina HiSeq 2500 platform, resulting in 11.8 to 12.7 Gbp. The datasets are available under accession numbers SRX5904321 (strain A/J, mouse 1), SRX5904320 (strain A/J, mouse 2), SRX5904319 (strain BALB/c, mouse 1), SRX5904318 (strain C57BL/6, mouse 1) and SRX5904322 (strain C57BL/6, mouse 2).

Ideally, all reads should be classified as mouse reads.

Table 5 shows detailed classification results and running times. Considering the BALB/c and C57BL/6 strains first, it is evident that classification accuracy is high (over 98.9% mouse for *xengsort*, over 97.4% for *xenome*; with less than 0.64% human reads for both tools). The main difference between the tools is that *xenome* is more conservative, assigning a larger fraction of reads to the "ambiguous" (unclassified) category. With *xenome*, this happens for reads that contain two k-mers x, y, where x maps uniquely to human and y maps uniquely to mouse. The decision rule of *xengsort* is more permissive and tolerant towards small inconsistencies. Therefore, *xengsort* assigns more reads correctly to mouse, and fewer to the ambiguous category. Additionally, *xengsort* assigns fewer reads incorrectly to human.

However, the two samples of strain A/J give different results. Both *xengsort* and *xenome* assign a large fraction of reads (around 21% and 3.6% in the two samples) to the human genome, while *XenofilteR* assigns only 10.5% and 2.7%, respectively. While *xengsort* does assign more reads to mouse, it also assigns more reads to human, following its strategy of leaving fewer reads unassigned (ambiguous). Inspection of these reads revealed that almost all of them are low-complexity, i.e. consist of repetitive sequence, and a check with BLAT [11] revealed no hits in mouse and several gapped hits in the human genome. So the classification as human reads is not incorrect from a technical standpoint, but in fact these reads appear to point to techincal problems during then enrichment step of the library generation. An additional low-complexity filter would remove most problematic reads.

Concerning running times, we find that *xengsort* needs around 70 CPU minutes for one of these datasets, and less than 15 minutes of wall clock time using 8 threads. The speed-up being less than 8 results from serial intermediate I/O steps. While *xenome* makes better use of parallelism, it is slower overall, requiring 5 to 6 times the CPU work of *xenome*. For only scanning already aligned BAM files, *XenofilteR* is surprisingly slow, and we see that we can sort the reads from scratch in almost the same amount of CPU work that is required to compare alignment scores. When adding bwa mem alignment times (even without the time required for sorting the resulting BAM files), *XenofilteR* needs an additional 887 to 1424 CPU minutes for the human alignments and an additional 424 to 777 minutes for the mouse alignments per dataset, making the alignment-based approach far less efficient than the alignment-free approach.

Human genome (GIAB) matepair library. We obtained FASTQ files of an Illumina-sequenced 6kb matepair library from the Genome In A Bottle (GIAB) Ashkenazim trio dataset according to the provided sequence file index (`ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data_indexes/AshkenazimTrio/sequence.index.AJtrio_Illumina_6kb_matepair_wgs_08032015`). The data represents a family (mother, father, son). Ideally, we see only human reads.

Figure 3A shows the classification results for *xengsort* and *xenome*. *XenofilteR* reported that the BAM files were too large to be processed and did not give a result (400 GB total for human and mouse; each BAM file over 30 GB in size). We see that almost all reads are correctly identified as human, while a small fraction is neither, which could be adapter dimers or other technical issues. However, *xenome* classifies a similarly small fraction as ambiguous. We observe the same wall clock time ratio (about 3.5) between *xenome* and *xengsort* as for the mouse exome dataset.

Because this is a very large dataset (112 GB gzipped FASTQ), we additionally evaluated the effects of using *xengsort*'s "quick mode". We observed a significant reduction in processing time (by about 33%) and almost unchanged classification results. We also ran the *xengsort* classification with the optimized hash table (using an optimized assignment computed using the methods from [18] and found a small reduction (9%) in running time.

We conclude that both alignment-free tools accurately recognize that this is a pure human dataset, and that *xengsort* is again more CPU-efficient and faster, given the same resources.

Chicken genome. We obtained a paired-end (2x101bp) Illumina whole genome sequencing run of a chicken genome from a whole blood sample (accession SRX6911418) with a total of 251 million paired-end reads. Ideally, none of these reads are recognized as mouse or human reads. Figure 3B shows divergent results. For *XenofilteR*, we can only say that almost no reads are extracted as human; the remainder is unclassified. *Xenome* assigns a small number of reads to each category and only around 90% into the "neither" category, while *xengsort* assigns 98.11% of the reads as "neither".

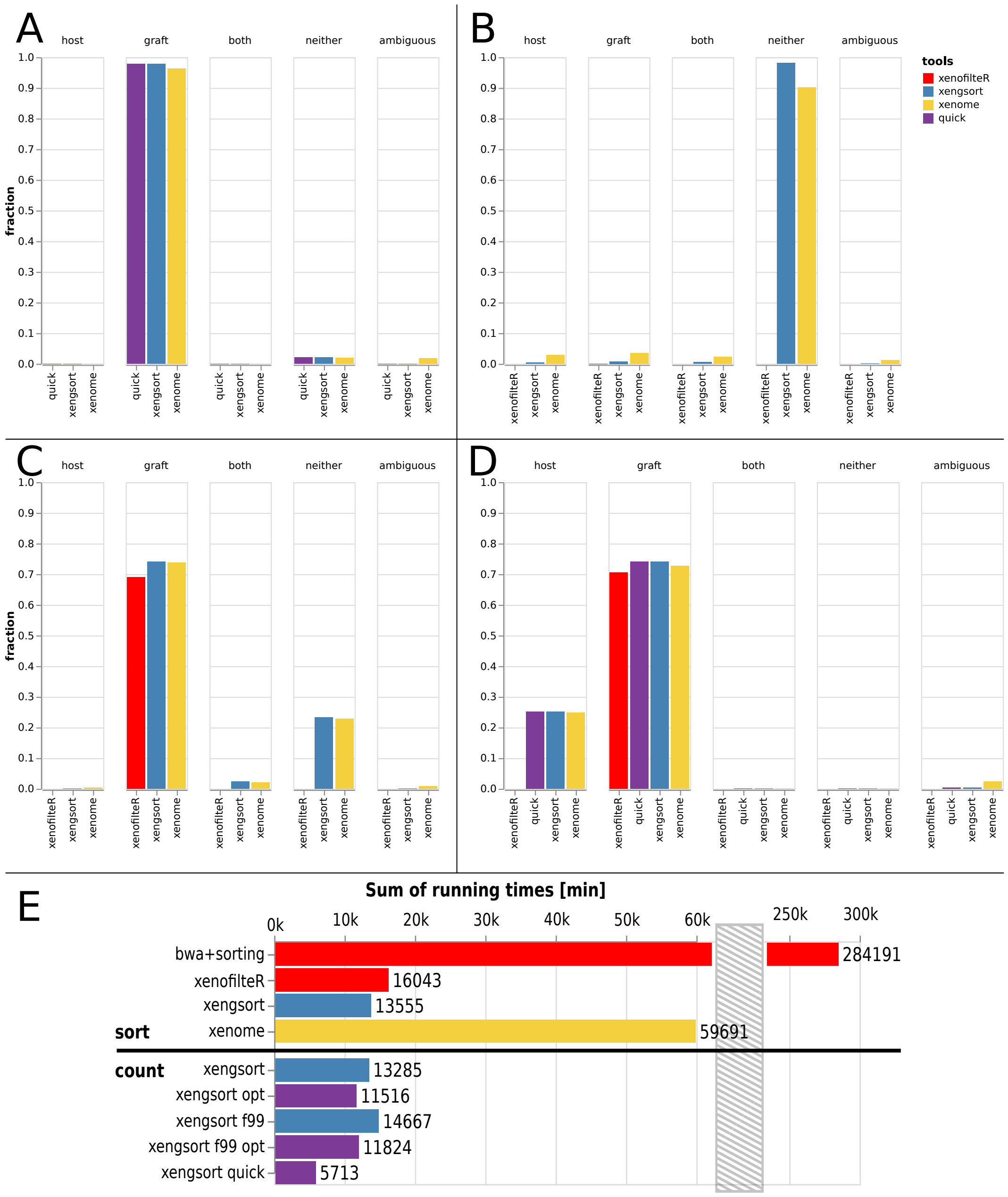


Figure 3 Classification results of different tools (*XenofilteR*, *xenome*, *xengsort*, and partially *xengsort* with “quick” option) on several datasets: A: GIAB human matepair dataset (*XenofilteR* did not run on this dataset); B: Chicken genome; C: Human lymphocytic leukemia RNA-seq data; D: Patient-derived xenograft (PDX) RNA-seq data. E: CPU times on the PDX RNA-seq dataset with different tools and different *xengsort* parameters (see text).

Concerning running time (cf. Table 4), the scan of *XenofilteR* here beats the alignment-free tools because both BAM files are essentially empty, as very few reads align against human or mouse. Also, the speed advantage of *xengsort* over *xenome* is less on this dataset, mainly because most k-mers are not found in the index and require $h = 3$ memory lookups and likely cache misses. Such a dataset that contains neither graft nor host reads is aversarial for our design of *xengsort*; it is also unlikely to be encountered in practice.

Human lymphocytic leukemia tumor RNA-seq data. We obtained single-end FASTQ files from RNA-seq data of 5 human T-cell large granular lymphocytic leukemia samples, where recurrent alterations of TNFAIP3 were observed, and 5 matched controls (13.4 Gbp to 27.5 Gbp). The files are available from SRA accession SRP059322 (datasets SRX1055051 to SRX1055060). Surprisingly, not all fragments were recognized as originating from human tissue (Figure 3C). While *xenome* and *xengsort* agreed that the human fraction is close to 75%, *XenofilteR* assigned considerably fewer reads to human origins (less than 70%).

For this and other RNA-seq datasets, we trimmed the Illumina adapters using cutadapt [15] prior to classification, as some RNA fragments may be shorter than the read length. If this step is omitted, even fewer fragments are classified as human (graft): just below 70% for *xenome* and *xengsort*, and only about 53% for *XenofilteR*. The number of fragments classified as neither increases correspondingly.

We investigated the reads classified by *xengsort* as neither human nor mouse. Quality control with FastQC [2] revealed nothing of concern, but showed an unusual biomodal per-fragment GC content distribution with peaks at 45% and 55%. BLASTing the fragments against the non-redundant nucleotide database [6] yielded no hits at all for 97% of these fragments. A small number (2%) originated from the bacteriophage PhiX, which was to be expected, because it is a typical spike-in for Illumina libraries. The remaining 1% of fragments showed random hits over many species without a distinctive pattern. We therefore concluded that the neither fragments mainly consisted of artefacts from library construction, such as ligated and then sequenced random primers.

Concerning running times (Table 4), we observed again that *xengsort* is more than 3 times faster than *xenome*and that *xengsort* needs time comparable to *XenofilteR*even when only the time for sorting and scanning existing BAM files is taken into account. Producing the alignments takes much longer.

Patient-derived xenograft (PDX) RNA-seq samples from human pancreatic tumors. We evaluated 174 pancreatic tumor patient-derived xenograft (PDX) RNA-seq samples that are available internally at University Hospital Essen. Figure 3D shows that all three tools classify between 70% and 74% as graft (human) fragments. Again, *XenofilteR* seems to be the most conservative tool with about 70%, and *xenome* classifies about 72% as human and *xengsort* 74%. The remaining reads are not classified by *XenofilteR*, while *xenome* and *xengsort* both assign about 25% to host (mouse). Furthermore, *xenome* classifies about 2% and *xengsort* less than 1% as ambiguous. So we observe that on all datasets, *xengsort* is more decisive than *xenome* and, judging from the pure human and mouse datasets, mostly correct about it. Because this is a large dataset, we also applied *xengsort*'s quick mode and found essentially no differences in classification results (less than 0.001 percentage points in each class; e.g. for graft: quick 74.0111% vs. standard 74.0105% of all reads; difference 0.0006%; cf. Fig. 3D).

Concerning running time, Figure 3E shows that the alignment using *bwa-mem* and the sorting of the BAM file for *XenofilteR* took over 284 191 CPU minutes (close to 200 days). After that, *XenofilteR* required an additional 16 043 CPU minutes (over 11 days) to classify the aligned and sorted reads. In comparison, *xenome* with 59 691 CPU minutes (41.5 days) took only 20% of the time used by *bwa-mem* and *XenofilteR*, and *xengsort* needed 13 555 CPU minutes (9.5 CPU days) to sort all reads and is therefore even faster than the classification by *XenofilteR* alone, even excluding the alignment and sorting steps, and over 4 times faster than *xenome*. Using the "quick mode" with an optimized hash table at 88% load needed only 5 713 CPU minutes (less than 4 CPU days), i.e., less than half of the time of a full analysis.

We additionally examined some trade-offs for this dataset. First, we note that only counting proportions without output ("count" operation) is not much faster than sorting the reads into different output files ("sort" operation): 13285 vs. 13555 CPU minutes (2% faster). We additionally measured the running time of the *xengsort*'s count operation on hash tables with different load factors (88% and 99%) using both the standard assignment by random walk and an optimal assignment [18]. As expected, a load factor of 99% was slower than 88% (by 10.4% on the random walk assignment, but only by 2.6% on the optimized assignment). Using the optimal assignment gives a speed boost (13.3% faster at 88% load; 19.3% at 99% load). The optimized assignment at 99% load yields an even faster running time than the random walk assignment at 88% load by 11% (11 824 vs. 13 285 CPU minutes).

4 Discussion and Conclusion

We revisited the xenograft sorting problem and improved upon the state of the art in alignment-free methods with our implementation of *xengsort*.

On typical datasets (PDX RNA-seq), it is at least four times faster and needs less memory than the comparable *xenome* tool. Our experiments show that it provides accurate classification results, and classifies more reads than *xenome*, which more often bails out when uncertain. Surprisingly, on PDX datasets, our approach is even faster than scanning already aligned BAM files. This favorable behavior arises because almost every k-mer in every read can be expected to be found in the key-value store, and lookups of present keys are faster than lookups of absent keys with our data structure.

On adversarial datasets (e.g., a sequenced chicken genome, where almost none of the k-mers can be found in the hash table), *xengsort* is 2 times faster than *xenome* and about 8 times slower than scanning pre-aligned and pre-sorted BAM files (which are mostly empty).

However, given that producing and sorting the BAM files takes significant additional time, especially for computing the (non-existing) alignments, our results show that overall, alignment-free methods require significantly less computational resources than alignment-based methods. In view of the current worldwide discussions on climate change and energy efficiency, we advocate that the most resource-efficient available methods should be used for a task, and we propose that *xengsort* is preferable to existing work in this regard. Even though one could argue that alignments are needed later anyway, we find that this is not always true: First, to analyze PDX samples, typically only the graft reads are further considered and need to be aligned. Second, recent research has shown that more and more application areas can be addressed by alignment-free methods, even structural variation and variant calling [16], so alignments may not be needed at all.

On the methodological side, we developed a general key-value store for DNA/RNA k-mers that allows extremely fast lookups, often only a single random memory access, and that has a low memory footprint thanks to a high load factor and the technique of quotienting.

Thus this work might be seen as a blueprint for implementations of other alignment-free methods (for gene expression quantification, metagenomics, etc.). In principle, one could replace the underlying key-value store of each published k-mer based method by the hashing approach presented here and probably obtain a speed-up of factor 2 to 4, while at the same time saving some space for the hash table. In practice, such an approach may be difficult because the code in question is often deeply nested in the application. However, we would like to suggest that for future implementations, three-way bucketed Cuckoo hash tables with quotienting should be given serious consideration.

A (small) limitation of our approach is that the size of the hash table must be known (at least approximately) in advance. (Growing it would mean re-hashing everything). Fortunately, the total length of the sequences in the k-mer key-value store provides an easily calculated upper bound. The advantage of such a static approach is that only little additional memory is required during construction.

The software *xengsort* is available at `http://gitlab.com/genomeinformatics/xengsort` under the MIT license. Installation and usage instructions are provided within the README file of the repository. The software is written in Python, but makes use of just-in-time compilation at runtime using the numba package [14]. While requiring an additional 1–2 seconds of startup time, this allows for many optimizations, because certain parameters that become only known at run time, such as random parameters for the hash functions, can be compiled as constants into the code. These optimizations yield savings that can exceed the initial compilation effort.

Further variants of our approach can be explored and evaluated: We already introduced a “quick mode”, similar to the one in kallisto [3], that is faster, but may falsely classify problematic (amiguous) reads as belonging to a specific species. In practice, this does not appear to be a problem. In the future, we may alternatively reduce the number of k-mer lookups by not examining every k-mer, but only minimizers in windows of fixed size, using min-hashing or other sampling methods. Another alternative is to base the classification not on the number of (overlapping) k-mers belonging to each species, but on the number of basepairs covered by k-mers of each species. Such investigations are ongoing.

While we have indications that classification results agree well overall among all methods and variants, we concur with a recent study [10] that there exist subtle differences, whose effects can propagate through computational pipelines and influence, for example, variant calling results downstream, and we believe that further evaluation studies are necessary. In contrast to their study, we however suggest that a best practice workflow for PDX analysis should start (after quality control and adapter trimming on RNA-seq data) with *alignment-free* xenograft sorting, followed by aligning the graft reads and the reads that can originate from both genomes to the graft genome. In any workflow, the latter reads, classified as “both”, may pose problems, because one may not be able to decide the species of origin. Indeed, ultraconserved regions of DNA sequence exist between human and mouse. In this sense we believe that full read sorting (into categories host, graft, both, neither, ambiguous, as opposed to extracting graft reads only) gives the highest flexibility for downstream steps and is prefereable to filter-only apporaches.

References

1 M. J. Ahdesmäki, S. R. Gray, J. H. Johnson, and Z. Lai. Disambiguate: An open-source application for disambiguating two species in next generation sequencing data from grafted samples. *F1000Res*, 5:2741, 2016.

2 Simon Andrews. FastQC: A quality control tool for high throughput sequence data, 2010. URL: `http://www.bioinformatics.babraham.ac.uk/projects/fastqc/`.

3 N. L. Bray, H. Pimentel, P. Melsted, and L. Pachter. Near-optimal probabilistic RNA-seq quantification. *Nat. Biotechnol.*, 34(5):525–527, May 2016. Erratum in Nat. Biotechnol. 34(8):888 (2016).

4 Brian Bushnell. BBsplit, 2014–2020. Part of BBTools, `https://jgi.doe.gov/data-and-tools/bbtools/`.

5 M. Callari, A. S. Batra, R. N. Batra, S. J. Sammut, W. Greenwood, H. Clifford, C. Hercus, S. F. Chin, A. Bruna, O. M. Rueda, and C. Caldas. Computational approach to discriminate human and mouse sequences in patient-derived tumour xenografts. *BMC Genomics*, 19(1):19, 2018.

6 C. Camacho, G. Coulouris, V. Avagyan, N. Ma, J. Papadopoulos, K. Bealer, and T. L. Madden. BLAST+: architecture and applications. *BMC Bioinformatics*, 10:421, December 2009.

7 T. Conway, J. Wazny, A. Bromage, M. Tymms, D. Sooraj, E. D. Williams, and B. Beresford-Smith. Xenome–a tool for classifying reads from xenograft samples. *Bioinformatics*, 28(12):i172–i178, June 2012.

8 W. Dai, J. Liu, Q. Li, W. Liu, Y. X. Li, and Y. Y. Li. A comparison of next-generation sequencing analysis methods for cancer xenograft samples. *J Genet Genomics*, 45(7):345–350, 2018.

9 Gnöknur Giner. XenoSplit, 2019. Unpublished; source code available at `https://github.com/goknurginer/XenoSplit`.

10 S. Y. Jo, E. Kim, and S. Kim. Impact of mouse contamination in genomic profiling of patient-derived models and best practice for robust analysis. *Genome Biology*, 20(1):Article 231, November 2019. URL: `https://europepmc.org/article/med/31707992`.

11 W. J. Kent. BLAT–the BLAST-like alignment tool. *Genome Res.*, 12(4):656–664, April 2002.

12 G. Khandelwal, M. R. Girotti, C. Smowton, S. Taylor, C. Wirth, M. Dynowski, K. K. Frese, G. Brady, C. Dive, R. Marais, and C. Miller. Next-generation sequencing analysis and algorithms for PDX and CDX models. *Mol. Cancer Res.*, 15(8):1012–1016, August 2017.

13 R. J. C. Kluin, K. Kemper, T. Kuilman, J. R. de Ruiter, V. Iyer, J. V. Forment, P. Cornelissen-Steijger, I. de Rink, P. Ter Brugge, J. Y. Song, S. Klarenbeek, U. McDermott, J. Jonkers, A. Velds, D. J. Adams, D. S. Peeper, and O. Krijgsman. XenofilteR: computational deconvolution of mouse and human reads in tumor xenograft sequence data. *BMC Bioinformatics*, 19(1):366, October 2018.

14 Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: a LLVM-based python JIT compiler. In Hal Finkel, editor, *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC, LLVM 2015*, pages 7:1–7:6. ACM, 2015. `doi:10.1145/2833157.2833162`.

15 Marcel Martin. Cutadapt removes adapter sequences from high-throughput sequencing reads. *EMBnet.journal*, 17(1):10–12, May 2011. `doi:10.14806/ej.17.1.200`.

16 D. S. Standage, C. T. Brown, and F. Hormozdiari. Kevlar: A mapping-free framework for accurate discovery of de novo variants. *iScience*, 18:28–36, July 2019.

17 Stefan Walzer. Load thresholds for cuckoo hashing with overlapping blocks. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 102:1–102:10. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. `doi:10.4230/LIPIcs.ICALP.2018.102`.

18 Jens Zentgraf, Henning Timm, and Sven Rahmann. Cost-optimal assignment of elements in genome-scale multi-way bucketed cuckoo hash tables. In *Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX) 2020*, pages 186–198. SIAM, 2020. `doi:10.1137/1.9781611976007.15`.

Phyolin: Identifying a Linear Perfect Phylogeny in Single-Cell DNA Sequencing Data of Tumors

Leah L. Weber
Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL
leahlw2@illinois.edu

Mohammed El-Kebir[1]
Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL
melkebir@illinois.edu

Abstract

Cancer arises from an evolutionary process where somatic mutations occur and eventually give rise to clonal expansions. Modeling this evolutionary process as a phylogeny is useful for treatment decision-making as well as understanding evolutionary patterns across patients and cancer types. However, cancer phylogeny inference from single-cell DNA sequencing data of tumors is challenging due to limitations with sequencing technology and the complexity of the resulting problem. Therefore, as a first step some value might be obtained from correctly classifying the evolutionary process as either linear or branched. The biological implications of these two high-level patterns are different and understanding what cancer types and which patients have each of these trajectories could provide useful insight for both clinicians and researchers. Here, we introduce the Linear Perfect Phylogeny Flipping Problem as a means of testing a null model that the tree topology is linear and show that it is NP-hard. We develop Phyolin and, through both *in silico* experiments and real data application, show that it is an accurate, easy to use and a reasonably fast method for classifying an evolutionary trajectory as linear or branched.

2012 ACM Subject Classification Applied computing → Molecular evolution

Keywords and phrases Constraint programming, intra-tumor heterogeneity, combinatorial optimization

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.5

Supplementary Material Code and data are available at `https://github.com/elkebir-group/phyolin`.

Funding *Mohammed El-Kebir*: This work was supported by the National Science Foundation under award number CCF 1850502.

Acknowledgements This work was a project in the course CS598MEB (Computational Cancer Genomics, Spring 2020) at UIUC. We thank the students in this course for their valuable feedback.

1 Introduction

The clonal theory of cancer states that tumors arise from the accumulation of somatic mutations in a population of cells [18]. This process leads to a tumor comprised of heterogeneous clones – groups of cells with similar genotypes – or what is commonly referred to as intra-tumor heterogeneity. By performing bulk and/or single-cell DNA sequencing of a heterogeneous tumor biopsy, researchers and clinicians may infer reasonable models of this evolutionary process for important downstream analysis and clinical decision-making. Specifically, the evolution of a tumor is represented by a phylogeny, i.e. a rooted tree where the leaves of the tree represent the extant cells of the tumor, internal vertices represent

[1] Corresponding author

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti; Article No. 5; pp. 5:1–5:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

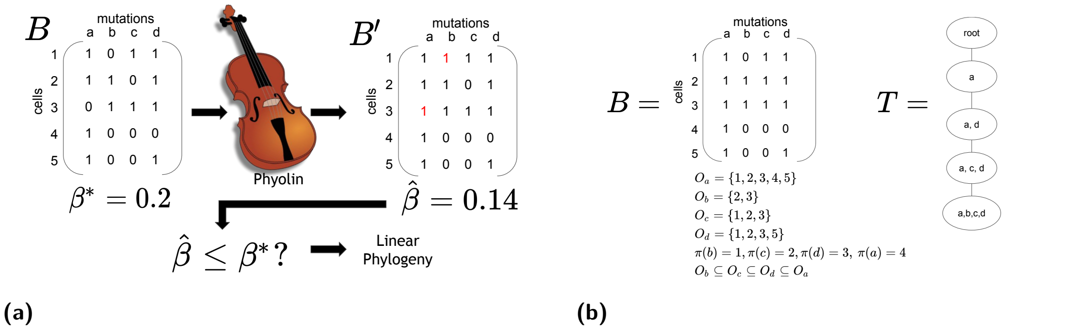


Figure 1 Phyolin identifying a linear perfect phylogeny in single-cell DNA sequencing data (a) A graphical depiction of Phyolin identifying a linear perfect phylogeny in single-cell DNA sequencing data when given a binary matrix B and a false negative rate β^*. (b) An example of error free single-cell data that represents a linear perfect phylogeny and the equivalent clonal tree representation.

ancestral tumor cells, and the root represents a normal cell. Due to trade-offs between the two data types, techniques for phylogeny inference have been developed for both bulk-sequencing and single-cell data individually [4–6,13,20] as well as combined for joint inference [15,16,25]. Bulk-sequencing data is less costly than single-cell data but results in a set of plausible phylogenies making it difficult to uniquely determine the true evolutionary history of a tumor [7,19]. Conversely, single-cell data allows high resolution of the evolutionary process but is subject to high rates of sequencing errors and is more expensive than bulk-sequencing. In particular, single-cell sequencing has a high false negative rate, as much as 40% [8], implying that actual mutations present in a cell might not be indicated correctly in the resulting data. Doublets, where multiple cells are simultaneously sequenced as a single cell, are also a unique challenge of single-cell data.

One important open question is whether certain types or subtypes of cancers follow specific evolutionary patterns. Since tumors are typically biopsied at only a single point in time for reasons related to patient care, there does not yet exist sufficient longitudinal data to fully answer this question. However, it is believed that there are four high-level categories of tumor evolution: linear evolution, branching evolution, neutral evolution and punctuated evolution [3]. Of these four types, the simplest is linear evolution and will be the focus of this work. Linear evolution results when subsequent driver mutations develop a strong selective advantage and outcompete other clones during a clonal expansion [3]. By contrast, in branching evolution, a clone can diverge into separate lineages resulting in distinct branches and a tree-like model of evolution.

A useful first step in gaining insight into the evolutionary patterns of different cancer types is to determine the likelihood of each evolutionary process under available sequencing data. Suppose, we are given single-cell data in the form of a matrix where each row in the data is a cell and each column is a single-nucleotide variant (SNV), hereafter referred to as mutation. The entries in the data would then be either 1 or 0 indicating the presence of a mutation in a particular cell. Suppose also that we assume a null model of linear evolution and are given a false negative rate for the technology under which the single-cell sequencing was performed. We could then determine the minimum number of changes from 0 to 1, indicating the entry was a false negative, such that the data is representative of a linear perfect phylogeny. This would then provide an estimate of the false negative rate which could be compared with the expected false negative rate.

Azer et al. [1] utilized a deep learning approach to decide if single-cell data indicates whether a tumor followed a linear or branched evolutionary process. Although their method is fast at prediction time and performs well on simulated data, it has not yet been proved whether the problem of identifying the minimum number of flips to obtain a linear perfect phylogeny is NP-hard. Additionally, the neural networks are trained on inputs of a fixed size and while padding could be used in predicting an input smaller than the fixed size [1], a new network would have to be trained if the input size is larger than the trained network. This drawback significantly reduces the speed advantage of such an approach as training of neural networks is both a time consuming and intensive process.

Here, we prove that the problem of determining the minimum number of flips from 0 to 1 in single-cell data in order for the data to represent a linear perfect phylogeny under the infinite sites model is NP-hard. Therefore, we develop a method called Phyolin that makes use of constraint programming to find the minimum number of flips required to represent a linear perfect phylogeny. The outputted number of flips from Phyolin is then used to compute the estimated false negative rate to assess the plausibility of a linear evolutionary pattern (Figure 1(a)). We evaluate the performance of Phyolin on both simulated and real datasets, demonstrating that our method is an accurate and reasonably fast method for classifying an evolutionary trajectory as linear or branched.

2 Problem Statement

Let n be the number of single cells sequenced and m be the number of unique mutations present in the n cells. When a single cell is sequenced, assuming no errors, the group of mutations present in that cell form a clone of the tumor. Under the infinite sites assumption (ISA), where each mutation i is gained exactly once and never subsequently lost, each sequenced cell corresponds to a leaf and we may infer a two-state perfect phylogeny using a polynomial time algorithm [12] where the binary character states encode the presence of mutation i in a cell j. We may equivalently represent a perfect phylogeny T as a binary matrix $B \in \{0,1\}^{n \times m}$ where $b_{ij} = 1$ if cell i harbors mutation j and 0 otherwise, for all $i \in [n]$ and $j \in [m]$. We provide the following definition [11] for convenience:

▶ **Definition 1.** *Given an n by m binary-character matrix B for n cells and m mutations, a* perfect phylogeny for *B is a rooted tree T with exactly n leaves provided that:*

1. *Each of the n cells labels exactly one leaf of T.*
2. *Each of the m mutations labels exactly one edge of T.*
3. *For any cell p, the mutations that label the edges along the unique path from the corresponding leaf to the root specify all of the mutations of p whose state is one.*

Next, we formalize the notation of a set of cells that contain a mutation as the *one-state*.

▶ **Definition 2.** *The* one-state *O_j of mutation j is the set of single cells i where $b_{ij} = 1$.*

A perfect phylogeny T either depicts linear evolution or branched evolution. Intuitively, a matrix B represents linear evolution if there exists a total order of the set of *one-states* with respect to the subset relation. Otherwise, we say perfect phylogeny T represents branched evolution. Also, we note that perfect phylogeny T is not necessarily bifurcating.

Utilizing the collection of *one-states* for all m mutations, we determine if a given binary matrix B represents a linear perfect phylogeny as follows (Figure 1(b)):

▶ **Definition 3.** *A binary matrix $B \in \{0,1\}^{n \times m}$ represents a* linear perfect phylogeny *if there exists a permutation $\pi : [m] \rightarrow [m]$ such that $O_{\pi(1)} \subseteq O_{\pi(2)} \subseteq \ldots \subseteq O_{\pi(|m|)}$.*

However, single-cell sequencing is not error free and matrix B can fail to represent a linear perfect phylogeny even when it is representative of the true evolutionary process. False negatives, where a mutation that is present is not indicated as such, are particularly problematic with rates of up to 0.4 [8]. False positives, where absent mutations are indicated as present, are less of an issue in practice with rates less than 0.0005 for typically used whole-genome amplification strategies [9]. Given that false negatives are particularly prevalent, we would like to know how many false negatives would have had to occur in order for the inferred perfect phylogeny under the ISA to have a linear structure? This leads to the following problem statement.

▶ **Problem 1** (Linear Perfect Phylogeny Flipping Problem (LPPFP)). *Given a matrix $B \in \{0{,}1\}^{n \times m}$, find the minimum number of bit flips from* 0 *to* 1 *such that B represents a linear perfect phylogeny.*

Under the null model, the true phylogeny is linear and thus the input data B must represent a linear perfect phylogeny. Therefore, any implied branching in matrix B is interpreted as a false negative and must be corrected through flipping to represent the presumed linear perfect phylogeny. It is important to note that under this null hypothesis a trivial solution always exists for any input. In the worst case, every 0 can be flipped to a 1. This results in a binary matrix with all values equal to 1, suggesting a linear perfect phylogeny with a single clone harboring all of the mutations. By seeking the solution that requires the fewest number of flips, we maximize the likelihood of the null model given the observed data, assuming an estimated false negative rate $\beta^* \leq 0.5$ of the sequencing technology. Upon obtaining the solution to the LPPFP, we reverse the problem and compute the implied false negative rate $\hat{\beta}$ that resulted from flipping in order to assess the plausibility of our null model. Then given the following null hypothesis, $H_0 : \hat{\beta} \leq \beta^*$, rejection of H_0 is equivalent to concluding that a linear perfect phylogeny is not plausible.

3 Complexity

Following [2], we will prove that LPPFP is NP-hard by a reduction from the chain graph insertion problem, a known NP-complete problem [24].

▶ **Theorem 4.** *LPPFP is NP-hard.*

Proof. We prove that LPPFP is NP-hard by considering a decision version k-LPPFP asking whether there exist k bit flips in input matrix B from 0 to 1 yielding a linear perfect phylogeny. We claim that k-LPPFP is NP-complete by reduction from the chain graph insertion problem.

We begin by stating the definition of a chain graph and introduce the chain graph insertion problem.

▶ **Definition 5.** *A bipartite graph $G = (X \cup Y, E)$ is a* chain graph *if there exists a permutation $\phi : \{1, \ldots, |Y|\} \to Y$ such that $\eta(\phi(1)) \subseteq \eta(\phi(2)) \subseteq \ldots \subseteq \eta(\phi(|Y|)$ where $\eta(v) = \{w \in X : (v{,}w) \in E\}$ is the set of adjacent nodes of v.*

▶ **Problem 2** (Chain Graph Insertion Problem (CG-IP) [24]). *Given a bipartite graph $G = (X \cup Y{,}E)$ and integer k, does there exist a chain graph $G' = (X \cup Y{,}E')$ such that $E \subseteq E'$ and $|E| + k = |E'|$?*

k-LPPFP $\in$ NP because given a certificate (set of k flips from 0 to 1) to k-LPPFP, we could order the columns by increasing cardinality of the resulting one-state sets and then check if that permutation satisfies the definition of a linear perfect phylogeny. We will now show that CG-IP $\leq_p$ k-LPPFP.

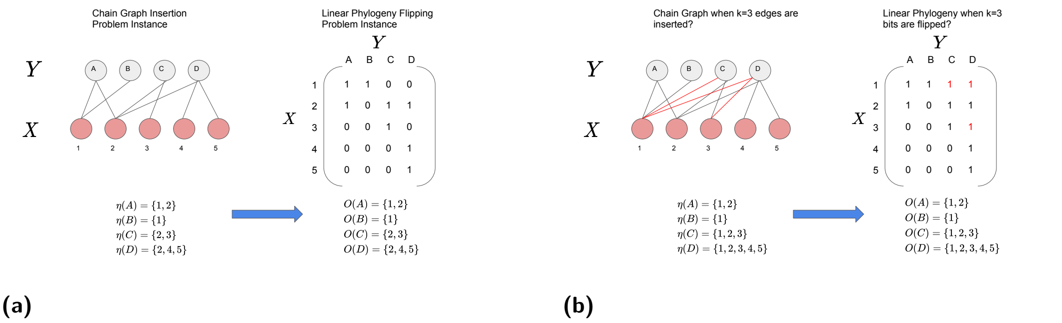


Figure 2 Chain graph insertion problem reduction (a) Polynomial time reduction of the CG-IP to LPPFP. (b) An equivalent solution of the CG-IP and LPPFP when $k = 3$.

Starting from an instance $(G = (X \cup Y,E),k)$ of CG-IP, we construct an instance B of k-LPPFP in the following manner. Binary matrix B has $|X|$ rows and $|Y|$ columns, and its entries are directly obtained from the edge set E of G: For each $v \in Y$, we set $b_{iv} = 1$ if i is a neighbor of v for all $i \in X$ and let $b_{iv} = 0$ otherwise. This can be done in polynomial time. Figure 2(a) demonstrates the polynomial time reduction. We claim that k edge insertions suffice to obtain a chain graph from G if and only if B represents a linear perfect phylogeny when k bits are flipped from 0 to 1.

$(\Longrightarrow)$ Suppose $(G,k) \in$ CG-IP. Then there exists an edge set $D \subseteq \{(u,v) : u \in X, v \in Y\} - E$ such that $|D| = k$ and $H = (X \cup Y, E \cup D)$ a chain graph. Then by definition of chain graph, there exists an permutation $\phi : \{1 \ldots |Y|\} \rightarrow Y$ such that $\eta(\phi(1)) \subseteq \eta(\phi(2)) \subseteq \ldots \subseteq \eta(\phi(|Y|)$. It is easy to see that by construction, $\eta(v) = O_v$, for all $v \in Y$. Since ϕ exists, a permutation π of the one-states also exists. Therefore, B represents a linear perfect phylogeny.

$(\Longleftarrow)$ Suppose $(B,k) \in k$-LPPFP. Then there exists a set F of positions (i,j) where $b_{ij} = 0$ to $b_{ij} = 1$ such that $|F| = k$. Let B^* be the resulting matrix after each flip at position $(i,j) \in F$ is made. Then B^* represents a linear perfect phylogeny and there exists a permutation $\pi : \{1 \ldots m\} \rightarrow [m]$ such that $O_{\pi(1)} \subseteq O_{\pi(2)} \subseteq \ldots \subseteq O_{\pi(|m|)}$. Using the equivalence between one-states and neighbors, $H = (X \cup Y,E)$ can be constructed from B^* in the following manner. First, create the set $X = [n]$ from the rows of B^* and the set $Y = [m]$ from the columns of B^*. Then, create the set E of edges as $\{(x,y) \in X \times Y \mid b^*_{xy} = 1\}$. By construction, $H = (X \cup Y, E)$ is a chain graph. ◀

4 Method

4.1 Model

To solve the LPPFP, we formulate a constraint optimization problem (COP) [21]. A COP is a constraint satisfaction problem (CSP) with an objective function that specifies which feasible solutions are preferred based on an optimization criteria. A CSP is defined by a tuple $(\mathcal{X},\mathcal{D},\mathcal{C})$, where $\mathcal{X} = \{x_1, \ldots ,x_n\}$ is the set of decision variables, $\mathcal{D} = \{d_1, \ldots d_n\}$ is the set of domains for $\mathcal{X}$ and $\mathcal{C}$ is a set of constraints that must be satisfied. A solution $a \in \mathcal{A}(\mathcal{X},\mathcal{D},\mathcal{C})$ to a CSP is an assignment of values $\{x_1 \mapsto v_1, \ldots , x_n \mapsto v_n\}$ such that $v_i \in d_i$ for all $i \in [n]$ and all constraints C are satisfied. To facilitate an objective function $f : \mathcal{A}(\mathcal{X},\mathcal{D},\mathcal{C}) \rightarrow \mathbb{R}$, an initial assignment $\hat{a} \in \mathcal{A}(\mathcal{X},\mathcal{D},\mathcal{C})$ is found. Then, a preference constraint is added to C, such that $f(a) \leq f(\hat{a})$ for a minimization problem or $f(a) \geq f(\hat{a})$ for a maximization problem. The search is continued and the preference constraint updated each time a feasible assignment $\hat{a}$ is found until no more feasible assignments exist. When this occurs, the assignment $\hat{a}$ is returned and $f(\hat{a})$ is the objective value. We note that in problems where multiple optimal solutions exist, it is possible to return all such valid solutions. But even though multiple solutions may exist, our focus is on assessing the plausibility of the null hypothesis. Therefore, it is sufficient to consider any optimal solution even if the respective assignments yield different linear perfect phylogenies.

First, we describe the set $\mathcal{X}$ of decision variables and the associated domains $\mathcal{D}$ used in the formulation. The set $\mathcal{X}$ contains the variables $\mathbf{x}$ and $\mathbf{c}$. Intuitively, the values taken by $\mathbf{x}$ represent a binary matrix B' used to represent a linear perfect phylogeny after flipping. More formally, given a set n of cells and a set m of mutations, let $x_{ij} = 1$ if cell i has mutation j in the linear perfect phylogeny B' after flipping and 0 otherwise for each cell $i \in [n]$, and mutation $j \in [m]$. Then, $\mathcal{D}(x_{ij}) = \{0,1\}$, for all $i \in [n], j \in [m]$. Note that a decision variable is defined for every entry in B, even though only flips from 0 to 1 are allowed. This is to facilitate future handling of false positives. The variables $\mathbf{c}$, are used to define a permutation of the columns in B', such that after flipping is completed, B' will adhere to the definition of a linear perfect phylogeny. Recall that in order to represent a linear perfect phylogeny, there must exist permutation $\pi : [m] \to [m]$ such that $O_{\pi(1)} \subseteq O_{\pi(2)} \subseteq \ldots \subseteq O_{\pi(m)}$. Let $c_j = \pi(j)$ for all $j \in [m]$. Then $\mathcal{D}(c_j) = [m]$ for all $j \in [m]$.

Since, our goal is to find the linear perfect phylogeny that requires as few flips as possible, that is minimizing the number of false negatives we infer, we define an objective function that minimizes the number of flips from 0 to 1,

$$\min \sum_{i \in [n]} \sum_{j \in [m]} (x_{ij} - b_{ij}). \tag{1}$$

The set $\mathcal{C}$ ensures that the outputted binary matrix B' meets the conditions of representing a linear perfect phylogeny and also that only flips from 0 to 1 can be made. The set $\mathcal{C}$ consists of three constraints,

$$\textsc{alldifferent}(c), \tag{2}$$

$$(b_{ij} = 1) \Rightarrow (x_{ij} = 1) \qquad \forall i \in [n], \forall j \in [m], \tag{3}$$

$$(c_k < c_j) \Rightarrow (x_{ij} \leq x_{ik}) \qquad \forall k,j \in [m], \forall i \in [n]. \tag{4}$$

Equation (2) is a global constraint that ensures that every mutation is assigned a unique ordering in the permutation. Equation (3) ensures that all entries in B, where $b_{ij} = 1$ remain 1 in the linear perfect phylogeny B'. That is, they cannot be flipped from 1 to 0. Finally, Equation (4) ensures the defining property of a linear perfect phylogeny is met by ensuring that if $\pi(k)$ is less than $\pi(j)$ then it must hold that $O_k \subseteq O_j$ for all $k,j \in [m]$.

We implemented Phyolin in C++ utilizing IBM ILOG CP OPTIMIZER[2]. Phyolin is publicly available at `https://github.com/elkebir-group/phyolin`.

4.2 Null Hypothesis

The formulation of the method assumes a null hypothesis that the phylogeny is linear. The output of Phyolin is an estimate of the false negative rate $\hat{\beta}$ under this hypothesis such that

$$\hat{\beta} = \frac{\sum_{i \in [n]} \sum_{j \in [m]} \mathbb{1}(b'_{ij} = 1, b_{ij} = 0)}{\sum_{i \in [n]} \sum_{j \in [m]} b'_{ij}}, \tag{5}$$

[2] `https://www.ibm.com/analytics/cplex-cp-optimizer`

where $B' = [b'_{ij}]$ is the value of the decision variables $\mathbf{x}$ obtained from the solution of Phyolin and $B = [b_{ij}]$ is the input matrix.

Given some threshold β^*, which could be based on knowledge of the system estimated false negative rate or alternatively conservatively set at 0.4 [8], then we can reject the null hypothesis that the phylogeny is linear whenever $\hat{\beta} > \beta^*$.

5 Results

In order to evaluate Phyolin, we perform *in silico* experiments as well as run Phyolin on real data. First, we seek to evaluate the performance of Phyolin when the simulated data closely approximates a recently published high throughput single-cell DNA sequencing study of an acute myeloid leukemia (AML) cohort [17] (Section 5.1). Section 5.2 describes the application of Phyolin to patients with childhood acute lymphoblastic leukemia [10]. All experiments were conducted on a server with Intel Xeon Gold 5120 dual CPUs with 14 cores each at 2.20GHz and 512 GB RAM.

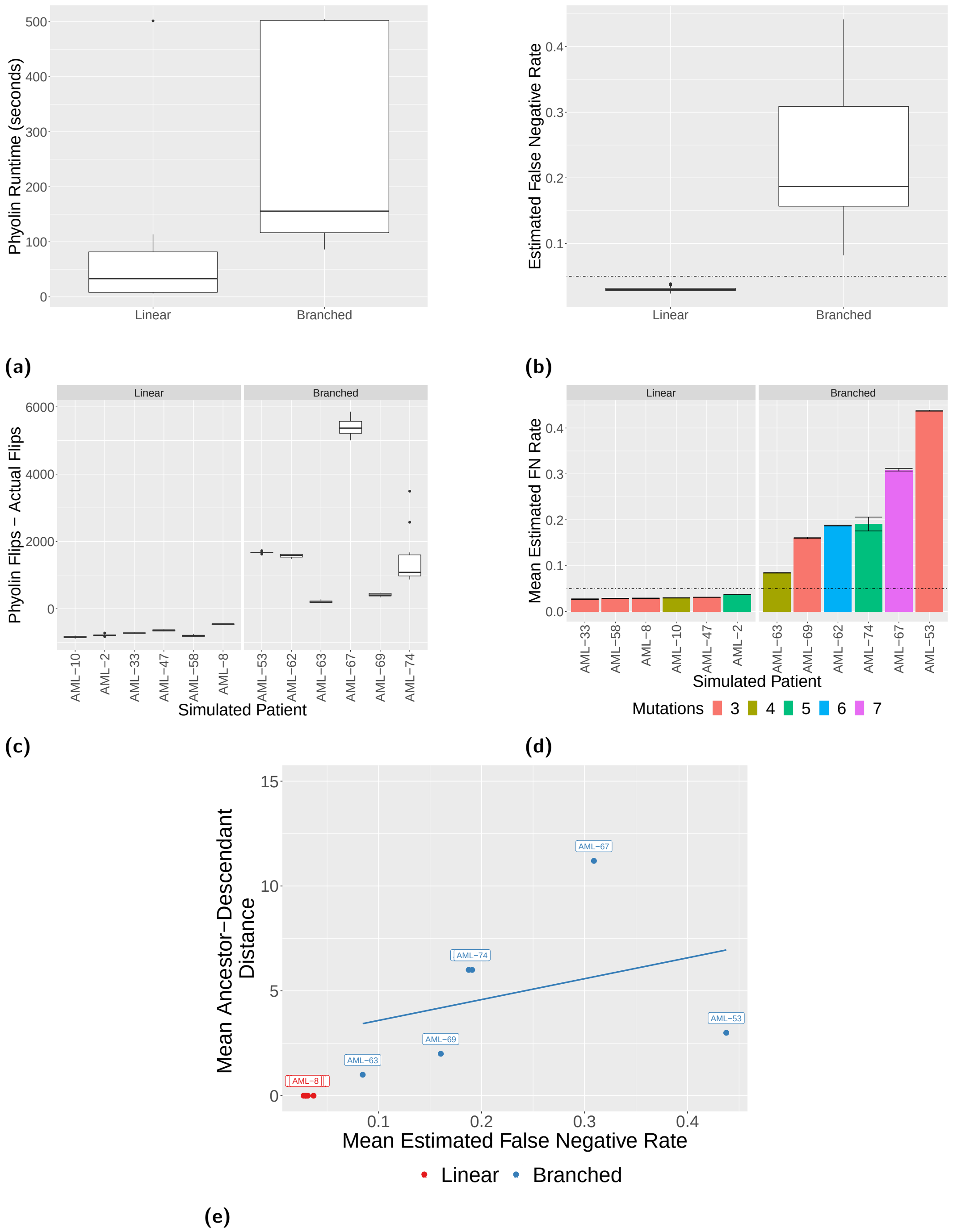
5.1 Simulations Approximating an Acute Myeloid Leukemia Cohort

In a recent study, Morita et al. [17] performed high-throughput targeted droplet microfluidic DNA single-cell sequencing on a cohort of 77 patients with acute myeloid leukemia (AML) and inferred the evolutionary tree of each patient using SCITE [13]. Utilizing high-throughput sequencing resulted in a median of 7,584 cells sequenced per patient [17]. AML is a cancer type where both linear and branching evolutionary patterns are suspected [17]. As a result, the set of trees published [17] were a mix of linear and branched patterns.

Unfortunately, the single-cell data is not yet publicly available but the estimated clonal prevalence of each clone in the inferred publish tree was included in the supplementary material along with the estimated per patient false negative rate for the sequencing technology. The clonal prevalence of clone i is the number of cells in the sample mapped to clone i divided by the total number of cells in the sample. We utilize this published data to evaluate Phyolin on a scenario that closely aligns to a highly realistic scenario. Therefore, we utilize the published clonal prevalence rates for a subset of 12 patients in the cohort in order to simulate the total number of single-cells sequenced at false negative rate $\beta = 0.05$. This value of β is to approximate the average system false negative rate for the sequencing technology used in [17]. The subset contains six patients with linear trees and mutations ranging from 3-5 and six patients with branched trees and mutations ranging from 3-7. A total of 10 replications were performed per simulated patient. Table 1 shows a summary of the patients selected for inclusion in the simulation study.

We set an upper limit on runtime of Phyolin at 500 seconds with 80% of the replications returning an optimal solution in under the time limit. We chose the 500 second time limit to facilitate timely analysis of the input data. Figure 3(a) shows the distribution of runtime by the simulated evolutionary pattern. Linear patterns resulted in a median runtime of 33 seconds (IQR: 8-82 seconds) and branched patterns resulted in a median of 156 seconds (IQR: 117-502 seconds). The median input size (cells × mutations) of the linear patterns was 24,435 and 28,775 for branched patterns. The largest input was for AML-74 with 9,279 cells by 5 mutations and no replications completed within the time limit. Only 1 replication with a linear pattern did not complete within the time limit. This implies that optimal solutions are found much faster when the true pattern is linear.

Figure 3(b) compares the distribution of the estimated false negative rate $\hat{\beta}$ for the patients with linear versus branched published trees over all 10 replications. The simulated system false negative rate was 5% and is shown as a dashed line where relevant. From



(a)

(b)

(c)

(d)

(e)

Figure 3 Phyolin results of the *in silico* experiments on a simulated cohort of patients with AML (a) A comparison of Phyolin runtimes in seconds between instances with different evolutionary patterns. (b) A comparison of the distribution of $\hat{\beta}$ between simulated linear and branched topologies over 10 replications. (c) The distribution of the difference between the number of flips performed by Phyolin and the simulated flips per patient. (d) The mean value of $\hat{\beta}$ for each patient along with the standard error. (e) Relationship between estimated false negative rate and the ancestor-descendant distance. Each point represents the mean value over 10 replications and is labeled by the numerical patient identifier. A linear trend line is shown with a 95% confidence interval.

Table 1 Simulation study based on characteristics of a published AML cohort [17] Shown is the patient identifier, the published evolutionary pattern of the tree, the number of mutations, the total cells sequenced [17], the median number of false negatives over 10 replications, Phyolin estimated number of false negatives over 10 replications, the median $\hat{\beta}$ over 10 replications, and the median probability of a linear perfect phylogeny as determined by the comparison deep learning method [1].

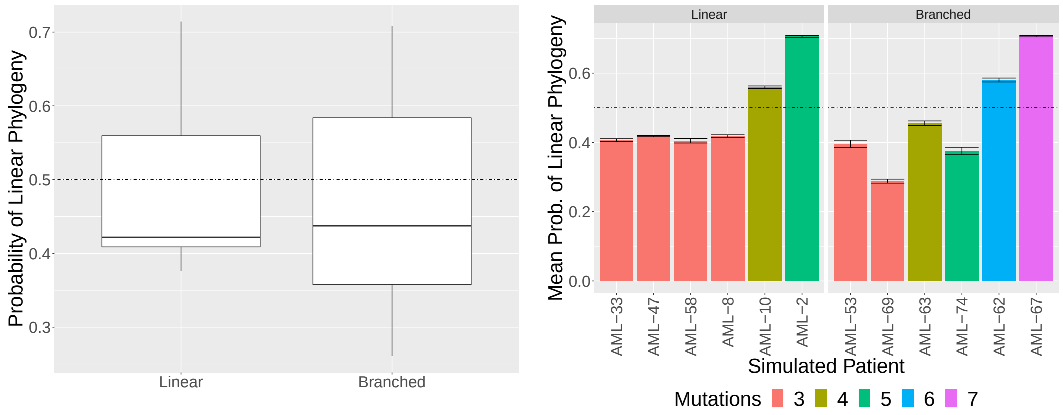
patient	pattern	m	n [17]	median flips	Phyolin median flips	median $\hat{\beta}$	med prob.
AML-2	Linear	5	7931	1826	1039	0.037	0.70
AML-8	Linear	3	4675	759	294	0.029	0.42
AML-10	Linear	4	8729	1427	584	0.037	0.56
AML-33	Linear	3	8120	1091	350	0.027	0.41
AML-47	Linear	3	6491	1135	488	0.032	0.42
AML-58	Linear	3	8170	1280	472	0.029	0.40
AML-53	Branched	3	8013	544	2220	0.44	0.39
AML-62	Branched	6	4027	726.5	2299	0.19	0.58
AML-63	Branched	4	8347	1238.5	1432	0.084	0.44
AML-67	Branched	7	6024	1061.5	6440	0.31	0.71
AML-69	Branched	3	7462	651.5	2122	0.16	0.29
AML-74	Branched	5	9279	1020	294	0.17	0.38

Figure 3(b), we note a significant difference in the distributions between linear and branched instances. Additionally, the median of the linear perfect phylogeny patients is 0.03 (IQR: 0.028-0.032) and every linear replication is less than $\beta^* = 0.05$ while the median of the branched perfect phylogeny patients is 0.19 (IQR: 0.16-0.31) and every branched replication is greater than $\beta^* = 0.05$.

Figure 3(c) compares the difference between the number of flips performed by Phyolin and the actual number of simulated false negatives. Interestingly, Phyolin performs fewer flips than simulated false negatives for all linear patients and performs a much higher number of flips than simulated for branched patterns. This underestimation can be attributed to the fact that not every false negative implies that branching occurs and so Phyolin only needs to flip those that do. Thus, the difference between the number of Phyolin flips and the number of simulated flips may be negative. The fact that Phyolin performs more flips than was actually simulated in the branched cases aligns with the original intuition for its design as we not only need to correct false negatives but also true negatives in order to force the pattern to be linear.

Figure 3(d) shows the mean $\hat{\beta}$ and standard error for each simulated patient over the 10 replications. The number of mutations are also shown in order to investigate if increasing number of mutations increases $\hat{\beta}$. Although there appears to be some effect when increasing the number of mutations, it does not strictly hold. However, utilizing a strict threshold of $\beta^* = 0.05$ results in perfect classification of the topology for all patients and all replications.

Since the number of mutations does not necessarily impact the estimated false negative rate $\hat{\beta}$, another consideration is whether or not $\hat{\beta}$ increases with the amount of branching. To this end, we compare the ancestor-descendant distance between the simulated true tree B^* and the inferred linear tree B'. A mutation x is an *ancestor* of mutation y if x occurs on the path from the root to y, in which case y is said to be a descendant of x. Ancestor-descendant (AD) distance is defined as the size of the symmetric difference between the sets of ordered pairs of characters, or ancestor-descendant pairs, introduced on distinct edges of perfect phylogenies B^* and B'. A higher AD distance implies a greater degree of branching in



(a) **(b)**

Figure 4 Results of the Deep Learning approach [1] applied to the *in silico* experiments on a simulated cohort of patients with AML (a) A comparison of the distribution of the probability of a linear perfect phylogeny between simulated linear and branched topologies over 10 replications. (d) The mean value of the probability of a linear perfect phylogeny for each patient along with the standard error. A horizontal line indicates the threshold probability (0.05) used to classify an input as linear.

the true tree. For example AML-63 has only one branch and AML-67 has three distinct branching events. Figure 3(e) shows the relationship between the mean estimated false negative rate $\hat{\beta}$ and the mean AD distance per simulated patient over 10 replications. The AD distance is 0 for all patients with a simulated linear perfect phylogeny. This means that Phyolin correctly infers the true tree when it is linear. Also, there is evidence of a correlation between the estimated false negative rate $\hat{\beta}$ and the AD distance.

Azer et al.'s [1] deep learning method for classifying topology is the most similar method for comparison with Phyolin. Therefore, we retrained this deep neural network to support our input size of 9300 cells and 7 mutations. We used a default hidden layer size of 100 and drop-out rate of 0.9, 5000 training examples and 500 epochs. We did not modify any other hyperparameters. The input size was selected so that only one network needed to be trained for all *in silico* experiments and we used padding for any instances where the $n < 9300$ or $m < 7$. After 200 epochs the best validation accuracy was 64.1% and completed in 2168 seconds (36.1 minutes). After 500 epochs, the best validation accuracy was 64.8% and completed in 3997 seconds (66.6 minutes). This suggests that further learning was unlikely. We report the probability that the phylogeny is linear on the same simulation instances when evaluated with the trained model.

Table 1 shows the median probability that the phylogeny is linear over the 10 replications per simulated patient. We use a cutoff of 0.5 as the threshold for classifying a topology as linear. Figure 4(a) shows the distribution of the probabilities over all patient replications by ground truth topology. Classification accuracy was 100% for 6 of the 12 simulated patients (Linear: AML-2, AML-10, Branched: AML-53, AML-63, AML-69 and AML-74) and 0% for the remaining 6 simulated patients. Figure 4(b) shows mean estimated probability per patient and standard error for the 10 replications. The predicted probability tends to increase as the number of mutations increases.

In summary, the simulated AML cohort results show that, in contrast to the Deep Learning approach [1], Phyolin correctly and quickly classifies large instances as linear with a strict threshold β^* set at the system estimated false negative rate. Furthermore, as the

Table 2 Summary of Phyolin analysis of two patients with ALL Shown is the patient identifier, the number of cells sequenced [10], the number of mutations, the number of flips performed by Phyolin, the estimated false negative rate $\hat{\beta}$ and the false negative rate threshold β^* that was estimated for the sequencing technology [14].

patient	cells sequenced [10]	mutations	Phyolin flips	$\hat{\beta}$	β^* [10]
Patient 2	115	16	403	0.36	0.18
Patient 6	146	10	191	0.15	0.18

amount of branching increases, the estimated $\hat{\beta}$ tends to increase. Thus the greater the difference between $\hat{\beta}$ and β^*, the more confident we can be in rejecting the null hypothesis that the phylogeny is linear.

5.2 Real Data of Childhood Acute Lympoblastic Leukemia Patients

Gawad et al. [10] performed single-cell DNA sequencing on a cohort of six patients with childhood acute lymphoblastic leukemia (ALL). As a subtype of leukemia, ALL is also postulated to follow both linear and branched trajectories [22]. We evaluate Phyolin on two of the six patients in this cohort: Patient 2 and Patient 6. The input size for Patient 2 was 115 cells by 16 mutations. Two independent, previous analyses of the sequencing data of Patient 2 suggested a branched topology [15, 23].

In addition, we consider Patient 6 because this patient was analyzed by the deep learning method [1]. In another line of work, Kuipers et al. [14] investigated the validity of the ISA in single-cell data within the ALL dataset [10]. Using their method, they compared the likelihood of the data under both a finite sites and infinite sites model via a Bayes Factor and determined with high probability that Patient 6 suffered a loss of the *SUSD2* mutation. They used SCITE [13] to infer trees from the single-cell data under both an infinite sites and finite sites model. These trees are show in Figure 5. The tree inferred under the ISA is linear (Figure 5(a)) while the tree inferred under the finite sites model is branched due to the loss of *SUSD2* (Figure 5(b)). The input size for Patient 6 was 146 cells by 10 mutations. Table 2 summarizes results obtained by Phyolin.

For Patient 2, Phyolin estimated a false negative rate of 0.36, which is much greater than the rate of 0.18 estimated in [10]. Taking $\beta^* = 0.18$ implies rejecting the null hypothesis of a linear perfect phylogeny. This concurs with the branched trees published in [15, 23]. Phyolin utilized the 500 second time limit to complete its run for Patient 2 despite the small input size.

For Patient 6, Phyolin estimated a false negative rate of 0.15. The published false negative rate β^* in [10] was 0.18. This implies that we cannot reject the null hypothesis of a perfect linear perfect phylogeny. Indeed, the linear tree output by Phyolin does concur with Figure 5(a). The comparison deep learning approach [1] also concluded that the phylogeny was linear with probability 0.79. It is important to note that allowing for mutation loss of *SUSD2*, in line with [14], will lead to a smaller false negative rate $\hat{\beta}$. This suggests that incorporating mutation loss into Phyolin is an important area for future study.

6 Discussion

In this work, we introduced the Linear Perfect Phylogeny Flipping Problem and showed that it is NP-hard. To solve the LPPFP, we developed a method named Phyolin that takes as input a binary matrix of single-cell DNA sequencing data and then identifies a linear

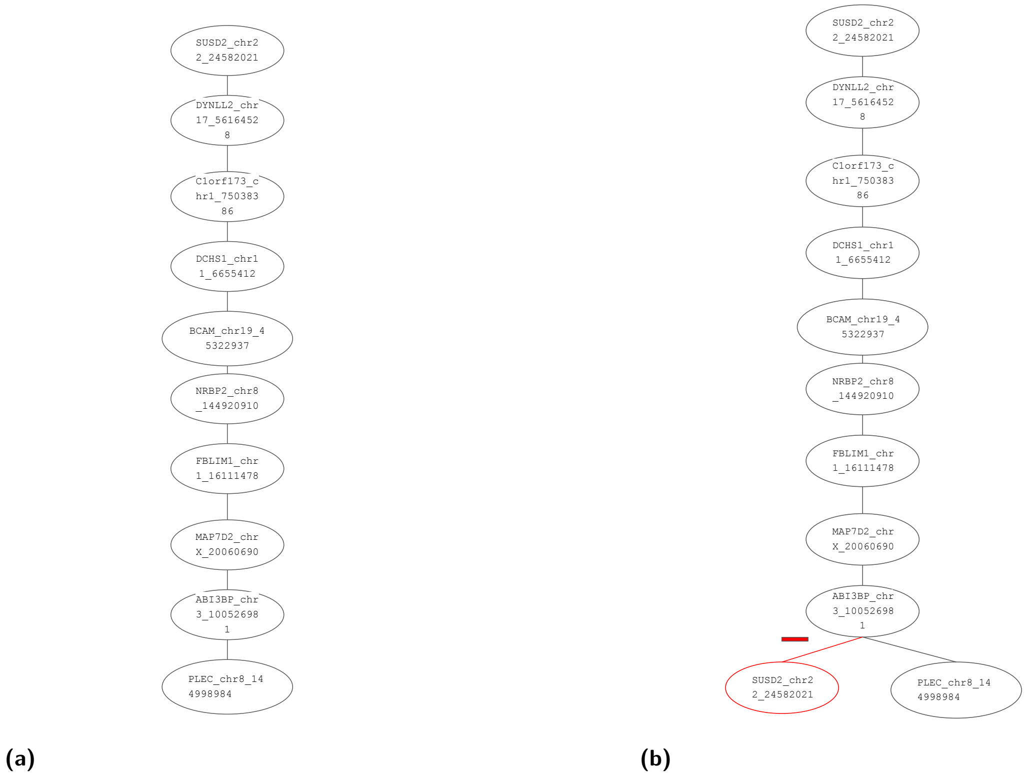


Figure 5 Patient 6 candidate trees Two possible trees published in [14]. (a) Tree adheres to the ISA, (b) Tree with a branched topology indicating a mutation loss of *SUSD2*.

perfect phylogeny in the data by assuming that any implied branching are actually false negatives. It returns an estimate of the false negative rate under the null hypothesis that the perfect phylogeny is linear and allows the user to compare this estimate to a false negative threshold at which the null hypothesis of a linear perfect phylogeny can subsequently be accepted or rejected. We tested Phyolin on both simulated data and real data and showed that it is more accurate than a recent deep learning method [1]. In conclusion, Phyolin is a reliable, easy to use and fast method to assess the likelihood of a linear evolution before more complex reconstruction methods are utilized.

There are several future research directions. First, Phyolin lacks an absolute criterion or threshold for rejecting the null hypothesis that the phylogeny is linear. To address this, we plan to modify Phyolin so that instead of trying to solve the problem to optimality, a user could input a false negative rate at which he or she would fail to reject the null hypothesis of a linear perfect phylogeny if any solution exists below the supplied threshold. This would allow Phyolin to explicitly solve a constraint satisfaction problem and likely reduce the runtime.

Second, Phyolin can be modified to allow false positives, which means allowing flips from 1 to 0. However, before that modification is made, more robust *in silico* experiments should be conducted with simulated false positives and doublets. Although false positives are rare, it is possible that a single or a few critically positioned false positives requires excessive inference of false negatives in order to represent a linear perfect phylogeny. High doublet rates could potentially result in low estimates of false negative when the true phylogeny is branched. Therefore, a constraint could also be incorporated to ignore up to the inputted number of doublets.

Third, Phyolin could easily incorporate mutation and cell clustering through additional constraints when supplied with a number of cell clusters and/or mutation clusters. A search could be performed to find the optimal number of clusters such that the likelihood of the data of is maximized. Fourth, even when the phylogeny is branched, the trunk of the tree may be linear or there might be a long branch with linear evolution within that branch. This means that a subset of the mutations form a linear perfect phylogeny. A future direction is to explore if Phyolin can identify a subset of mutations that are likely to be truncal or form a long branch of the tree, thus potentially providing fast partial inference of the tree.

Finally, given the results of ALL Patient 6, exploring evolutionary models that allow ISA violations, such as mutation loss, is an exciting direction for future study. In particular, modeling Patient 6 as a 1-Dollo phylogeny, where each mutation is gained only once and subsequently lost at most once, could potentially be achieved by replicating each column once and then using Phyolin. If the two columns representing the same mutation are distinct in the inferred linear perfect phylogeny, then that implies that the mutation was lost once. The plausibility of a linear perfect phylogeny under both ISA and under a 1-Dollo model could be compared [5].

References

1 Erfan Sadeqi Azer, Mohammad Haghir Ebrahimabadi, Salem Malikić, Roni Khardon, and S Cenk Sahinalp. Tumor Phylogeny Topology Inference via Deep Learning. *bioRxiv*, 2020. doi:10.1101/2020.02.07.938852.

2 Duhong Chen, Oliver Eulenstein, David Fernandez-Baca, and Michael Sanderson. Minimum-flip supertrees: complexity and algorithms. *IEEE/ACM transactions on computational biology and bioinformatics*, 3(2):165–173, 2006.

3 Alexander Davis, Ruli Gao, and Nicholas Navin. Tumor evolution: Linear, branching, neutral or punctuated? *Biochimica et Biophysica Acta (BBA)-Reviews on Cancer*, 1867(2):151–161, 2017.

4 Amit G Deshwar, Shankar Vembu, Christina K Yung, Gun Ho Jang, Lincoln Stein, and Quaid Morris. PhyloWGS: reconstructing subclonal composition and evolution from whole-genome sequencing of tumors. *Genome biology*, 16(1):35, 2015.

5 Mohammed El-Kebir. SPhyR: tumor phylogeny estimation from single-cell sequencing data under loss and error. *Bioinformatics*, 34(17):i671–i679, 2018.

6 Mohammed El-Kebir, Layla Oesper, Hannah Acheson-Field, and Benjamin J Raphael. Reconstruction of clonal trees and tumor composition from multi-sample sequencing data. *Bioinformatics*, 31(12):i62–i70, 2015.

7 Mohammed El-Kebir, Gryte Satas, Layla Oesper, and Benjamin J Raphael. Inferring the mutational history of a tumor using multi-state perfect phylogeny mixtures. *Cell systems*, 3(1):43–53, 2016.

8 Yusi Fu, Chunmei Li, Sijia Lu, Wenxiong Zhou, Fuchou Tang, X Sunney Xie, and Yanyi Huang. Uniform and accurate single-cell sequencing based on emulsion whole-genome amplification. *Proceedings of the National Academy of Sciences*, 112(38):11923–11928, 2015.

9 Yusi Fu, Chunmei Li, Sijia Lu, Wenxiong Zhou, Fuchou Tang, X Sunney Xie, and Yanyi Huang. Uniform and accurate single-cell sequencing based on emulsion whole-genome amplification. *Proceedings of the National Academy of Sciences of the United States of America*, 112(38):11923–11928, September 2015.

10 Charles Gawad, Winston Koh, and Stephen R Quake. Dissecting the clonal origins of childhood acute lymphoblastic leukemia by single-cell genomics. *Proceedings of the National Academy of Sciences*, 111(50):17947–17952, 2014.

11 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/CBO9780511574931.

12 Dan Gusfield. *ReCombinatorics: the algorithmics of ancestral recombination graphs and explicit phylogenetic networks*. MIT press, 2014.

13 Katharina Jahn, Jack Kuipers, and Niko Beerenwinkel. Tree inference for single-cell data. *Genome biology*, 17(1):86, 2016.

14 Jack Kuipers, Katharina Jahn, Benjamin J Raphael, and Niko Beerenwinkel. Single-cell sequencing data reveal widespread recurrence and loss of mutational hits in the life histories of tumors. *Genome research*, 27(11):1885–1894, 2017.

15 Salem Malikic, Katharina Jahn, Jack Kuipers, S Cenk Sahinalp, and Niko Beerenwinkel. Integrative inference of subclonal tumour evolution from single-cell and bulk sequencing data. *Nature communications*, 10(1):1–12, 2019.

16 Salem Malikic, Farid Rashidi Mehrabadi, Simone Ciccolella, Md Khaledur Rahman, Camir Ricketts, Ehsan Haghshenas, Daniel Seidman, Faraz Hach, Iman Hajirasouliha, and S Cenk Sahinalp. PhISCS: a combinatorial approach for subperfect tumor phylogeny reconstruction via integrative use of single-cell and bulk sequencing data. *Genome research*, 29(11):1860–1877, 2019.

17 Kiyomi Morita, Feng Wang, Katharina Jahn, Jack Kuipers, Yuanqing Yan, Jairo Matthews, Latasha Little, Curtis Gumbs, Shujuan Chen, Jianhua Zhang, et al. Clonal evolution of acute myeloid leukemia revealed by high-throughput single-cell genomics. *bioRxiv*, 2020.

18 Peter C Nowell. The clonal evolution of tumor cell populations. *Science*, 194(4260):23–28, 1976.

19 Yuanyuan Qi, Dikshant Pradhan, and Mohammed El-Kebir. Implications of non-uniqueness in phylogenetic deconvolution of bulk DNA samples of tumors. *Algorithms for Molecular Biology*, 14(1):19, 2019.

20 Edith M Ross and Florian Markowetz. OncoNEM: inferring tumor evolution from single-cell sequencing data. *Genome biology*, 17(1):1–14, 2016.

21 Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach Third Edition*. Pearson, 2010.

22 Anna Schuh, Jennifer Becq, Sean Humphray, Adrian Alexa, Adam Burns, Ruth Clifford, Stephan M Feller, Russell Grocock, Shirley Henderson, Irina Khrebtukova, et al. Monitoring chronic lymphocytic leukemia progression by whole genome sequencing reveals heterogeneous clonal evolution patterns. *Blood, The Journal of the American Society of Hematology*, 120(20):4191–4196, 2012.

23 Leah Weber, Nuraini Aguse, Nicholas Chia, and Mohammed El-Kebir. PhyDOSE: Design of follow-up single-cell sequencing experiments of tumors. *BioRxiv*, 2020.

24 Mihalis Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981.

25 Hamim Zafar, Nicholas Navin, Ken Chen, and Luay Nakhleh. SiCloneFit: Bayesian inference of population structure, genotype, and phylogeny of tumor clones from single-cell genome sequencing data. *Genome research*, 29(11):1847–1859, 2019.

The Longest Run Subsequence Problem

Sven Schrinner
Algorithmic Bioinformatics, Heinrich Heine University Düsseldorf, Germany
sven.schrinner@hhu.de

Manish Goel
Max Planck Institute for Plant Breeding Research, Cologne, Germany

Michael Wulfert
Algorithmic Bioinformatics, Heinrich Heine University Düsseldorf, Germany

Philipp Spohr
Algorithmic Bioinformatics, Heinrich Heine University Düsseldorf, Germany

Korbinian Schneeberger
Max Planck Institute for Plant Breeding Research, Cologne, Germany
Cluster of Excellence on Plant Sciences (CEPLAS), Heinrich Heine University Düsseldorf, Germany
Faculty of Biology, LMU Munich, Planegg-Martinsried, Germany

Gunnar W. Klau[1]
Algorithmic Bioinformatics, Heinrich Heine University Düsseldorf, Germany
Cluster of Excellence on Plant Sciences (CEPLAS), Heinrich Heine University Düsseldorf, Germany
gunnar.klau@hhu.de

Abstract

Genome assembly is one of the most important problems in computational genomics. Here, we suggest addressing the scaffolding phase, in which contigs need to be linked and ordered to obtain larger pseudo-chromosomes, by means of a second incomplete assembly of a related species. The idea is to use alignments of binned regions in one contig to find the most homologous contig in the other assembly. We show that ordering the contigs of the other assembly can be expressed by a new string problem, the longest run subsequence problem (LRS). We show that LRS is NP-hard and present reduction rules and two algorithmic approaches that, together, are able to solve large instances of LRS to provable optimality. In particular, they can solve realistic instances resulting from partial *Arabidopsis thaliana* assemblies in short computation time. Our source code and all data used in the experiments are freely available.

2012 ACM Subject Classification Theory of computation → Dynamic programming; Mathematics of computing → Combinatorial optimization; Applied computing → Computational genomics

Keywords and phrases alignments, assembly, string algorithm, longest subsequence

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.6

Supplementary Material Source code and all data used in the experiments available at `https://github.com/AlBi-HHU/longest-run-subsequence`.

Funding Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – projectID 395192176 – as well as under Germany's Excellence Strategy – EXC 2048/1 – projectID 390686111.

1 Introduction

Genome assembly from sequencing reads enables the analysis of an organism at its genome level and is one of the most important problems in computational genomics. The first step is usually to assemble the reads based on overlap or k-mer based approaches to create contigs,

[1] Corresponding author

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti; Article No. 6; pp. 6:1–6:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

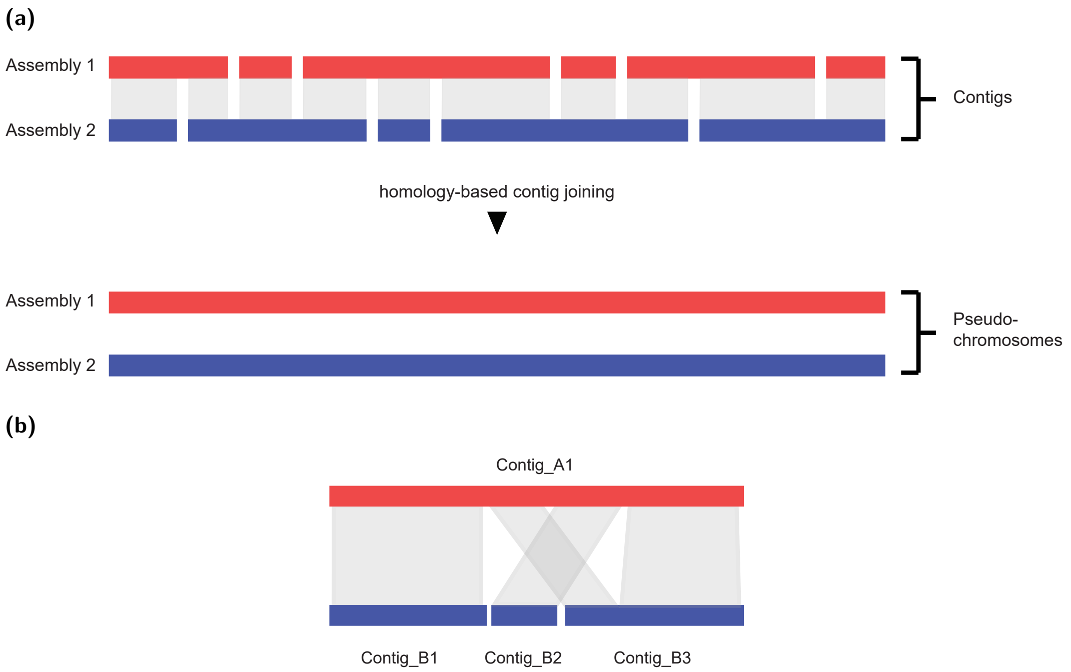


Figure 1 Homology-based scaffolding. (a) Independent initial assemblies (contigs), which are joined into pseudo-chromosomes by using homologies between contigs for scaffolding. (b) Alignments between contigs from different samples. A1 determines the order of B1, B2 and B3.

which then need to be put into correct order and orientation in a scaffolding phase to generate the final assembly of pseudo-chromosomes. Presence of a high-quality chromosome-level reference genome of the same species can significantly simplify assembly generation as it can be used as a template to order these contigs [1, 3]. However, for many species, such a reference genome is not available.

There are two commonly used approaches for scaffolding. First, different types of maps provide anchors for the contigs in the genome. These could be, for example, genetic maps, physical maps or cytological maps providing markers with known positions in the genome and known distances between each other [9]. The other approach is to use long-range genomic information to link multiple contigs and put them into correct order and orientation. Prominent examples are linked barcoded reads like 10X sequencing [10], Hi-C data based on chromatin conformation capture [2] and optical mapping [7].

Yet another way for contig scaffolding is to use two or more incomplete assemblies from closely related samples [4]. Regions of unconnected contigs for one sample might be connected with the help of another, related sample, e.g., a genome assembly of an individual of the same species, providing an overall gain in information for both samples. Local similarities between contigs from different samples can be used to align and order them. Ideally, this leads to long chromosome like sequences called pseudo-chromosomes, where the contigs of different samples are aligned like shingles next to each other, as illustrated in Figure 1(a).

Note that structural rearrangements (such as translocations or inversions) and repeat regions between genomes can result in non-sequential and non-unique mappings within contigs and can thus lead to misleading connections between contigs. These events need to be considered when finding homologous contigs as shown in Figure 1(b).

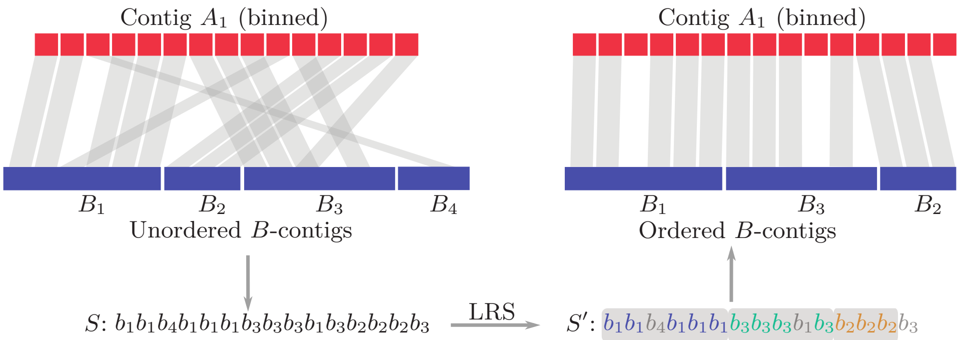


Figure 2 Processing of a single contig A_1. The bins are matched against all contigs of another sample B. Solving Longest Run Subsequence (LRS) on the corresponding string S, yields a maximal subsequence with at most one run for every contig. This induces the optimal order for a subset of B-contigs.

In the simplest setting of two incomplete assemblies we are given two sets of contigs $A = \{A_1, \ldots, A_l\}$ and $B = \{B_1, \ldots, B_m\}$ computed from two different samples. As already stated, the contigs are not ordered with respect to genome positions, and it is this order we rather want to compute. More precisely, we aim at inferring the most likely order from between-sample overlaps among the contigs.

Assuming we want to order the contigs in B, we divide every contig A_i of A into smaller, equally sized chunks, called *bins*, map them against the contigs in B and determine the best matching contig for every bin. If A_i actually overlaps with multiple contigs in B, we should be able to partition A_i into smaller parts based on mapping the bins to different contigs in B. However, sequencing or mapping errors as well as mutations between the samples can cause some bins to map onto a "wrong" contig, i.e., a contig belonging to a different area than the bin. Therefore a method to find the best partition of A_i needs to distinguish between actual transitions from one B-contig to another and noise introduced by errors or mutations.

Figure 2 illustrates the different steps in solving this problem. Starting from a binned contig from A, here A_1 for illustration, and its mapping preferences to the unordered contigs in B, we reformulate this ordering problem as a string problem. In essence, we want to find the longest subsequence of the input string of mapping preferences that consists only of consecutive runs of contigs from B where each such run may occur at most once. This subsequence corresponds to an ordering of the contigs in B, which can be transferred to the original problem.

In this paper we formalize this process and introduce the *Longest Run Subsequence* problem (LRS). We show that LRS is NP-hard. Nevertheless, we want to solve large instances of LRS to provable optimality in reasonable running time and therefore present a number of reduction rules and two algorithms based on integer linear programming and dynamic programming, respectively. We evaluate both approaches on synthetic instances and find that they show complementary strengths regarding the number of runs and the alphabet size. We also test our approaches on realistic instances, which occurred during assembly of *Arabidopsis thaliana* samples and could not be solved by a brute-force method. We show that all those instances can be solved within short computation time. Our code and all data used in the experiments are freely available at `https://github.com/AlBi-HHU/longest-run-subsequence`.

2 Problem Formulation

A string $S = s_1, \dots, s_m$ is a sequence over characters from a finite alphabet Σ. A subsequence of S is a sequence $s_{i_1}, \dots, s_{i_k}$, such that $1 \leq i_1 < i_2 < \dots < i_k \leq m$. We denote the *substring* $s_i, \dots, s_j$ of S as $S[i,j]$ and k consecutive occurrences of a character σ inside a string S as σ^k and call it a *run*. Let $\sigma(r)$ be the character of the run r and $L(r)$ its length. By summarizing the characters of S into maximally long runs, S can be represented as a unique sequence of runs $r_1, \dots, r_n = \sigma(r_1)^{L(r_1)} \dots \sigma(r_n)^{L(r_n)}$. For every $\sigma \in \Sigma$ we define $P_\sigma(i)$ as the index of the last run before r_i containing σ in S (0 if it does not occur). As an example, the string from Figure 2 can be compressed to ${b_1}^2{b_4}^3{b_1}^3{b_3}^3{b_1}^1{b_3}^1{b_2}^3{b_3}^1$ with $P_{b_1}(4) = 3$, $P_{b_1}(3) = 1$ and $P_{b_4}(1) = 0$.

We propose to model the optimal partition of a single contig as a string optimization problem. Formally, we use the contigs from set B as the alphabet, that is $\Sigma = \{b_1, \dots, b_l\}$ and write the contig A_i as a string $S = b_{i_1} \dots b_{i_m}$ over Σ by replacing the bins of A_i with the corresponding character of the best match from B. On the one hand, we want every single bin to be assigned to its preferred contig in B, but we also want a simple partition, which is not skewed by wrong mappings of single bins. We therefore restrict valid partitions of A_i to contain at most one continuous part for every contig in B. This prevents large parts to be interrupted by single mismatching bins, at the cost of not being able to capture short-ranged translocations as seen in Figure 1(b). A partition can be represented as a subsequence S' of the string S, which only contains at most one run for every $\sigma \in \Sigma$. The runs in S' are the parts corresponding to one B-contig each, while the dropped characters from S are bins in conflict with S'. Finding the best partition can thus be stated as the following optimization problem:

▶ **Problem 1** (Longest Run Subsequence, LRS)**.** Given an alphabet $\Sigma = \{\sigma_1, \dots, \sigma_{|\Sigma|}\}$ and a string $s = s_1, \dots, s_m$ with $s_i \in \Sigma$, find a longest subsequence $S' = s'_1, \dots, s'_k$ of S, such that S' contains at most one run for every $\sigma \in \Sigma$. That is, for every pair of positions i and j with $i < j$, it holds that

$$s'_i = s'_j \Rightarrow s'_l = s'_i \quad \text{for all } i < l < j \ .$$

We denote the length of an optimal LRS solution for S with $\mathrm{LRS}\,(S)$. Since we want to maximize the length of the run subsequence, it is always beneficial to either completely add or completely remove a run of S. Once a character $s_i \in \Sigma$ from a run s_i^k is added to s', there can never be any other occurrence of s_i outside this run. Thus, the entire run must be added to s' to achieve maximum length. We will therefore mainly refer to runs instead of single characters.

3 Complexity

In this section we prove hardness of the Longest Run Subsequence problem. More precisely, we show that dLRS, the decision version of the problem is NP-complete. An instance of dLRS is given by a tuple (S, k) and consists in answering the question whether S has a longest run subsequence of length at least k.

▶ **Theorem 1.** *dLRS is NP-complete.*

Proof. It is easy to see that dLRS is in NP, because it can be checked in polynomial time whether a string s' is a soluton, that is, s' is a run subsequence and $|s'| \geq k$.

To prove NP-hardness, we reduce from the Linear Ordering Problem (LOP), which has been shown to be NP-hard [5]. LOP takes a complete directed graph with edge weights and no self-loops as input and looks for an ordering among the vertices, such that the total weights of edges following this order (i.e., edges leading from lower ordered vertices to higher ordered vertices) is maximized.

We show that dLOP, the decision problem of LOP, that is, the question whether a vertex ordering exists whose weight is at least a given threshold, can be polynomially reduced to dLRS. Let $G = (V, E)$ be a complete digraph with $|V| = n$. We denote the weight of $(v_i, v_j) \in E$ with w_{ij} and the sum of all weights of G as w_{sum}. Without loss of generality we can assume that all edge weights are positive: The number of edges following a linear order is fixed, so adding a sufficiently large offset to all weights only adds a fixed value to any solution without changing the core problem. This allows us to characterize LOP as finding an acyclic subgraph G' with maximum weight, because the non-negativity of the weights always forces either (v_i, v_j) or (v_j, v_i) to be in G' for every pair of vertices $v_i, v_j \in V$.

The proof consists of two parts. First, we show how to transform G into a string S. Second, we show that G has a LOP solution of weight k if and only if S has a LRS of size

$$f_G(k) := (n-1) \cdot M + \frac{n(n-1)(n-2)}{3} \cdot M' + n(n-1) \cdot w_{\mathrm{sum}} + 2k \tag{1}$$

with $M' := 4n^2 \cdot w_{\mathrm{sum}}$ and $M := M' \cdot n^3$.

For the transformation, we define Σ using three different types of characters:

1. Separators $\$_i$ for every vertex $v_i \in V$.
2. Edge signs $E_{\{i,j\}}$ for every pair $v_i, v_j \in V$. Note that $E_{\{i,j\}} = E_{\{j,i\}}$.
3. Triangle signs $\Delta_{(i,j,k)}$ for every triangle in G. Note that triangles between three vertices have an orientation and can be rotated. Therefore $\Delta_{(i,j,k)} = \Delta_{(j,k,i)} = \Delta_{(k,i,j)} \neq \Delta_{(i,k,j)} = \Delta_{(k,j,i)} = \Delta_{(j,i,k)}$.

On the highest level the string S is constructed as shown in Equation 2. It consists of one large block per vertex, each of them separated by a run of the associated separation sign of length M.

$$S = \underbrace{\overbrace{[\mathrm{EB}]_{1,2}}^{\substack{\text{edge block}\\ \text{for } (v_1,v_2)}} \; [\mathrm{EB}]_{1,3} \dots [\mathrm{EB}]_{1,n}}_{\text{vertex block for } v_1} \$_1^M [\mathrm{EB}]_{2,1} \dots [\mathrm{EB}]_{2,n} \$_2^M \dots \$_{n-1}^M [\mathrm{EB}]_{n,1} \dots [\mathrm{EB}]_{n,n-1} \tag{2}$$

Each vertex block consists of a series of *edge blocks* (EB), which we define as follows:

$$[\mathrm{EB}]_{i,j} = E_{\{i,j\}}^{w_{ij}+w_{\mathrm{sum}}} \quad \Delta_{(i,j,1)}^{M'} \dots \Delta_{(i,j,n)}^{M'} \quad E_{\{i,j\}}^{w_{ij}+w_{\mathrm{sum}}} \tag{3}$$

In the same way as the i-th vertex block is associated with vertex v_i, the edge substrings in it are associated with the outgoing edges of v_i. Note that there is one EB missing in every vertex block, as self-loops are not allowed. Finally, $[\mathrm{EB}]_{i,j}$ contains all triangle signs for triangles, in which (v_i, v_j) occurs, i.e., $\{\Delta_{(i,j,k)} \mid 1 \leq k \leq n, k \neq i, k \neq j\}$, which, for the sake of notation, is written as $\Delta_{(i,j,1)}^{M'} \dots \Delta_{(i,j,n)}^{M'}$ in Equation 3. The triangle signs are padded by edge signs for (v_i, v_j). Every edge sign $E_{\{i,j\}}$ occurs only in the two edge blocks $[\mathrm{EB}]_{i,j}$ and $[\mathrm{EB}]_{j,i}$. The length of the edge sign runs depends on the weight of the corresponding edge (in either direction), rewarding the higher weighted edge. We also add w_{sum} to the length every edge sign run $E_{\{i,j\}}$.

As for the numbers M and M', the latter is chosen to be larger than the combined length of all edge sign runs. This makes a single triangle sign run more profitable than any selection of edge sign runs. In the same manner, M is chosen to be larger than all triangle sign runs combined.

Using this construction, a valid solution $G' = (V, E')$ for a dLOP instance (G, k), i.e., an acyclic subgraph of G with total weight at least k, can be transformed into a valid solution for a dLRS instance $(S, f_G(k))$. First, all separation runs are selected, yielding a total length of $(n-1) \cdot M$. Second, for every edge in E', all edge signs in the corresponding edge blocks are selected. Since $|E'| = \frac{n(n-1)}{2}$, this adds at least $2 \cdot \left(\frac{n(n-1)}{2} \cdot w_{\text{sum}} + k\right)$ characters to the solution. Finally, G' is acyclic, so for every triangle in G, there is at least one edge missing in G'. Thus, by construction of S, one run can be selected for every triangle sign without interfering with the edge sign runs, adding the missing $\frac{n(n-1)(n-2)}{3} \cdot M'$ characters.

Given a solution S' for the dLRS instance $(S, f_G(k))$, we show how to obtain a subgraph G' of total weight at least k for the original dLOP instance. The subsequence S' must contain all separation runs and a run for every triangle sign, because without all separation and triangle signs selected at some place, it is (by choice of M and M') impossible to reach length $f_G(k)$ for any k. Every selected edge sign run is therefore restricted to a single edge block. The idea is that the choice of selecting $E_{\{i,j\}}$ either in $[\text{EB}]_{i,j}$ or $[\text{EB}]_{j,i}$ corresponds to the choice of having either (i, j) or (j, i) in the DAG G' for the original LOP. Since we added w_{sum} to the length of every edge sign run and there are only $\frac{n(n-1)}{2}$ edge signs in total, S' must contain both runs inside an edge block, in order to reach length $n(n-1) \cdot w_{\text{sum}}$ (the third summand in $f_G(k)$). Thus, either edge signs or triangle signs may be selected inside an edge block, but not both. G' is finally obtained by selecting an edge e if and only if the edge sign runs in the corresponding edge block are selected. This yields $\frac{n(n-1)}{2}$ edges with a total weight of at least k. For every vertex pair v_i, v_j, exactly one of the edges (v_i, v_j) and (v_j, v_i) is selected, because their corresponding edge blocks share the same edge sign.

It remains to be shown that the obtained subgraph G' is acyclic. We can directly conclude that G' contains no triangles, since every triangle sign $\Delta_{(i,j,k)}$ has to be taken, prohibiting either (i, j), (j, k) or (k, i) (or two of them) to be part of G'. Assume that G' contains a cycle $v_{i_1}, v_{i_2}, v_{i_3}, \ldots, v_{i_l}, v_{i_1}$ of length $l \geq 4$. Then, either (v_{i_1}, v_{i_3}) or (v_{i_3}, v_{i_1}) must be in G'. The latter would lead to a triangle, which we could already exclude from G'. But $(v_{i_1}, v_{i_3}) \in G'$ implies that a circle of length $l - 1$ also exists in G'. Repeated use of this argument implies that G' also has a cycle with length 3, which is a contradiction to triangles being excluded. Thus, G' cannot contain a cycle of length 4 or greater and must be acyclic.

In summary, the decision problem whether there is a solution for a dLOP instance (G, k) can be reduced to the decision problem whether a solution for the dLRS instance $(S, f_G(k))$ obtained from G exists. ◀

4 Solution Strategies

To solve instances of LRS in practice we propose three reduction rules and two algorithmic approaches. As of Theorem 1 we cannot guarantee a polynomial running time.

4.1 Reduction Rules

In Section 2 we already pointed out that an optimal solution for LRS always selects complete runs of characters and we reduced the notation of the input to runs of characters with a certain length each. This can also be seen as a reduction rule to the original problem formulation as the remaining size of the solution space now depends on the number of runs n instead of the actual string length m. We refer to this as the *run rule*. Two more reduction rules rely on the following lemma:

▶ **Lemma 2.** *Let S, T be two strings over the disjoint alphabets Σ_S and Σ_T. Then the optimal LRS solutions for S and T can be concatenated to form an optimal solution for the concatenated string ST.*

Proof. Since the two alphabets are disjoint, an LRS solution for S does not contain any characters from Σ_T. Therefore the choice of the subsequence for S does not influence the valid subsequences for T and vice versa. This means that optimal solutions for S and T can be computed independently and concatenated fo form a valid solution for ST. Obviously, an optimal solution for ST cannot be longer than the combined length of optimal solutions for the S and T, otherwise the latter would not be optimal. ◀

According to Lemma 2 we can divide an LRS instance S into smaller independent instances, if we find a prefix $r_1, \ldots, r_l$ of S, which uses an exclusive sub-alphabet Σ_p, i.e., $r_1, \ldots, r_l \in \Sigma_p^*$ and $r_{l+1}, \ldots, r_n \in (\Sigma \setminus \Sigma_p)^*$. This *prefix rule* can be applied in linear time by starting with the prefix r_1 and extending it until we either reach the end of S, in which case no independent suffix exists, or until the prefix is closed regarding the used characters. Let l be the index of the last occurrence of $\sigma(r_1)$. Since $\sigma(r_1)$ is used in the prefix, all runs $r_2, \ldots, r_l$ must belong to the prefix. Now start with $k = 2$ and update l to the index of the last occurrence of $\sigma(r_k)$ (if this index is higher than l), increase k by 1 and repeat until $k > l$. If $l < n$, an independent prefix is found, otherwise such a prefix does not exist.

This idea can be extended to the *infix rule*, which finds independent infixes via the following lemma.

▶ **Lemma 3.** *Let S, T be two strings over the disjoint alphabets Σ_S and Σ_T and let l be an arbitrary position in S. Then it holds that*

$$LRS(s_1 \ldots s_l T s_{l+1} \ldots s_m) = LRS\left(s_1 \ldots s_l \$^{LRS(T)} s_{l+1} \ldots s\right)$$

with $\$ \notin \Sigma_S \cup \Sigma_T$.

Proof. For the same reason as in Lemma 2 LRS for T can be solved independently from S. For the combined string $s_1 \ldots s_l T s_{l+1} \ldots s_m$ the infix T is either entirely dropped in the optimal subsequence or the optimal solution of T itself is entirely taken as a part of the combined solution. Thus, T contributes either 0 or LRS (T) characters to the optimal combined solution. Therefore, if the solution for T is already known, $s_1 \ldots s_l T s_{l+1} \ldots s_m$ can be solved by replacing T with a run of length LRS (T) of a new character \$. ◀

Following Lemma 3 we can search for an independent infix in S to obtain two smaller instances. Instead of starting with r_1, we start with an arbitrary character $\sigma \in \Sigma$ as anchor and use the infix $r_k, \ldots, r_l$ as a start with r_k and r_l being the first and last occurrence of σ, respectively. Similarly to the prefix search, we iterate over all runs in the infix and move the markers k, l to the left and right, whenever we encounter a new character with occurrences outside $r_k, \ldots, r_l$, until the infix is closed (with respect to used characters) or the entire string is contained. This is repeated with every character in Σ as anchor, possibly yielding multiple infixes. Adjacent independent infixes are merged into larger ones, since we want as many runs as possible to be replaced with a single run, as shown in Lemma 3. Infixes, which consist of only one run, are discarded, because they do not pose an actual reduction. Finding and merging all infixes can be done in time $\mathcal{O}(n \cdot |\Sigma|)$.

For a maximum reduction, the rules are applied as follows: First, the prefix rule is iteratively applied on S until no further independent prefix can be found. Second, the infix rule is applied on every sub-instance found so far. For every infix found the procedure is repeated by starting with the prefix rule again.

4.2 Solving with Integer Linear Programming

We present two algorithms to solve LRS to optimality, which have complementary strengths and weaknesses. The first is based on an Integer Linear Program (ILP). This approach scales well with large alphabets, but struggles with a large number of runs. We also propose a dynamic programming (DP) approach, which remains fast for long strings, but suffers from large alphabets. Both algorithms work exclusively on the runs of an input string S.

ILPs are a commonly used technique to model and solve combinatorial optimization problems. We model the LRS formulation from before as an ILP in the following way: Let n be the number of runs in S and let $x_1, \ldots, x_n$ be binary variables with $x_i = 1$ if r_i is in the optimal subsequence and $x_i = 0$ otherwise. Any possible subsequence of runs can therefore be represented by a variable assignment. Since we want to maximize the length of the subsequence, we define our objective function as the weighted sum over all taken runs, using their lengths as weights:

$$\max \sum_{i=1}^{n} x_i L(r_i) \tag{4}$$

Let r_i, r_j be two runs with $i < j$ and $\sigma(r_i) = \sigma(r_j)$. If both runs are selected, no other intermediate run with a different character can be selected according to the LRS conditions. This is ensured by adding the following constraints for any pair of run r_i, r_j, with the same character.

$$\text{subject to} \quad (j-i)x_i + \left(\sum_{\substack{i<l<j \\ \sigma(r_l) \neq \sigma(r_i)}} x_l \right) + (j-i)x_j \leq 2(j-i) \quad \text{for all } 1 \leq i < j \tag{5}$$

If either r_i or r_j are not taken, the constraint does not prevent any other combination of runs between them. The total number of constraints is bounded by $\lceil \frac{n}{2} \rceil^2$ and the number of non-zero entries in the constraint matrix is bounded by $n \cdot \lceil \frac{n}{2} \rceil^2$. The ILP has been implemented using the Python interface of PuLP, which solves the ILP with the free solver CoinOR[2].

4.3 Solving with Dynamic Programming

As an alternative to the ILP formulation the problem can also be solved bottom-up by a DP. Let $D[i, F]$ be the length of an optimal LRS solution for $r_1 \ldots r_i$, which includes r_i itself and one run for every $\sigma \in F \subseteq \Sigma$ only. The DP can be initialized with $D[0, \emptyset] = 0$ and $D[0, F] = -\infty$ for $F \neq \emptyset$. Known solutions can be extended by adding the runs of S iteratively, always selecting an optimal predecessor for each run and keeping track of already used characters with the second parameter F.

[2] `https://github.com/coin-or/pulp`

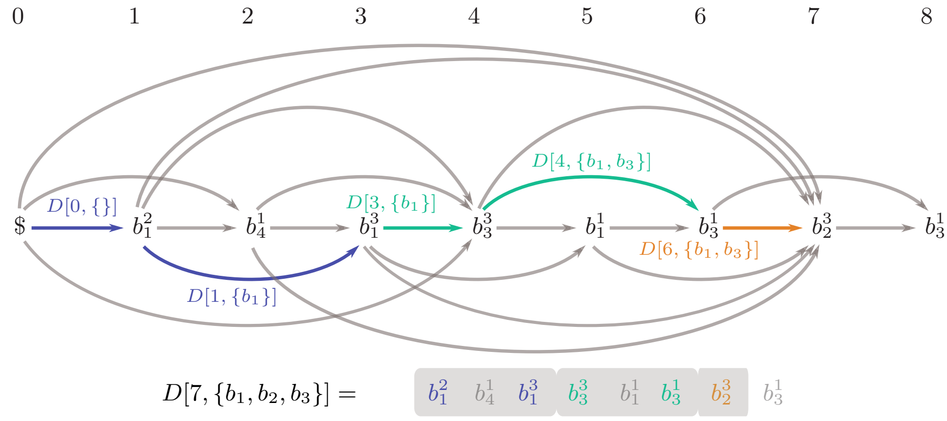


Figure 3 Graph visualizing the recursion for the running example. Arcs represent the possible predecessors for every run. Colors mark the optimal path and the DP entries taken by the recursion.

The full DP is as follows:

$$D[0, F] = 0 \qquad \forall F \subseteq \Sigma$$

$$D[0, F] = -\infty \qquad \forall F \neq \emptyset$$

$$D[i, F] = \begin{cases} \max\limits_{\sigma \in \Sigma \cup \{\$\}} \left\{ \begin{array}{ll} D[P_\sigma(i), F] + L(r_i) & \text{if } \sigma = \sigma(r_i) \\ D[P_\sigma(i), F \setminus \{\sigma(r_i)\}] + L(r_i), & \text{if } \sigma \neq \sigma(r_i) \end{array} \right\} & \text{if } \sigma(r_i) \in F \\ -\infty & \text{otherwise} \end{cases} \tag{6}$$

The recursion can be visualized by a directed acyclic graph as shown in Figure 3. It contains a start vertex corresponding to the empty prefix of S and one vertex for every run in S. Each vertex has an incoming edge from the start vertex and from position $P_\sigma(i)$ for every $\sigma \in \Sigma$. Every path in the graph corresponds to a (possibly invalid for LRS) subsequence of S. It is optimal to only consider the last occurrences of any character before r_i as predecessors for r_i itself, because if we take a LRS subsequence s', where r_i is proceeded by the second to last occurrence of some other character σ before r_i, we can make s' longer by adding the last occurrence of σ as well.

To compute $D[i, F]$ we first check whether the character of the i-th run is in F. If not, then there is no subsequence using only characters from F, but containing r_i, so the optimal value for this case is $-\infty$. Otherwise, $D[i, F]$ is computed by taking the maximum over all previous occurrences for every $\sigma \in \Sigma \cup \{\$\}$: For σ we take the longest solution from position $P_\sigma(i)$ and the set $F \setminus \{\sigma(r_i)\}$ as used characters, because such a solution can be extended to a solution for $D[i, F]$ by appending r_i. Note that \$ is a character not in Σ and that $P_\$(i) = 0$. The only exception is made for $\sigma = \sigma(r_i)$, where we know that $D[P_\sigma(i), F]$ corresponds to a subsequence ending with $\sigma(r_i)$, such that we can append r_i without any new characters being used.

The optimal solution for LRS can be found by taking the entry of D with the highest score and using the backtracking information from the DP to obtain the corresponding subsequence. The DP table has a total of $n + 1$ columns and $2^{|\Sigma|}$ rows with each entry taking $\mathcal{O}(|\Sigma|)$ time to compute. This leads to a worst-case runtime of $\mathcal{O}\left(|\Sigma| \cdot n \cdot 2^{|\Sigma|}\right)$ for the DP, making this a *fixed parameter tractable* (FPT) approach for LRS with the alphabet size as parameter.

5 Experiments

We performed computational experiments on two different types of instances. First, we generated random instances to see how the two algorithms scale on string length and alphabet size. Second, we ran both algorithms on instances, which we came across a different project [4]. Originally these instances were solved by a brute-force approach, but this failed for longer strings, which motivated us to further investigate this problem.

All tests were run on an AMD Ryzen 7 2700X with 32GB of memory on Ubuntu 20.04. The algorithms are implemented in Python and executed via Snakemake [8] using Python 3.7.6 and PuLP version 1.6.8.

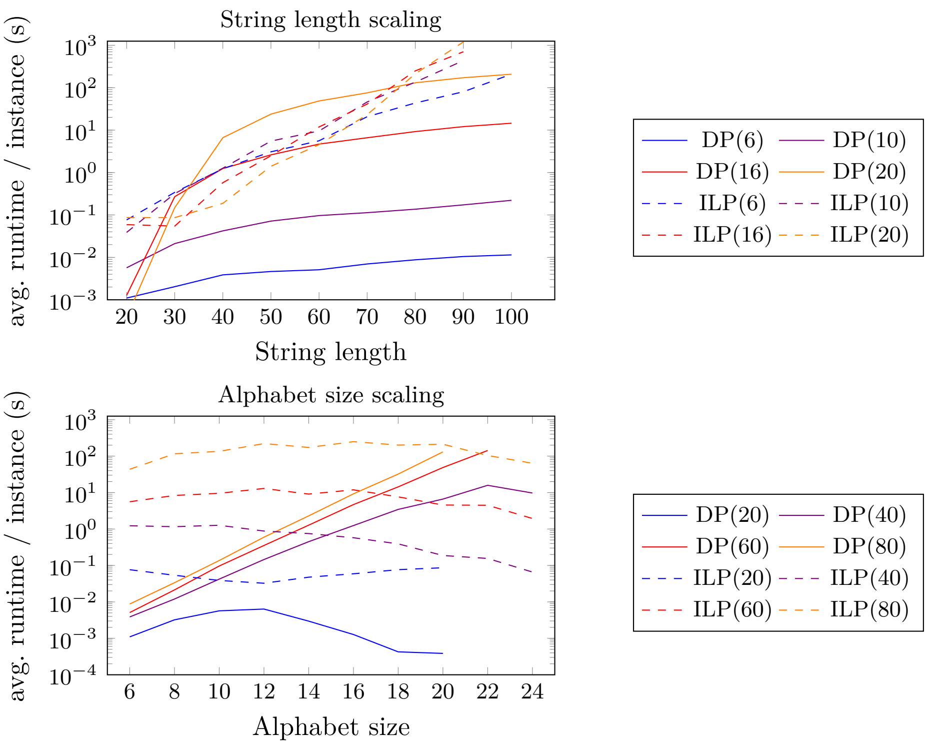


Figure 4 Run time plotted against string length (top) and alphabet size (bottom). Each curve represents an algorithm and an additional parameter (number in parentheses), which is alphabet size in the top blot and the string length in the bottom plot.

5.1 Synthetic Data

The synthetic data was created by randomly generating strings with different lengths and alphabet sizes. For any combination a total of 20 strings was generated, such that every string is guaranteed to use the entire alphabet assigned to it. These instances pose worst-case instances for our algorithms, as the proposed reduction rules can hardly be applied. The runs are quite short in general and since there is no structurally induced locality among the characters, instances could be split very rarely. All runs instances were solved with all reductions rules applied.

Figure 4 shows how the runtime scales with both increasing string lengths and increasing alphabet size. For a fixed alphabet size the runtime scales about exponentially with the string length for the ILP as shown in the top plot. In fact, the alphabet size only has very

Table 1 Comparison of runtime (in seconds) between DP and ILP on instances from real data. The times are for all 15 instances.

Algorithm	All rules	Runs and prefix	Only runs
DP	0.01	0.01	out of memory
ILP	2.15	2.03	0.09

minor effect on the ILP compared to the string length, which becomes visible in the bottom plot, with a slight favor of larger alphabets. The DP behaves complementary to the ILP, scaling exponentially in the alphabet size and sub-exponentially with string length. Longer strings take longer to process according to the top plot as opposed to the ILP being almost oblivious to the alphabet size. However, if the alphabet becomes very large in relation to the string length, the running time suddenly decreases for the DP.

The scaling can be explained by the properties of the algorithms. The ILP has a binary decision variable for every run, increasing the number of possible (but not necessarily feasible) variable assignments exponentially with the number of runs. Since ILP solvers usually fall back to branching in case cutting planes do not suffice, the bad scaling with running times appears logical. Larger alphabets might lead to a lower number of constraints (and thus a lower runtime), as the ILP contains one constraint for every pair of runs with the same character. As already pointed out in Section 4.3 the DP table grows linearly with the number of runs and exponentially with alphabet size, explaining the high running time on large alphabets. But also the memory consumption grew drastically. While the DP only needed 227MB of RAM for a string of length of 60 and $|\Sigma| = 16$, the consumption went up to 2,8GB for $|\Sigma| = 20$ and to 9,2GB for $|\Sigma| = 22$. With $|\Sigma| = 24$ the machine ran out of memory, making the memory consumption a larger bottleneck than the computation time. With the ILP no such issues could be observed. The decreasing running time for very large alphabets is caused by the reduction rules, as it leads to a higher number of characters occurring only in a single run and thus to a higher chance of the string being splittable into independent parts.

5.2 Real-World Data

We consider a dataset which was generated to test the performance of an approach to find structural rearrangements [4]. It consists of fragmented assemblies that have been generated by introducing random breaks in chromosome-level assemblies of the Col-0 and L*er* accessions of *Arabidopsis thaliana* [6, 4]. We tested 15 instances, which remained unsolved by brute-force methods, containing string lengths between 34 and 50 and alphabets of size 31 to 38.

Using all three reduction rules, both algorithms were able to solve all instances in very short time with the DP outperforming the ILP quite significantly, see Table 1. However, not using the prefix and infix rules caused the DP to run out of memory because of the large alphabets, while the ILP got faster. Since the reduction rules themselves only have minimal impact on the runtime, this must be due to overhead in the ILP solver when solving many small instances instead of one large one.

6 Discussion

The experiments showed that optimal LRS solutions can be found in short time for instance sizes that occur on assemblies of real samples. We presented two different algorithms whose running times depend on two important instance properties, namely string length and alphabet size. Random strings, however, do not seem to be a good indicator for actual assembly instances, which are already pre-sorted except for some noise or rearrangements. The reduction rules have little to no impact on random strings, while they reduce the assembly instances to almost trivial sub-instances. This implies that reduction rules might be more important in practice than the algorithm to process the remaining preprocessed instance.

One potential problem of the model itself was mentioned in Section 2. LRS only allows for one run per character, which automatically induces an ordering on the underlying contigs. This can be problematic if the binned contig contains a translocation that splits a long run into two, e.g., $b_1b_1b_1b_2b_2b_2b_1b_1b_1$. The LRS model will drop one of the b_1 runs, even though it would be better to leave the order of B_1 and B_2 open due to lack of evidence.

Another limitation arises while mapping the bins. Since only the best match for every bin is taken, any mapping ambiguity is ignored, which might drop valuable information. There is also no support for inversions inside the model. While inverted alignments can be taken into account for the mapping step of a single bin, the model stays unaware of inversions and the fact that an interval of bins is actually in the reverse order compared to the second assembly. However, this might not be as problematic as it sounds, because the bins are not mapped to other bins but to entire contigs. As long as inversions are contained in a single contig, they should have no impact on the ordering that the model produces.

7 Conclusion

Ordering contigs by means of an incomplete assembly of a related species occurs as a variant of homology-assisted assembly, which does not require chromosome-level assemblies already. We introduced the Longest Run Subsequence (LRS) problem, formalizing the contig ordering problem as a string problem. We proved that LRS is NP-complete and presented reduction rules and two algorithms, which work well for long instances and large alphabets, respectively, which we showed on a synthetic data set. Regarding real data, we managed to solve all instances that could not be solved by a brute force approach in short computation time.

From the theoretical side, we find it interesting to further investigate approximability and fixed-parameter tractability of LRS. From a practical perspective, we plan to further test the approach on real assembly data, also taking more than two related assemblies into account.

References

1 Michael Alonge, Sebastian Soyk, Srividya Ramakrishnan, Xingang Wang, Sara Goodwin, Fritz J. Sedlazeck, Zachary B. Lippman, and Michael C. Schatz. RaGOO: fast and accurate reference-guided scaffolding of draft genomes. *Genome Biology*, 20(1):224, October 2019. doi:10.1186/s13059-019-1829-6.

2 Joshua N. Burton, Andrew Adey, Rupali P. Patwardhan, Ruolan Qiu, Jacob O. Kitzman, and Jay Shendure. Chromosome-scale scaffolding of de novo genome assemblies based on chromatin interactions. *Nature Biotechnology*, 31(12):1119–1125, December 2013. doi:10.1038/nbt.2727.

3 Lauren Coombe, Vladimir Nikolić, Justin Chu, Inanc Birol, and René L Warren. ntJoin: Fast and lightweight assembly-guided scaffolding using minimizer graphs. *Bioinformatics*, April 2020. btaa253. doi:10.1093/bioinformatics/btaa253.

4 Manish Goel, Hequan Sun, Wen-Biao Jiao, and Korbinian Schneeberger. SyRI: finding genomic rearrangements and local sequence differences from whole-genome assemblies. *Genome Biology*, 20(1):277, December 2019. doi:10.1186/s13059-019-1911-0.

5 Martin Grötschel, Michael Jünger, and Gerhard Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32:1195–1220, December 1984. doi:10.1287/opre.32.6.1195.

6 The Arabidopsis Genome Initiative. Analysis of the genome sequence of the flowering plant Arabidopsis thaliana. *Nature*, 408(6814):796–815, 2000.

7 Wen-Biao Jiao, Gonzalo Garcia Accinelli, Benjamin Hartwig, Christiane Kiefer, David Baker, Edouard Severing, Eva-Maria Willing, Mathieu Piednoel, Stefan Woetzel, Eva Madrid-Herrero, Bruno Huettel, Ulrike Hümann, Richard Reinhard, Marcus A. Koch, Daniel Swan, Bernardo Clavijo, George Coupland, and Korbinian Schneeberger. Improving and correcting the contiguity of long-read genome assemblies of three plant species using optical mapping and chromosome conformation capture data. *Genome Research*, 27(5):778–786, May 2017. doi:10.1101/gr.213652.116.

8 Johannes Köster and Sven Rahmann. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, August 2012. doi:10.1093/bioinformatics/bts480.

9 Haibao Tang, Xingtan Zhang, Chenyong Miao, Jisen Zhang, Ray Ming, James C. Schnable, Patrick S. Schnable, Eric Lyons, and Jianguo Lu. ALLMAPS: robust scaffold ordering based on multiple maps. *Genome Biology*, 16(1):3, January 2015. doi:10.1186/s13059-014-0573-1.

10 Neil I. Weisenfeld, Vijay Kumar, Preyas Shah, Deanna M. Church, and David B. Jaffe. Direct determination of diploid genome sequences. *Genome Research*, 27(5):757–767, 2017. doi:10.1101/gr.214874.116.

Linear Time Construction of Indexable Founder Block Graphs

Veli Mäkinen
Department of Computer Science, University of Helsinki, Finland
veli.makinen@helsinki.fi

Bastien Cazaux
Department of Computer Science, University of Helsinki, Finland

Massimo Equi
Department of Computer Science, University of Helsinki, Finland
massimo.equi@helsinki.fi

Tuukka Norri
Department of Computer Science, University of Helsinki, Finland
tuukka.norri@helsinki.fi

Alexandru I. Tomescu
Department of Computer Science, University of Helsinki, Finland
alexandru.tomescu@helsinki.fi

Abstract

We introduce a compact pangenome representation based on an optimal segmentation concept that aims to reconstruct *founder sequences* from a multiple sequence alignment (MSA). Such founder sequences have the feature that each row of the MSA is a recombination of the founders. Several linear time dynamic programming algorithms have been previously devised to optimize segmentations that induce *founder blocks* that then can be concatenated into a set of founder sequences. All possible concatenation orders can be expressed as a *founder block graph*. We observe a key property of such graphs: if the node labels (founder segments) do not repeat in the paths of the graph, such graphs can be indexed for efficient string matching. We call such graphs *segment repeat-free founder block graphs*.

We give a linear time algorithm to construct a segment repeat-free founder block graph given an MSA. The algorithm combines techniques from the founder segmentation algorithms (Cazaux et al. SPIRE 2019) and *fully-functional bidirectional Burrows-Wheeler index* (Belazzougui and Cunial, CPM 2019). We derive a succinct index structure to support queries of arbitrary length in the paths of the graph.

Experiments on an MSA of SARS-CoV-2 strains are reported. An MSA of size 410 × 29811 is compacted in one minute into a segment repeat-free founder block graph of 3900 nodes and 4440 edges. The maximum length and total length of node labels is 12 and 34968, respectively. The index on the graph takes only 3% of the size of the MSA.

2012 ACM Subject Classification Theory of computation → Pattern matching; Theory of computation → Sorting and searching; Theory of computation → Dynamic programming; Applied computing → Genomics

Keywords and phrases Pangenome indexing, founder reconstruction, multiple sequence alignment, compressed data structures, string matching

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.7

Related Version `https://arxiv.org/abs/2005.09342`

Supplementary Material The implementation of several methods proposed in this paper is available at `https://github.com/algbio/founderblockgraphs`.

Funding This work was partly funded by the Academy of Finland (grants 309048 and 322595) and by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 851093, SAFEBIO).

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti; Article No. 7; pp. 7:1–7:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements We wish to thank the anonymous reviewers for useful suggestions to improve the readability.

1 Introduction

Computational pangenomics [13] ponders around the problem of expressing a reference genome of a species in a more meaningful way than as a string of symbols. The basic problem in such generalized representations is that one should still be able to support string matching type of operations on the content. Another problem is that any representation generalizing set of sequences also expresses sequences that may not be part of the real pangenome. That is, a good representation should have a feature to control over-expressiveness and simultaneously support efficient queries.

In this paper, we develop the theory around one promising pangenome representation candidate, the *founder block graph*. This graph is a natural derivative of segmentation algorithms [28, 12] related to *founder sequences* [34].

Consider a set of individuals represented as lists of variations from a common reference genome. Such a set can be expressed as a *variation graph* or as a *multiple sequence alignment*. The former expresses reference as a backbone of an automaton, and adds a subpath for each variant. The latter inputs all variations of an individual to the reference, creating a row for each individual into a multiple alignment. Figure 1 shows an example of both structures with 6 very short genomes.

A multiple alignment of much fewer *founder* sequences can be used to approximate the input represented as a multiple alignment as well as possible, meaning that each original row can be mapped to the founder multiple alignment with a minimum amount of row changes (discontinuities). Finding an optimal set of founders is NP-hard [29], but one can solve relaxed problem statements in linear time [28, 12], which are sufficient for our purposes. As an example on the usefulness of founders, Norri et al. [28] showed that, on a large public dataset of haplotypes of human genome, the solution was able to replace 5009 haplotypes with only 130 founders so that the average distance between row jumps was over 9000 base pairs [28]. This means that alignments of short reads (e.g. 100 bp) very rarely hit a discontinuity, and the space requirement drops from terabytes to just tens of gigabytes. Figure 1 shows such a solution on our toy example.

A *block graph* is a labelled directed acyclic graph consisting of consecutive *blocks*, where a block represents a set of sequences of the same length as parallel (unconnected) nodes. There are edges only from nodes of one block to the nodes of the next block. A *founder block graph* is a block graph with blocks representing the segments of founder sequences corresponding to the optimal segmentation [28]. Fig. 1 visualises such a founder block graph: There the founder set is divided into 3 blocks with the first, the second, and the third containing sequences of length 4, 4, and 3, respectively. The coloured connections between sequences in consecutive blocks define the edges. Such graphs interpreted as automata recognise the input sequences just like variation graphs, but otherwise recognise a much smaller subset of the language. With different optimisation criteria to compute the founder blocks, one can control the expressiveness of this pangenome representation.

In this paper, we show that there is a natural subclass of founder block graphs that admit efficient index structures to be built that support exact string matching on the paths of such graphs. Moreover, we give a linear time algorithm to construct such founder block graphs from a given multiple alignment. The construction algorithm can also be adjusted to produce a subclass of *elastic-degenerate strings* [8], which also support efficient indexing.

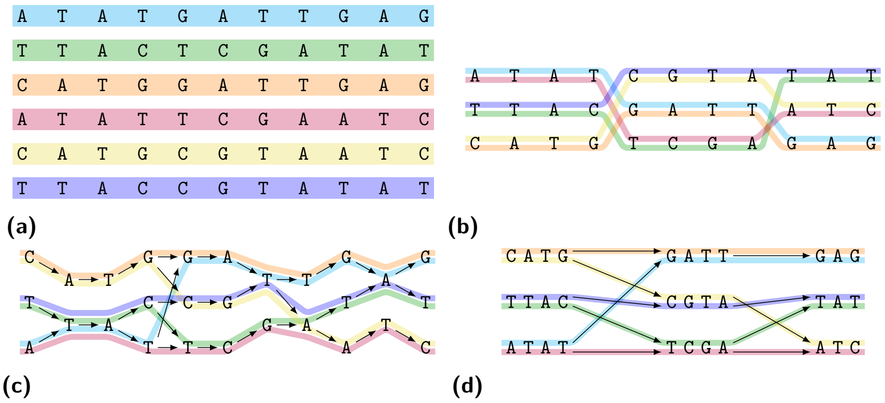


Figure 1 (a) Input multiple alignment, (b) a set of founders with common recombination positions as a solution to a relaxed version of founder reconstruction, (c) a variation graph encoding the input and (d) a founder block graph. Here the input alignment and the resulting variation graph are unrealistically bad; the example is made to illustrate the founders.

The founder block graph definition given above only makes sense if we assume that our input multiple alignment is *gapless*, meaning that the alignment is simply produced by putting strings of equal length under each other, like in Figure 1. We develop the theory around founder block graphs under gapless multiple alignments. However, most of the results can be extended to handle gaps properly.

We start in Sect. 2 by putting the work into the context of related work. In Sect. 3 we introduce the basic notions and tools. In Sect. 4 we study the property of founder block graphs that enable indexing. In Sect. 5 we give the linear time construction algorithm. In Sect. 6 we develop a succinct index structure that supports exact string matching in linear time. In Sect. 7 we consider the general case of having gap symbols in multiple alignment. We report some preliminary experiments in Sect. 8 on the construction and indexing of founder block graphs for a collection of SARS-CoV-2 strains. We consider future directions in Sect. 9.

2 Related work

Indexing directed acyclic graphs (DAGs) for exact string matching on its paths was first studied by Sirén et al. in WABI 2011 [31]. A generalization of *Burrows-Wheeler transform* [11] was proposed that supported near-linear time queries. However, the proposed transformation can grow exponentially in size in the worst case. Many practical solutions have been proposed since then, that either limit the search to short queries or use more time on queries [33, 22, 25, 19, 24, 32]. More recently, such approaches have been captured by the theory on *Wheeler graphs* [18, 20, 2].

Since it is NP-hard to recognize if a given graph is Wheeler [20], it is of interest to look for other graph classes that could provide some indexability functionality. Unfortunately, quite simple graphs turn out to be hard to index [15, 16] (under the Strong Exponential Time Hypothesis). In fact, the reductions by Equi et al. [15, 16] can be adjusted to show that block graphs cannot be indexed in polynomial time to support fast string matching. But as we will see later, further restrictions on block graphs change the situation: We show that there exists a family of founder block graphs that can be indexed in linear time to support linear time queries.

Block graphs have also tight connection to *generalized degenerate* (GD) strings and their *elastic* version. These can also be seen as DAGs with a very specific structure. Matching a GD string is computationally easier and even linear time online algorithms can be achieved to compare two such strings, as analyzed by Alzamel et al. [3]. The elastic counterpart requires more care, as studied by Bernardini et al. [8]. Our results on founder block graphs can be casted on GD strings and elastic strings, as we will show later.

Finally, our indexing solution has connections to succinct representations of *de Bruijn graphs* [10, 9, 7]. Compared to de Bruijn graphs that are cyclic and have limited memory (k-mer length), our solution retains the linear structure of the block graph.

3 Definitions and basic tools

3.1 Strings

We denote integer intervals by $[i..j]$. Let $\Sigma = \{1, \ldots, \sigma\}$ be an alphabet of size $|\Sigma| = \sigma$. A *string* $T[1..n]$ is a sequence of symbols from Σ, i.e. $T \in \Sigma^n$, where Σ^n denotes the set of strings of length n under the alphabet Σ. A *suffix* of string $T[1..n]$ is $T[i..n]$ for $1 \leq i \leq n$. A *prefix* of string $T[1..n]$ is $T[1..i]$ for $1 \leq i \leq n$. A *substring* of string $T[1..n]$ is $T[i..j]$ for $1 \leq i \leq j \leq n$. Substring $T[i..j]$ where $j < i$ is defined as the *empty string*.

The *lexicographic order* of two strings A and B is naturally defined by the order of the alphabet: $A < B$ iff $A[1..i] = B[1..i]$ and $A[i+1] < B[i+1]$ for some $i \geq 0$. If $i + 1 > \min(|A|, |B|)$, then the shorter one is regarded as smaller. However, we usually avoid this implicit comparison by adding *end marker* **0** to the strings.

Concatenation of strings A and B is denoted AB.

3.2 Founder block graphs

As mentioned in the introduction, our goal is to compactly represent a *gapless* multiple sequence alignment (MSA) using a founder block graph. In this section we formalize these concepts.

A *gapless multiple sequence alignment* $\texttt{MSA}[1..m, 1..n]$ is a set of m strings drawn from Σ, each of length n. Intuitively, it can be thought of as a matrix in which each row is one of the m strings. Such a structure can be partitioned into what we call a *segmentation*, that is, a collection of sets of shorter strings that can represent the original alignment.

▶ **Definition 1** (Segmentation). *Let* $\texttt{MSA}[1..m, 1..n]$ *be a gapless multiple alignment and let* $R_1, R_2, \ldots, R_m$ *be the strings in MSA. A* segmentation S *of MSA is a set of* b *sets of strings* $S^1, S^2, \ldots, S^b$ *such that for each* $1 \leq i \leq b$ *there exist interval* $[x^{(i)}..y^{(i)}]$ *such that* $S^i = \{R_t[x^{(i)}..y^{(i)}] \mid 1 \leq t \leq m\}$*. Furthermore, it holds* $x^{(1)} = 1$*,* $y^{(b)} = n$*, and* $y^{(i)} = x^{(i+1)} - 1$ *for all* $1 \leq i < b$*, so that* $S^1, S^2, \ldots, S^b$ covers *the MSA. We call* $W(S^i) = y^{(i)} - x^{(i)} + 1$ *the* width *of* S^i*.*

A segmentation of a MSA can naturally lead to the construction of a founder block graph. Let us first introduce the definition of a block graph.

▶ **Definition 2** (Block Graph). *A* block graph *is a graph* $G = (V, E, \ell)$ *where* $\ell : V \to \Sigma^+$ *is a function that assigns a string label to every node and for which the following properties hold.*

1. $\{V^1, V^2, \ldots, V^b\}$ *is a partition of* V*, that is,* $V = V^1 \cup V^2 \cup \ldots \cup V^b$ *and* $V^i \cap V^j = \emptyset$ *for all* $i \neq j$*;*
2. *if* $(v, w) \in E$ *then* $v \in V^i$ *and* $w \in V^{i+1}$ *for some* $1 \leq i \leq b - 1$*;*
3. *if* $v, w \in V^i$ *then* $|\ell(v)| = |\ell(w)|$ *for each* $1 \leq i \leq b$ *and if* $v \neq w$*,* $\ell(v) \neq \ell(w)$*.*

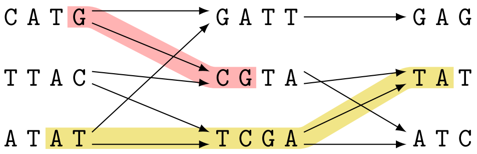


Figure 2 An example of two strings, GCG and ATTCGATA, occurring in $G(S)$.

As a convention we call every V^i a *block* and every $\ell(v)$ a *segment*.

Given a segmentation S of a MSA, we can define the *founder block graph* as a block graph *induced* by S. The idea is to have a graph in which the nodes represents the strings in S while the edges retain the information of how such strings can be recombined to spell any sequence in the original MSA.

▶ **Definition 3** (Founder Block Graph). *A* founder block graph *is a block graph* $G(S) = (V, E, \ell)$ induced *by S as follows: For each $1 \le i \le b$ we have $S^i = \{\ell(v) : v \in V^i\}$ and $(v, w) \in E$ if and only if there exists $i \in [1..b-1]$ and $t \in [1..m]$ such that $v \in V^i$, $w \in V^{i+1}$ and $R_t[j..j + |\ell(v)| + |\ell(w)| - 1] = \ell(v)\ell(w)$ with $j = 1 + \sum_{h=1}^{i-1} W(S^h)$.*

We regard the edges of (founder) block graphs to be directed from left to right. Consider a path P in $G(S)$ between any two nodes. The label $\ell(P)$ of P is the concatenation of labels of the nodes in the path. Let Q be a query string. We say that Q *occurs* in $G(S)$ if Q is a substring of $\ell(P)$ for any path P of $G(S)$. Figure 2 illustrates such queries.

In our example in Figure 1, the intervals corresponding to the segmentation would be $[1..4]$, $[5..8]$, $[9..11]$, and the induced founder block graph has thus 3 blocks with 9 nodes and 11 edges in total.

3.3 Basic tools

A *trie* [14] of a set of strings is a rooted directed tree with outgoing edges of each node labeled by distinct characters such that there is a root to leaf path spelling each string in the set; shared part of the root to leaf paths to two different leaves spell the common prefix of the corresponding strings. Such a trie can be computed in $O(N \log \sigma)$ time, where N is the total length of the strings, and it supports string queries that require $O(q \log \sigma)$ time, where q is the length of the queried string.

An *Aho-Corasick automaton* [1] is a trie of a set of strings with additional pointers (fail-links). While scanning a query string, these pointers (and some shortcut links on them) allow to identify all the positions in the query at which a match for any of the strings occurs. Construction of the automaton takes the same time as that of the trie. Queries take $O(q \log \sigma + \texttt{occ})$ time, where $\texttt{occ}$ is the number of matches.

A *suffix array* [26] of string T is an array $\texttt{SA}[1..n+1]$ such that $\texttt{SA}[i] = j$ if $T'[j..n+1]$ is the j-th smallest suffix of string $T' = T\mathbf{0}$, where $T \in \{1, 2, \ldots, \sigma\}^n$, and $\mathbf{0}$ is the end marker. Thus, $\texttt{SA}[1] = n + 1$.

Burrows-Wheeler transform $\mathrm{BWT}[1..n+1]$ [11] of string T is such that $\mathrm{BWT}[i] = T'[\texttt{SA}[i] - 1]$, where $T' = T\mathbf{0}$ and $T'[-1]$ is regarded as $T'[n+1] = \mathbf{0}$.

A *bidirectional BWT index* [30, 6] is a succinct index structure based on some auxiliary data structures on BWT. Given a string $T \in \Sigma^n$, with $\sigma \le n$, such index occupying $O(n \log \sigma)$ bits of space can be built in $O(n)$ time and it supports finding in $O(q)$ time if a query string $Q[1..q]$ appears as substring of T [6]. Moreover, such query returns an interval pair $([i..j], [i'..j'])$ such that suffixes of T starting at positions $\texttt{SA}[i], \texttt{SA}[i+1], \ldots, \texttt{SA}[j]$ share a common prefix matching the query. Interval $[i'..j']$ is the corresponding interval in the suffix array of the reverse of T. Let $([i..j],[i'..j'])$ be the interval pair corresponding to query substring $Q[l..r]$. A *bidirectional backward step* updates the interval pair $([i..j],[i'..j'])$ to the corresponding interval pair when the query substring $Q[l..r]$ is extended to the left into $Q[l-1..r]$ or to the right into $Q[l..r+1]$. This takes constant time [6]. A *fully-functional bidirectional BWT index* [4] expands the steps to allow contracting symbols from the left or from the right. That is, substring $Q[l..r]$ can be modified into $Q[l+1..r]$ or to $Q[l..r-1]$ and the the corresponding interval pair can be updated in constant time.

Among the auxiliary structures used in BWT-based indexes, we explicitly use the *rank* and *select* structures: String $B[1..n]$ from binary alphabet is called a *bitvector*. Operation $\texttt{rank}(B,i)$ returns the number of 1s in $B[1..i]$. Operation $\texttt{select}(B,j)$ returns the index i containing the j-th 1 in B. Both queries can be answered in constant time using an index requiring $o(n)$ bits in addition to the bitvector itself [23].

4 Subclass of founder block graphs admitting indexing

We now show that there exists a family of founder block graphs that admit a polynomial time constructable index structure supporting fast string matching. First, a trivial observation: the input multiple alignment is a founder block graph for the segmentation consisting of only one segment. Such founder block graph (set of sequences) can be indexed in linear time to support linear time string matching [6]. Now, the question is, are there other segmentations that allow the resulting founder block graph to be indexed in polynomial time? We show that this is the case.

▶ **Definition 4.** *Founder block graph $G(S)$ is* segment repeat-free *if each $\ell(v)$ for $v \in V$ occurs exactly once in $G(S)$.*

Our example graph (Fig. 1) is not quite segment repeat-free, as `TAT` occurs also as substring of paths starting with `ATAT`.

▶ **Proposition 5.** *Segment repeat-free founder block graphs can be indexed in polynomial time to support polynomial time string queries.*

To prove the proposition, we construct such an index and show how queries can be answered efficiently.

Let $P(v)$ be the set of all paths starting from node v and ending in a sink node. Let $P(v,i)$ be the set of *suffix path labels* $\{\ell(L)[i..] \mid L \in P(v)\}$ for $1 \leq i \leq |\ell(v)|$. Consider sorting $\mathcal{P} = \cup_{v \in V, 1 \leq i \leq |\ell(v)|} P(v,i)$ in lexicographic order. Then one can binary search any query string Q in $\mathcal{P}$ to find out if it occurs in $G(S)$ or not. The problem with this approach is that $\mathcal{P}$ is of exponential size.

However, if we know that $G(S)$ is segment repeat-free, we know that the lexicographic order of $\ell(L)[i..]$, $L \in P(v)$, is fully determined by the prefix $\ell(v)[i..|\ell(v)|]\ell(w)$ of $\ell(L)[i..]$, where w is the node following v on the path L. Let $P'(v,i)$ denote the set of suffix path labels cut in this manner. Now the corresponding set $\mathcal{P}' = \cup_{v \in V, 1 \leq i \leq |\ell(v)|} P'(v,i)$ is no longer of exponential size. Consider again binary searching a string Q on sorted $\mathcal{P}'$. If Q occurs in $\mathcal{P}'$ then it occurs in $G(S)$. If not, Q has to have some $\ell(v)$ for $v \in V$ as its substring in order to occur in $G(S)$.

To figure out if Q contains $\ell(v)$ for some $v \in V$ as its substring, we build an Aho-Corasick automaton [1] for $\{\ell(v) \mid v \in V\}$. Scanning this automaton takes $O(|Q| \log \sigma)$ time and returns such $v \in V$ if it exists.

To verify such potential match, we need several tries [14]. For each $v \in V$, we build tries $\mathcal{R}(v)$ and $\mathcal{F}(v)$ on the sets $\{\ell(u)^{-1} \mid (u,v) \in E\}$ and $\{\ell(w) \mid (v,w) \in E\}$, respectively, where X^{-1} denotes the reverse $x_{|X|}x_{|X|-1} \cdots x_1$ of string $X = x_1x_2 \cdots x_{|X|}$.

Assume now we have located (using the Aho-Corasick automaton) $v \in V$ with $\ell(v)$ such that $\ell(v) = Q[i..j]$, where v is at the k-th block of $G(S)$. We continue searching $Q[1..i-1]$ from right to left in trie $\mathcal{R}(v)$. If we reach a leaf after scanning $Q[i'..i-1]$, we continue the search with $Q[1..i'-1]$ on trie $\mathcal{R}(v')$, where $v' \in V$ is the node at block $k-1$ of $G(S)$ corresponding to the leaf we reached in the trie. If the search succeeds after reading $Q[1]$ we have found a path in $G(S)$ spelling $Q[1..j]$. We repeat the analogous procedure with $Q[j..m]$ starting from trie $\mathcal{F}(v)$. That is, we can verify a candidate occurrence of Q in $G(S)$ in $O(|Q|\log\sigma)$ time, as the search in the tries takes $O(\log\sigma)$ time per step. Note however, that there could be several labels $\ell(v)$ occurring as substrings of Q, so we need to do the verification process for each one of them separately. There can be at most $|Q|$ such candidate occurrences, due to the distinctness of node labels in $G(S)$. In total, this search can take at most $O(|Q|^2\log\sigma)$ time.

We are now ready to specify a theorem that reformulates Proposition 5 in detailed form.

▶ **Theorem 6.** *Let $G = (V,E)$ be a segment repeat-free founder block graph with blocks $V^1, V^2, \ldots, V^b$ such that $V = V^1 \cup V^2 \cup \cdots \cup V^b$. We can preprocess an index structure for G in $O((N+W|E|)\log\sigma)$ time, where $\{1,\ldots,\sigma\}$ is the alphabet for node labels, $W = \max_{v\in V} |\ell(v)|$, and $N = \sum_{v\in V} |\ell(v)|$. Given a query string $Q[1..q] \in \{1,\ldots,\sigma\}^q$, we can use the index structure to find out if Q occurs in G. This query takes $O(|Q|^2\log\sigma)$ time.*

Proof. With preprocessing time $O(N\log\sigma)$ we can build the Aho-Corasick automaton [1]. The tries can be built in $O(\log\sigma)(\sum_{v\in V}(\sum_{(u,v)\in E}|\ell(u)|+\sum_{(v,w)\in E}|\ell(w)|)) = O(|E|W\log\sigma)$ time. The required queries on these structures take $O(|Q|^2\log\sigma)$ time.

To avoid the costly binary search in sorted $\mathcal{P}'$, we instead construct the bidirectional BWT index [6] for the concatenation $C = \prod_{i\in\{1,2,\ldots,b\}}\prod_{v\in V^i,(v,w)\in E}\ell(v)\ell(w)\mathbf{0}$. Concatenation C is thus a string of length $O(|E|W)$ from alphabet $\{\mathbf{0},1,2,\ldots,\sigma\}$. The bidirectional BWT index for C can be constructed in $O(|C|)$ time, so that in $O(|Q|)$ time, one can find out if Q occurs in C [6]. This query equals that of binary search in $\mathcal{P}'$. ◀

We remark that founder block graphs have a connection with *generalized degenerate strings* (GD strings) [3]. In a GD string, sets of strings of equal length are placed one after the other to represent in a compact way a bigger set of strings. Such set contains all possible concatenations of those strings, which are obtained by scanning the individual sets from left to right and selecting one string from each set. The length of the strings in a specific set is called *width*, and the sum of all the widths of all sets in a GD string is the *total width*. Given two GD strings of the same total width it is possible to determine if the intersection of the sets of strings that they represent is non empty in linear time in the size of the GD strings [3]. Thus, the special case in which one of the two GD string is just a standard string can be seen also as a special case of a founder block graph in which every segment is fully connected with the next one and the length of the query string is equal to the maximal length of a path in the graph.

We consider the question of indexing GD strings (fully connected block graphs) to search for queries Q shorter than the total width. We can exploit the segment repeat-free property to yield such an index.

▶ **Theorem 7.** *Let $G = (V,E)$ be a* segment repeat-free GD string *a.k.a. a fully connected segment repeat-free founder block graph with blocks $V^1, V^2, \ldots, V^b$ such that $V = V^1 \cup V^2 \cup \cdots \cup V^b$ and $(v,w) \in E$ for all $v \in V^i$ and $w \in V^{i+1}$, $1 \le i < b$. We can preprocess an index* *structure for G in $O((N+W|E|)\log\sigma)$ time, where $\{1,\ldots,\sigma\}$ is the alphabet for node labels, $W = \max_{v\in V}\ell(v)$, and $N = \sum_{v\in V}\ell(v)$. Given a query string $Q[1..q] \in \{1,\ldots,\sigma\}^q$, we can use the index structure to find out if Q occurs in G. This query takes $O(|Q|\log\sigma)$ time.*

Proof. Recall the index structure of Theorem 6. for the case of GD strings, we can simplify it as follow.

We keep the same BWT index structure and the Aho-Corasick automaton, but we do not need any tries. After finding at most $|Q|$ occurrences of substrings of Q in the graph using the Aho-Corasick automaton on node labels, we mark the matching blocks accordingly. If 2 marked blocks have exactly one marked neighboring block and $|Q|-2$ blocks have 2 marked neighboring blocks, then we have found an occurrence, otherwise not. ◀

Observe that $\max(N, W|E|) \le mn$, where m and n are the number of rows and number of columns, respectively, in the multiple sequence alignment from where the founder block graph is induced. That is, the index construction algorithms of the above theorems can be seen to be almost linear time in the (original) input size. We study succinct variants of these indexes in Sect. 6, and also improve the construction and query times to linear as side product.

5 Construction of segment repeat-free founder block graphs

Now that we know how to index segment repeat-free founder block graphs, we turn our attention to the construction of such graphs from a given MSA. For this purpose, we will adapt the dynamic programming segmentation algorithms for founders [28, 12].

The idea is as follows. Let S be a segmentation of $\texttt{MSA}[1..m, 1..n]$. We say S is *valid* if it induces a segment repeat-free founder block graph $G(S) = (V, E)$. We build such valid S for prefixes of MSA from left to right, minimizing the maximum block length needed.

5.1 Characterization lemma

Given a segmentation S and founder block graph $G(S) = (V, E)$ induced by S, we can ensure that it is valid by checking if, for all $v \in V$, $\ell(v)$ occurs in the rows of the MSA only in the interval of the block V^i, where V^i is the block of V such that $v \in V^i$.

▶ **Lemma 8** (Characterization). *Let $x^{(i)} = 1+\sum_{h=1}^{i-1} W(S^h)$. A segmentation S is valid if and only if, for all blocks $V^i \subseteq V$, $1 \le t \le m$ and $j \neq x^{(i)}$, if $v \in V^i$ then $R_t[j..j+|\ell(v)|-1] \neq \ell(v)$.*

Proof. To see that this is a necessary condition for the validity of S, notice that each row of MSA can be read through G, so if $\ell(v)$ occurs elsewhere than inside the block, then these extra occurrences make S invalid. To see that this is a sufficient condition for the validity of S, we observe the following:

a) For all $(v, w) \in E$, $\ell(v)\ell(w)$ is a substring of some row of the input MSA.

b) Let $(x, u), (u, y) \in E$ be two edges such that $U = \ell(x)\ell(u)\ell(y)$ is not a substring of any row of input MSA. Then any substring of U either occurs in some row of the input MSA or it includes $\ell(u)$ as its substring.

c) Thus, any substring of a path in G either is a substring of some row of the input MSA, or it includes $\ell(u)$ of case b) as its substring.

d) Let α be a substring of a path of G that includes $\ell(u)$ as its substring. If $\ell(z) = \alpha$ for some $z \in V$, then $\ell(u)$ appears at least twice in the MSA. Substring α makes S invalid only if $\ell(u)$ does. ◀

5.2 From characterization to a segmentation

Among the valid segmentations, we wish to select an *optimal* segmentation under some goodness criteria. Earlier work [28, 12] has considered various goodness criteria, solving the associated segmentation problems in linear time. In this paper, we focus on minimizing the maximum width of the segments. For example, the (non-valid) segmentation in Figure 1 has score 4, as the intervals of the segments are $[1..4], [5..8]$, and $[9..11]$. That is, as $W(S^1) = 4-1+1 = 4$, $W(S^2) = 8-5+1 = 4$, $W(S^3) = 11-9+1 = 3$, the maximum of these is 4. This criteria is analogous to one studied by Cazaux et al. [12], and appears to be the most natural one to be studied together with validity constraint; both deal directly with the segment intervals.

Let us now develop a dynamic programming recurrence for finding the minimum scoring valid segmentation with the score being the maximum width. Let $s(j')$ be the score of a minimum scoring valid segmentation $S^1, S^2, \ldots, S^b$ of prefix $\texttt{MSA}[1..m, 1..j']$, where the score is defined as $\max\limits_{i:1 \le i \le b} W(S^i)$. We can compute

$$s(j) = \min_{j':0 \le j' \le v(j)} \max(j-j', s(j')), \tag{1}$$

where $v(j)$ is the largest integer such that segment $\texttt{MSA}[1..m, v(j)+1..j]$ is valid. The segment is valid iff each substring $\texttt{MSA}[i, v(j)+1..j]$, for $1 \le i \le m$, occurs as many times in $\texttt{MSA}[1..m, v(j)+1..j]$ as in the whole MSA. If such $v(j)$ does not exist for some j, we set $v(j) = 0$. The intuition is that $[j'+1..j]$ forms the last valid segment of the segmentation, and since $s(j')$ is the score of optimal valid segmentation of $\texttt{MSA}[1..m, 1..j']$, the maximum of $s(j')$ and the length of the last segment decides the score $s(j)$. To initialize the recurrence, one can set $s(0) = 0$ and $s(j) = \infty$ for $1 \le j \le J$ for which $v(j)$ is undefined. The recurrence can then be applied for $j \in [J+1..n]$.

We can compute the score $s(n)$ of the optimal segmentation of MSA in $O(ns_{\texttt{max}})$ time after preprocessing values $v(j)$ in $O(mns_{\texttt{max}} \log \sigma)$ time, where $s_{\texttt{max}} = \max_{j:v(j)>0} s(j)$. For the former, one can start comparing $\max(j-j', s(j'))$ from $j' = v(j)$ decreasing j' by one each step, and then the value $j-j'$ grows bigger than $s(j')$ at latest after $s(j)-(j-v(j)) \le s(j)$ steps. For preprocessing, we build the bidirectional BWT index of the MSA in $O(mn)$ time [6]. At column j, consider the trie containing the reverse of the rows of $M[1..m, 1..j]$. Search the trie paths from the bidirectional BWT index until the number of leaves in each trie subtree equals the length of the corresponding BWT interval. Let j' be the column closest to j where this holds for all trie paths. Then one can set $v(j) = j'$. The $O(m(j-v(j)) \log \sigma)$ time construction of the trie has to be repeated for each column. As $j - v(j) \le s(j)$, the claimed preprocessing time follows.

5.3 Faster preprocessing

We can do the preprocessing in $O(mn)$ time.

▶ **Theorem 9.** *Given a multiple sequence alignment* $\texttt{MSA}[1\ldots m, 1\ldots n]$*, values* $v(j)$ *for each* $1 \le j \le n$ *can be computed in* $O(mn)$ *time, where* $v(j)$ *is the largest integer such that segment* $\texttt{MSA}[1..m, v(j)+1..j]$ *is valid.*

Proof. Let us build the bidirectional BWT index [6] of MSA rows concatenated into one long string. We will run several algorithms in synchronization over this BWT index, but we explain them first as if they would be run independently.

Algorithm 1 searches in parallel all rows from right to left advancing each by one position at a time. Let k be the number of parallel of steps done so far. We can maintain a bitvector M that at k-th step stores $M[i] = 1$ iff $BWT[i]$ is the k-th last symbol of some row.

Algorithm 2 uses the *variable length sliding window* approach of Belazzougui and Cunial [4] to compute values $v(j)$. Let the first row of MSA be $T[1..n]$. Search $T[1..n]$ backwards in the fully-functional bidirectional BWT index [4]. Stop the search at $T[j'+1..n]$ such that the corresponding BWT interval $[i'..i]$ contains only suffixes originating from column $j'+1$ of the MSA, that is, spelling $\texttt{MSA}[a, j'+1..n]$ in the concatenation, for some rows a. Set $v^b(n) = j'$ for row $b = 1$. Contract $T[n]$ from the search string and modify BWT interval accordingly [4]. Continue the search (decreasing j' by one each step) to find $T[j'+1..n-1]$ s.t. again the corresponding BWT interval $[i'..i]$ contains only suffixes originating from column $j'+1$. Update $v^b(n-1) = j'$ for row $b = 1$. Continue like this throughout T. Repeat the process for all remaining rows $b \in [2..m]$, to compute $v^2(j), v^3(j), \ldots, v^m(j)$ for all j. Set $v(j) = \min_i v^i(j)$ for all j.

Let us call the instances of the Algorithm 2 run on the rest of the rows as Algorithms $3, 4, \ldots, m+1$.

Let the current BWT interval in Algorithms 2 to $m+1$ be $[j'+1..j]$. The problematic part in them is checking if the corresponding *active* BWT intervals $[i'_a..i_a]$ for Algorithms $a \in \{2, 3, \ldots, m+1\}$ contain only suffixes originating from column $j'+1$. To solve this, we run Algorithm 1 as well as Algorithms 2 to $m+1$ in synchronization so that we are at the k-th step in Algorithm 1 when we are processing interval $[j'+1..j]$ in rest of the algorithms, for $k = n - j'$. In addition, we maintain bitvectors B and E such that $B[i'_a] = 1$ and $E[i_a] = 1$ for $a \in \{2, 3, \ldots, m+1\}$. For each $M[i]$ that we set to 1 at step k with $B[i] = 0$ and $E[i] = 0$, we check if $M[i-1] = 1$ and $M[i+1] = 1$. If and only if this check fails on any i, there is a suffix starting outside column $j'+1$. This follows from the fact that each suffix starting at column $j'+1$ must be contained in exactly one of the distinct intervals of the set $I = \{[i'_a..i_a]\}_{a \in \{2,3\ldots m+1\}}$. This is because I cannot contain nested interval pairs as all strings in segment $[j'+1..j]$ of MSA are of equal length, and thus their BWT intervals cannot overlap except if the intervals are exactly the same.

Finally, the running time is $O(mn)$, since each extend-left and contract-right operations take constant time [4], and since the bitvectors are manipulated locally only on indexes that are maintained as variables during the execution. ◀

5.4 Faster main algorithm

Recall Eq. (1). Before proceeding to the involved optimal solution, we give some insights by first improving the running time to logarithmic per entry.

As it holds $v(1) \leq v(2) \leq \cdots \leq v(n)$, the range where the minimum is taken grows as j grows. Now, $[j'..j'+s(j')]$ can be seen as the *effect range* of $s(j')$: for columns $j > j'+s(j')$ the maximum from the options is $j - j'$. Consider maintaining (key, value) pairs $(s(j')+j', s(j'))$ in a binary search tree (BST). When computing $s(j)$ we should have pairs $(s(j')+j', s(j'))$ for $1 \leq j' \leq v(j)$ in BST. Value $s(j)$ can be computed by taking range minimum on BST values with keys in range $[j..\infty]$. Such query is easy to solve in $O(\log n)$ time. If there is nothing in the interval, $s(j) = j - v(j)$. Since this is semi-open interval on keys in range $[1 \ldots 2n]$, BST can be replaced by van Emde Boas tree to obtain $O(n \log\log n)$ time computation of all values [17]. Alternatively, we can remove elements from the BST once they no longer can be answers to queries, and we can get $O(n \log s_{\texttt{max}})$ solution. To obtain better running time, we need to exploit more structural properties of the recurrence.

Cazaux et al. [12] considered a similar recurrence and gave a linear time solution for it. In what follows we modify that technique to work with valid ranges.

For j between 1 and n, we define

$$x(j) = \max Argmin_{j' \in [1..v(j)]} \max(j - j', s(j'))$$

▶ **Lemma 10.** *For any $j \in [1..n-1]$, we have $x(j) \leq x(j+1)$.*

Proof. By the definition of $x(.)$, for any $j \in [1..n]$, we have for $j' \in [1..x(j)-1]$, $\max(j - j', s(j')) \geq \max(j - x(j), s(x(j)))$ and for $j' \in [x(j)+1..v(j)]$, $\max(j - j', s(j')) > \max(j - x(j), s(x(j)))$.

We assume that there exists $j \in [1..n-1]$, such that $x(j+1) < x(j)$. In this case, $x(j+1) \in [1..x(j)-1]$ and we have $\max(j - x(j+1), s(x(j+1))) \geq \max(j - x(j), s(x(j)))$. As $v(j+1) \geq v(j)$, $x(j) \in [x(j+1)+1..v(j+1)]$ and thus $\max(j+1-x(j+1), s(x(j+1))) < \max(j+1-x(j), s(x(j)))$. As $x(j+1) < x(j)$, we have $j - x(j+1) > j - x(j)$. To simplify the proof, we take $A = j - x(j+1)$, $B = s(x(j+1))$, $C = j - x(j)$ and $D = s(x(j))$. Hence, we have $\max(A, B) \geq \max(C, D)$, $\max(A+1, B) < \max(C+1, D)$ and $A > C$. Now we are going to prove that this system admits no solution.

- Case where $A = \max(A, B)$ and $C = \max(C, D)$. As $A > C$, we have $A + 1 > C + 1$ and thus $\max(A+1, B) > \max(C+1, D)$ which is impossible because $\max(A+1, B) < \max(C+1, D)$.
- Case where $B = \max(A, B)$ and $C = \max(C, D)$. We can assume that $B > A$ (in the other case, we take $A = \max(A, B)$) and as $A > C$, we have $B > C + 1$ and thus $\max(A+1, B) > \max(C+1, D)$ which is impossible because $\max(A+1, B) < \max(C+1, D)$.
- Case where $A = \max(A, B)$ and $D = \max(C, D)$. We have $A > D$ and $A > C$, thus $\max(A+1, B) > \max(C+1, D)$ which is impossible because $\max(A+1, B) < \max(C+1, D)$.
- Case where $B = \max(A, B)$ and $D = \max(C, D)$. We have $B \geq D$ and $A > C$, thus $\max(A+1, B) \geq \max(C+1, D)$ which is impossible because $\max(A+1, B) < \max(C+1, D)$.

◀

▶ **Lemma 11.** *By initialising $s(1)$ to a threshold K, for any $j \in [1..n]$, we have $s(j) \leq \max(j, K)$.*

Proof. We are going to show by induction. The base case is obvious because $s(1) = K \leq \max(1, K)$. As $s(j) = \min_{j' : 1 \leq j' \leq v(j)} \max(j - j', s(j'))$, by using induction, $s(j) \leq \min_{j' : 1 \leq j' \leq v(j)} \max(j, K) \leq \max(j, K)$ ◀

Thanks to Lemma 11, by taking the threshold $K = n + 1$, the values $s(j)$ are in $O(n)$ for all j in $[1..n-1]$.

▶ **Lemma 12.** *Given $j^\star \in [x(j-1)+1..v(j)]$, we can compute in constant time if*

$$j^\star = \max Argmin_{j' \in [j^\star..v(j)]} \max(j - j', s(j')).$$

Proof. We need just to compare $k = \max(j - j^\star, s(j^\star))$ and $s(j^\diamond)$ where $j^\diamond$ is in $Argmin_{j' \in [j^\star+1..v(j)]} s(j')$. If k is smaller than $s(j^\diamond)$, k is smaller than all the $s(j')$ with $j' \in [j^\star+1..v(j)]$ and thus for all $\max(j - j', s(j'))$. Hence we have $j^\star = \max Argmin_{j' \in [j^\star..v(j)]} \max(j - j', s(j'))$.

Otherwise, $s(j^\diamond) \geq k$ and as $k \geq j - j^\star$, $\max(j - j^\diamond, s(j^\diamond)) \geq k$. In this case $j^\star \neq \max Argmin_{j' \in [j^\star..v(j)]} \max(j - j', s(j'))$. By using the constant time semi-dynamic range maximum query by Cazaux et al. [12] on the array $s(.)$, we can obtain in constant time $j^\diamond$ and thus check the equality in constant time. ◀

▶ **Theorem 13.** *The values $s(j)$, for all $j \in [1..n]$, can be computed in $O(n)$ time after an $O(nm)$ time preprocessing.*

Proof. We begin by preprocessing all the values of $v(j)$ in $O(mn)$ (Theorem 9). The idea is to compute all the values $s(j)$ by increasing order of j and by using the values $x(j)$. For each $j \in [1..n]$, we check all the j' from $x(j-1)$ to $v(j)$ with the equality of Lemma 12 until one is true and thus corresponds to $x(j)$. Finally, we add $s(j) = \max(j - x(j), s(x(j)))$ to the constant time semi-dynamic range maximum query and continue with $j + 1$. ◀

6 Succinct index for segment-free founder block graphs

Recall the indexing solutions of Sect. 4 and the definitions from Sect. 3.

We now show that explicit tries and Aho-Corasick automaton can be replaced by some auxiliary data structures associated with the Burrows-Wheeler transformation of the concatenation $C = \prod_{i \in \{1,2,\ldots,b\}} \prod_{v \in V^i, (v,w) \in E} \ell(v)\ell(w)\mathbf{0}$.

Consider interval $\mathtt{SA}[i..k]$ in the suffix array of C corresponding to suffixes having $\ell(v)$ as prefix for some $v \in V$. From the segment repeat-free property it follows that this interval can be split into two subintervals, $\mathtt{SA}[i..j]$ and $\mathtt{SA}[j+1..k]$, such that suffixes in $\mathtt{SA}[i..j]$ start with $\ell(v)\mathbf{0}$ and suffixes in $\mathtt{SA}[j+1..k]$ start with $\ell(v)\ell(w)$, where $(v,w) \in E$. Moreover, $\mathrm{BWT}[i..j]$ equals multiset $\{\ell(u)[|\ell(u)|-1] \mid (u,v) \in E\}$ *sorted in lexicographic order.* This follows by considering the lexicographic order of suffixes $\ell(u)[|\ell(u)|-1]\ell(v)\mathbf{0}\ldots$ for $(u,v) \in E$. That is, $\mathrm{BWT}[i..j]$ (interpreted as a set) represents the children of the root of the trie $\mathcal{R}(v)$ considered in Sect. 4.

We are now ready to present the search algorithm that uses only the BWT of C and some small auxiliary data structures. We associate two bitvectors B and E to the BWT of C as follows. We set $B[i] = 1$ and $E[k] = 1$ iff $\mathtt{SA}[i..k]$ is maximal interval with all suffixes starting with $\ell(v)$ for some $v \in V$.

Consider the backward search with query $Q[1..q]$. Let $\mathtt{SA}[j'..k']$ be the interval after matching the shortest suffix $Q[q'..q]$ such that $\mathrm{BWT}[j'] = \mathbf{0}$. Let $i = \mathtt{select}(B, \mathtt{rank}(B, j'))$ and $k = \mathtt{select}(E, \mathtt{rank}(B, j'))$. If $i \leq j'$ and $k' \leq k$, index j' lies inside an interval $\mathtt{SA}[i..k]$ where all suffixes start with $\ell(v)$ for some v. We modify the range into $\mathtt{SA}[i..k]$, and continue with the backward step on $Q[q'-1]$. We check the same condition in each step and expand the interval if the condition is met. Let us call this procedure *expanded backward search.*

We can now strictly improve Theorems 6 and 7 as follows.

▶ **Theorem 14.** *Let $G = (V,E)$ be a segment repeat-free founder block graph (or a segment repeat-free GD string) with blocks $V^1, V^2, \ldots, V^b$ such that $V = V^1 \cup V^2 \cup \cdots \cup V^b$. We can preprocess an index structure for G occupying $O(W|E|\log\sigma)$ bits in $O(W|E|)$ time, where $\{1,\ldots,\sigma\}$ is the alphabet for node labels and $W = \max_{v \in V} \ell(v)$. Given a query string $Q[1..q] \in \{1,\ldots,\sigma\}^q$, we can use expanded backward search with the index structure to find out if Q occurs in G. This query takes $O(|Q|)$ time.*

Proof. (sketch) As we expand the search interval in BWT, it is evident that we still find all occurrences for short patterns that span at most two nodes, like in the proof of Theorem 6. We need to show that a) the expansions do not yield spurious occurrences for such short

Table 1 Segment of MSA with and without gaps.

segment of MSA	gaps removed
-AC-CGATC-	ACCGATC
-A-CCGATCC	ACCGATCC
AAC-CGATC-	AACCGATC
AAC-CGA-C-	AACCGAC

patterns and b) the expansions yield exactly the occurrences for long patterns that we earlier found with the Aho-Corasick and tries approach. In case b), notice that after an expansion step we are indeed in an interval $\mathtt{SA}[i..k]$ where all suffixes match $\ell(v)$ and thus corresponds to a node $v \in V$. The suffix of the query processed before reaching interval $\mathtt{SA}[i..k]$ must be at least of length $|\ell(v)|$. That is, to mimic Aho-Corasick approach, we should continue with the trie $\mathcal{R}(v)$. This is identical to taking backward step from $\mathrm{BWT}[i..k]$, and continuing therein to follow the rest of this implicit trie.

To conclude case b), we still need to show that we reach all the same nodes as when using Aho-Corasick, and that the search to other direction with $\mathcal{L}(v)$ can be avoided. These follow from case a), as we see.

In case a), before doing the first expansion, the search is identical to the original algorithm in the proof of Theorem 6. After the expansion, all matches to be found are those of case b). That is, no spurious matches are reported. Finally, no search interval can include two distinct node labels, so the search reaches the only relevant node label, where the Aho-Corasick and trie search simulation takes place. We reach all such nodes that can yield a full match for the query. ◀

7 Gaps in multiple alignment

We have so far assumed that our input is a gapless multiple alignment. Let us now consider how to extend the results to the general case. The idea is that gaps are only used in the segmentation algorithm to define the valid ranges, and that is the only place where special attention needs to be taken; elsewhere, whenever a substring from MSA rows is read, gaps are treated as empty strings. That is, A-GC-TA- becomes AGCTA.

It turns out that allowing gaps in MSA indeed makes the computation of valid ranges more difficult. To see this, consider the example in Table 1. The second column in Table 1 shows the sequences after gaps are removed. Without even seeing the rest of the MSA, one can see that this is not a valid block, as the first string is a prefix of the second. With gapless MSAs this was not possible and the algorithm in Sect. 5.3 exploited this fact.

Modifying the construction algorithm to handle gaps properly is possible, but non-trivial. We leave this extension to journal version; see however Sections 8 and 9 for some further insights.

Despite the computation of the segmentation is affected by gaps in MSA, once such valid segmentation is found, the rest of the results stay unaffected. All the proposed definitions extend to this interpretation by just omitting gap symbols when reading the strings. The consequence for founder block graph is that strings inside a block can be of variable length. Interestingly, with this interpretation Theorem 7 can be expressed with GD strings replaced by *elastic strings* [8].

8 Implementation and experiments

We implemented several methods proposed in this paper. The implementation is available at `https://github.com/algbio/founderblockgraphs`. Some preliminary experiments are reported below.

8.1 Construction

We implemented the founder block graph construction algorithm of Sect. 5.2 with the faster preprocessing routine of Sect. 5.3. In place of fully-functional bidirectional BWT index, we used similar routines implemented in compressed suffix trees of SDSL library [21]; this affects the theoretical running time by a logarithm factor.

To test the implementation we downloaded 1484 strains of SARS-CoV-2 strains stored in NCBI database.[1] We created a multiple sequence alignment of the strains using *ViralMSA*[27]. We then filtered out rows that contained gaps or N's. We were left with a multiple sequence alignment of 410 rows and 29811 columns. Our algorithm took 58 seconds to produce the optimal segmentation on Intel(R) Xeon(R) CPU, E5-2690, v4, 2.60GHz. There were 3352 segments in the segmentation, the maximum segment length was 12, and the maximum number of founder segments in a block was 12. The founder block graph had 3900 nodes and 4440 edges. The total length of node labels was 34968. The graph size was thus less than 1% of the MSA size.

We also implemented support to construct founder block graphs for general MSA's that contain gaps. The modification to the gapless case was that nested BWT intervals needed to be detected. We stopped left-extension as soon as the BWT intervals contained no other repeats than those caused by nestedness. This left valid range undefined on such columns, but for the rest the valid range can still be computed correctly (undefined values could be postprocessed using matching statistics [5] on all pairs of prefixes preceding suffixes causing nested intervals). Initial experiments show very similar behaviour to the gapless case, but we defer further experiments until the implementation is mature enough.

8.2 Indexing

We implemented the succinct indexing approach of Sect. 6. On the founder block graph of the previous experiment, the index occupied 87 KB. This is 3% of the original input size, as the encoding of the input MSA with 2 bits per nucleotide takes 2984 KB.

Figure 3 shows an experiment with indexes built on different size samples of the MSA rows, and with querying patterns of varying length sampled from the same rows. As can be seen, the query times are not affected by the size of the MSA samples (showing independence of the input MSA), but only on their length (showing linear dependency on the query length).

9 Discussion

One characterization of our solution is that we compact those vertical repeats in MSA that are not horizontal repeats. This can be seen as positional extension of variable order de Bruijn graphs. Also, our solution is parameter-free unlike de Bruijn approaches that always need some threshold k, even in the variable order case.

[1] `https://www.ncbi.nlm.nih.gov/`, accessed 24.04.2020.

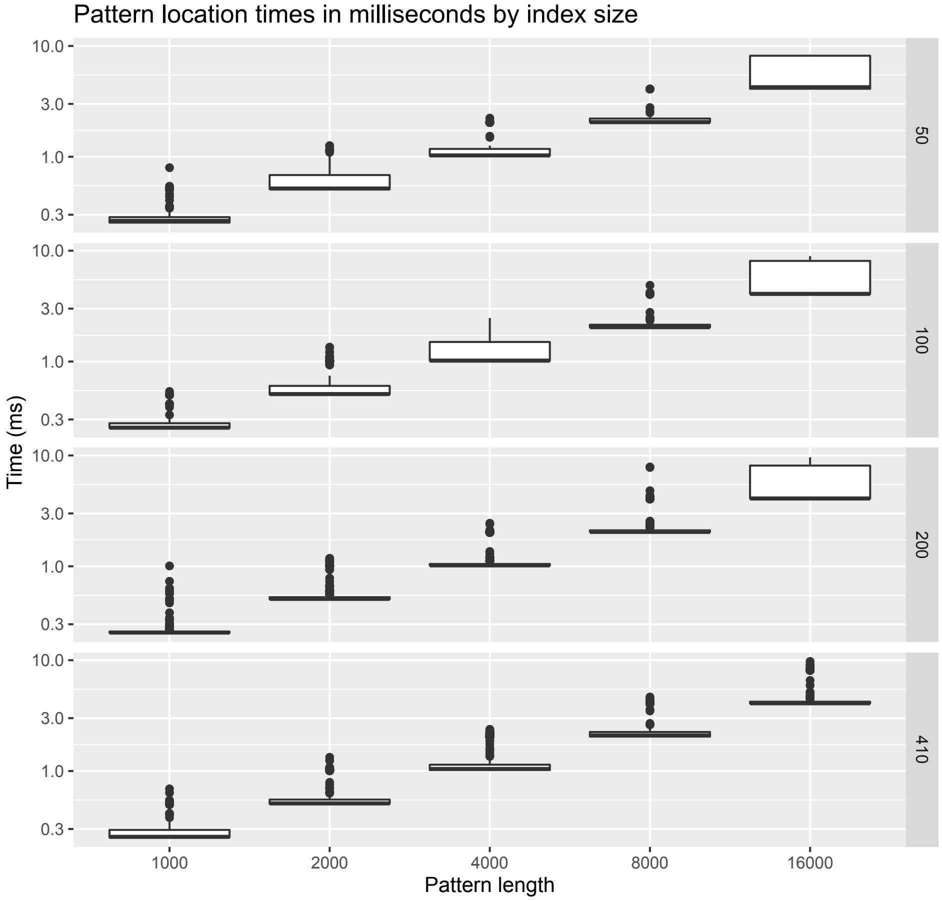


Figure 3 Running time of querying patterns from the founder block graph. The sample sizes (for MSA row subsets) are shown on the right-hand side of each plot. The plots show averages and distribution over 10 repeats of each search, where one search consist of a set of query patterns of given length randomly sampled from the respective MSA row subset. The pattern set sample size (10,20,30,40, respectively) grows by the MSA sample size, but the reported numbers are normalized so that the query time (milliseconds) is per pattern. This experiment was run on Intel(R) Core(TM) i5-4308U CPU, 2.80GHz.

The founder block graph concept could also be generalized so that it is not directly induced from a segmentation. One could consider cyclic graphs having the same segment repeat-free property. This could be an interesting direction in defining parameter-free de Bruijn graphs.

For journal version we plan to include a proper extension of the approach to general MSA's containing gaps. Our implementation already contains support for such MSA's, but the theory framework still requires some more work to show that such extension can be done without any effect on the running time.

On the experimental side, there is still much more work to be done. So far, we have not optimized any of the algorithms for multithreading nor for space usage. For example, one could use BWT indexes engineered for highly repetitive text collections to build the founder block graphs in space proportional of the compressed MSA. Such optimizations are essential for applying the approach on e.g. human genome data. Past experience on similar solutions [28] indicate that our approach should easily be applicable to much larger datasets than those we covered in our preliminary experiments.

Finally, this paper only scratches the surface of a new family of pangenome representations. There are myriad of options how to optimize among the valid segmentations [28, 12], e.g. by optimizing the number of founder segments [28] or controlling the over-expressiveness of the graph, rather than minimizing the maximum block size as studied here.

References

1 Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975.

2 Jarno Alanko, Giovanna D'Agostino, Alberto Policriti, and Nicola Prezza. Regular languages meet prefix sorting. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 911–930. SIAM, 2020.

3 Mai Alzamel, Lorraine A. K. Ayad, Giulia Bernardini, Roberto Grossi, Costas S. Iliopoulos, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Degenerate string comparison and applications. In Laxmi Parida and Esko Ukkonen, editors, *18th International Workshop on Algorithms in Bioinformatics, WABI 2018, August 20-22, 2018, Helsinki, Finland*, volume 113 of *LIPIcs*, pages 21:1–21:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

4 Djamal Belazzougui and Fabio Cunial. Fully-functional bidirectional burrows-wheeler indexes and infinite-order de bruijn graphs. In Nadia Pisanti and Solon P. Pissis, editors, *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019, June 18-20, 2019, Pisa, Italy*, volume 128 of *LIPIcs*, pages 10:1–10:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

5 Djamal Belazzougui, Fabio Cunial, and Olgert Denas. Fast matching statistics in small space. In Gianlorenzo D'Angelo, editor, *17th International Symposium on Experimental Algorithms, SEA 2018, June 27-29, 2018, L'Aquila, Italy*, volume 103 of *LIPIcs*, pages 17:1–17:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. `doi:10.4230/LIPIcs.SEA.2018.17`.

6 Djamal Belazzougui, Fabio Cunial, Juha Kärkkäinen, and Veli Mäkinen. Linear-time string indexing and analysis in small space. *ACM Trans. Algorithms*, 16(2), March 2020. `doi:10.1145/3381417`.

7 Djamal Belazzougui, Travis Gagie, Veli Mäkinen, Marco Previtali, and Simon J. Puglisi. Bidirectional variable-order de bruijn graphs. *Int. J. Found. Comput. Sci.*, 29(8):1279–1295, 2018. `doi:10.1142/S0129054118430037`.

8 Giulia Bernardini, Pawel Gawrychowski, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Even Faster Elastic-Degenerate String Matching via Fast Matrix Multiplication. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.ICALP.2019.21`.

9 Christina Boucher, Alexander Bowe, Travis Gagie, Simon J. Puglisi, and Kunihiko Sadakane. Variable-order de bruijn graphs. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2015 Data Compression Conference, DCC 2015, Snowbird, UT, USA, April 7-9, 2015*, pages 383–392. IEEE, 2015. `doi:10.1109/DCC.2015.70`.

10 Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de bruijn graphs. In Benjamin J. Raphael and Jijun Tang, editors, *Algorithms in Bioinformatics - 12th International Workshop, WABI 2012, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, volume 7534 of *Lecture Notes in Computer Science*, pages 225–235. Springer, 2012. `doi:10.1007/978-3-642-33122-0_18`.

11 M. Burrows and D. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.

12 Bastien Cazaux, Dmitry Kosolobov, Veli Mäkinen, and Tuukka Norri. Linear time maximum segmentation problems in column stream model. In Nieves R. Brisaboa and Simon J. Puglisi, editors, *String Processing and Information Retrieval - 26th International Symposium, SPIRE 2019, Segovia, Spain, October 7-9, 2019, Proceedings*, volume 11811 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 2019.

13 Computational Pan-Genomics Consortium et al. Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, page bbw089, 2016.

14 Rene De La Briandais. File searching using variable length keys. In *Papers Presented at the the March 3-5, 1959, Western Joint Computer Conference*, IRE-AIEE-ACM '59 (Western), page 295–298, New York, NY, USA, 1959. Association for Computing Machinery. doi:10.1145/1457838.1457895.

15 Massimo Equi, Roberto Grossi, Veli Mäkinen, and Alexandru I. Tomescu. On the complexity of string matching for graphs. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 55:1–55:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

16 Massimo Equi, Veli Mäkinen, and Alexandru I. Tomescu. Graphs cannot be indexed in polynomial time for sub-quadratic time string matching, unless seth fails, 2020. arXiv:2002.00629.

17 Harold N. Gabow, Jon Louis Bentley, and Robert Endre Tarjan. Scaling and related techniques for geometry problems. In Richard A. DeMillo, editor, *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1984, Washington, DC, USA*, pages 135–143. ACM, 1984. doi:10.1145/800057.808675.

18 Travis Gagie, Giovanni Manzini, and Jouni Sirén. Wheeler graphs: A framework for bwt-based data structures. *Theor. Comput. Sci.*, 698:67–78, 2017.

19 Erik Garrison, Jouni Sirén, Adam Novak, Glenn Hickey, Jordan Eizenga, Eric Dawson, William Jones, Shilpa Garg, Charles Markello, Michael Lin, and Benedict Paten. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature Biotechnology*, 36, August 2018. doi:10.1038/nbt.4227.

20 Daniel Gibney and Sharma V. Thankachan. On the hardness and inapproximability of recognizing wheeler graphs. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 51:1–51:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

21 Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In *13th International Symposium on Experimental Algorithms, (SEA 2014)*, pages 326–337, 2014. doi:10.1007/978-3-319-07959-2_28.

22 Lin Huang, Victoria Popic, and Serafim Batzoglou. Short read alignment with populations of genomes. *Bioinformatics*, 29(13):361–370, 2013.

23 G. Jacobson. Space-efficient static trees and graphs. In *Proc. FOCS*, pages 549–554, 1989.

24 Daehwan Kim, Joseph Paggi, Chanhee Park, Christopher Bennett, and Steven Salzberg. Graph-based genome alignment and genotyping with hisat2 and hisat-genotype. *Nature Biotechnology*, 37:1, August 2019. doi:10.1038/s41587-019-0201-4.

25 Sorina Maciuca, Carlos del Ojo Elias, Gil McVean, and Zamin Iqbal. A natural encoding of genetic variation in a Burrows-Wheeler transform to enable mapping and genome inference. In *Algorithms in Bioinformatics - 16th International Workshop, WABI 2016, Aarhus, Denmark, August 22-24, 2016. Proceedings*, volume 9838 of *Lecture Notes in Computer Science*, pages 222–233. Springer, 2016.

26 Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993. doi:10.1137/0222058.

27 Niema Moshiri. ViralMSA: Massively scalable reference-guided multiple sequence alignment of viral genomes. *bioRxiv*, 2020. doi:10.1101/2020.04.20.052068.

28 Tuukka Norri, Bastien Cazaux, Dmitry Kosolobov, and Veli Mäkinen. Linear time minimum segmentation enables scalable founder reconstruction. *Algorithms Mol. Biol.*, 14(1):12:1–12:15, 2019.

29 Pasi Rastas and Esko Ukkonen. Haplotype inference via hierarchical genotype parsing. In *Algorithms in Bioinformatics, 7th International Workshop, WABI 2007, Philadelphia, PA, USA, September 8-9, 2007, Proceedings*, pages 85–97, 2007.

30 Thomas Schnattinger, Enno Ohlebusch, and Simon Gog. Bidirectional search in a string with wavelet trees and bidirectional matching statistics. *Inf. Comput.*, 213:13–22, 2012.

31 Jouni Sirén, Niko Välimäki, and Veli Mäkinen. Indexing graphs for path queries with applications in genome research. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 11(2):375–388, 2014.

32 Jouni Sirén, Erik Garrison, Adam M Novak, Benedict Paten, and Richard Durbin. Haplotype-aware graph indexes. *Bioinformatics*, 36(2):400–407, July 2019. doi:10.1093/bioinformatics/btz575.

33 Chris Thachuk. Indexing hypertext. *Journal of Discrete Algorithms*, 18:113–122, 2012.

34 Esko Ukkonen. Finding founder sequences from a set of recombinants. In *Algorithms in Bioinformatics, Second International Workshop, WABI 2002, Rome, Italy, September 17-21, 2002, Proceedings*, pages 277–286, 2002.

GraphBin2: Refined and Overlapped Binning of Metagenomic Contigs Using Assembly Graphs

Vijini G. Mallawaarachchi
Research School of Computer Science, College of Engineering and Computer Science,
Australian National University, Canberra, Australia
vijini.mallawaarachchi@anu.edu.au

Anuradha S. Wickramarachchi
Research School of Computer Science, College of Engineering and Computer Science,
Australian National University, Canberra, Australia
anuradha.wickramarachchi@anu.edu.au

Yu Lin
Research School of Computer Science, College of Engineering and Computer Science,
Australian National University, Canberra, Australia
yu.lin@anu.edu.au

Abstract

Metagenomic sequencing allows us to study structure, diversity and ecology in microbial communities without the necessity of obtaining pure cultures. In many metagenomics studies, the reads obtained from metagenomics sequencing are first assembled into longer contigs and these contigs are then binned into clusters of contigs where contigs in a cluster are expected to come from the same species. As different species may share common sequences in their genomes, one assembled contig may belong to multiple species. However, existing tools for contig binning only support non-overlapped binning, *i.e.*, each contig is assigned to at most one bin (species). In this paper, we introduce GraphBin2 which refines the binning results obtained from existing tools and, more importantly, is able to assign contigs to multiple bins. GraphBin2 uses the connectivity and coverage information from assembly graphs to adjust existing binning results on contigs and to infer contigs shared by multiple species. Experimental results on both simulated and real datasets demonstrate that GraphBin2 not only improves binning results of existing tools but also supports to assign contigs to multiple bins.

2012 ACM Subject Classification Applied computing → Bioinformatics; Applied computing → Computational genomics

Keywords and phrases Metagenomics binning, contigs, assembly graphs, overlapped binning

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.8

Supplementary Material Our program can be found at `https://github.com/Vini2/GraphBin2`.

Acknowledgements We would like to thank the anonymous reviewers for their valuable comments. Furthermore, this research was undertaken with the assistance of resources and services from the National Computational Infrastructure (NCI Australia), an NCRIS enabled capability supported by the Australian Government.

1 Introduction

With the advent of high throughput sequencing approaches, the field of metagenomics has enabled us to access and study the genetic material of entire microbial communities [25, 32]. A microbial community is usually a complex mixture of multiple species and recovering these species is crucial to understand the behaviour and functions within such communities. To characterise the composition of a sample, we cluster metagenomic sequences into groups that represent different taxonomic groups such as species, genera or higher levels [28]. This process

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti; Article No. 8; pp. 8:1–8:21
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is known as *metagenomics binning*. Although it is possible to bin reads directly (before assembly) [1, 8, 10, 18, 23, 27, 33], reads are usually too short to enable accurate binning results [34]. Hence, a typical approach in metagenomics analysis starts from assembling short reads into longer contigs and then bin these resulting contigs into groups representing different taxonomic groups [28].

Existing contig-binning tools can be divided into two categories, (1) reference-based and (2) reference-free. Reference-based binning approaches [3, 15, 20, 37] rely on a database of reference genomes and thus may not be applicable in many metagenomic samples when reference genomes are not available. Reference-free binning tools use unsupervised approaches to group contigs into unlabelled bins which correspond to different taxonomic groups solely based on the information obtained from the contigs [28]. These reference-free binning methods become very useful when analysing environmental samples where many species are not found in the current reference databases [16]. Most of the reference-free tools make use of the composition and/or abundance (coverage) information of contigs to bin them [2, 12, 13, 14, 31, 36, 38]. Although contigs are assembled from reads using assembly graphs, most existing binning tools do not use the information of the assembly graph. More recently, GraphBin [19] has been developed to use the connectivity information in the assembly graph to refine the binning results of existing tools because contigs connected to each other in the assembly graph are more likely to belong to the same taxonomic group [5].

Different bacterial genomes in a metagenomic sample may share similar genes and genomic regions [26], which is a major challenge in assembling metagenomic reads into contigs [22]. Therefore, some assembled contigs from metagenomic reads may be shared by multiple species in the sample. However, very few contig-binning tools support overlapped binning (*i.e.*, assigning shared contigs to multiple species). S-GSOM [7] abstracts the flanking sequences of highly conserved 16S rRNA and incorporates them into Growing Self-Organising Maps (GSOM) to bin contigs into overlapping bins. MetaPhase [6] uses Hi-C reads to scaffold assembled contigs into assemblies of individual species and allows certain contigs to belong to multiple species. However, the applications of S-GSOM and MetaPhase are limited due to their required additional sequencing effort (*e.g.*, 16S RNA or Hi-C sequencing). As shared contigs correspond to shared vertices between different genomic paths on the assembly graph [22], it is worth investigating whether it is possible to infer such shared contigs from the assembly graph without additional sequencing requirements.

In this paper, we present GraphBin2, the new generation of GraphBin, to refine binning results using the assembly graph. While GraphBin only uses the topology information of the assembly graph, GraphBin2 improves the algorithms to adjust existing binning results and to support overlapped binning based on both the connectivity and coverage information of assembly graphs. Experimental results show that GraphBin2 not only improves existing binning results, but also infers contigs that may belong to multiple species.

2 Methods

Figure 1 denotes the workflow of GraphBin2. The preprocessing steps of GraphBin2 assemble reads into contigs using the assembly graph and then bin the contigs (*i.e.*, assign coloured labels to contigs) using existing contig-binning tools. GraphBin2 takes this labelled assembly graph as the input, removes unsupported labels, corrects the labels of inconsistent vertices, propagates labels to unlabelled vertices and finally infers vertices with multiple labels (colours).

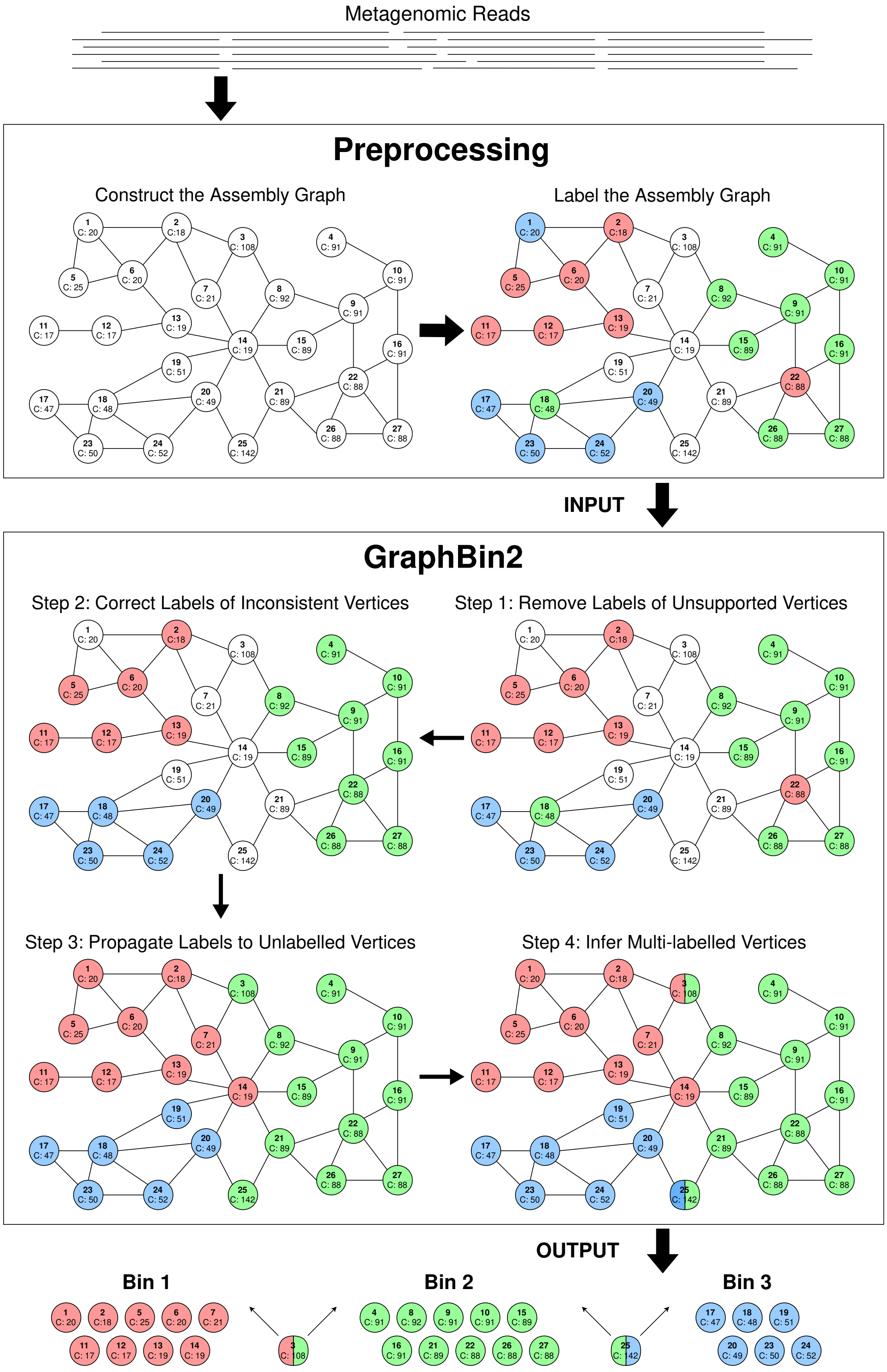


Figure 1 The workflow of GraphBin2.

2.1 Preprocessing

In this step, we assemble the next generation reads (*e.g.*, Illumina reads with length ranging from 75bp to 300bp) into contigs using the assembly graph. There are two dominant paradigms for genome assembly: overlap-layout-consensus (or string graphs) [21] and de Bruijn graphs [24]. We select two representative assemblers from each paradigm, SGA [30] and metaSPAdes [22] respectively, to demonstrate the adaptability of GraphBin2. In the assembly graph, each vertex represents a contig with *coverage* denoting the average number of reads that map to each base of the contig and each edge indicates a significant overlap between a pair of contigs. In an ideal case, a genome corresponds to a path in the assembly graph and its genomic sequence corresponds to the concatenation of contigs along this path. Hence, if two contigs are connected by an edge in the assembly graph, they are more likely to belong to the same genome. Previous studies [5, 19] have shown that the connectivity information between contigs can be used to refine binning results. In the assembly graph of metagenomic datasets, different genomes usually correspond to different paths in the assembly graph. If two genomes share a common contig (*e.g.*, unresolved "interspecies repeat" [22]), the corresponding vertex would be shared by two genomic paths in the assembly graph.

After assembling reads into contigs using assembly graphs, GraphBin2 uses an existing contig-binning tool to derive an initial binning result. Note that most of the existing tools for binning contigs require a minimum length for the contigs (*e.g.,* 1,000bp for MaxBin2 [38] and SolidBin [36], 500bp for BusyBee Web [16] and 1500bp for MetaBAT2 [13]). Therefore, many short contigs in the assembly graph will be discarded, resulting in low recall values as a common limitation of existing binning tools. For example, 65% of the contigs in the metaSPAdes assembly of the **Sharon-All** dataset were discarded by MaxBin2 due to their short length.

2.2 Step 1: Remove Labels of Unsupported Vertices

A linear (or circular) chromosome usually corresponds to a path (or a cycle) that traverses multiple vertices in the assembly graph. If two contigs belong to the same chromosome, they are likely to be connected by a path which consists of other contigs from the same chromosome. Therefore, a labelled vertex is defined as *supported* if and only if one of the following conditions hold.

- It is an isolated vertex
- It directly connects to a vertex of the same label
- It connects to a vertex of the same label through a path that consists of only unlabelled vertices

Otherwise, a labelled vertex is defined as *unsupported.* Note that the definition of *unsupported* vertices in GraphBin2 is more strict than *ambiguous* vertices in GraphBin.[1] For example, in the initial labelled assembly graph of Figure 1, vertex 2 in red is supported by vertex 6 in red as they are directly connected. Note that vertex 18 in green is also supported by vertex 15 in green as there exists a path (*i.e.*, 18 → 19 → 14 → 15) between them that traverses only unlabelled vertices (*i.e.*, 19 and 14). However, vertex 1 in blue is unsupported as it cannot reach another blue vertex through a path consisting of only unlabelled (white coloured) vertices.

[1] In GraphBin, a vertex i is denoted as an *ambiguous* vertex if at least one of its closest labelled vertices has a label that is different than the label of the vertex i.
An *ambiguous* vertex in GraphBin may be *supported* (in GraphBin2) by another vertex of the same label if they are directly connected or connected through a path consisting of only unlabelled vertices.
An *unsupported* vertex in GraphBin2 is always *ambiguous* in GraphBin.

To check whether a labelled vertex is *supported* or *unsupported*, a naive approach is to perform a breadth-first-search from each labelled vertex. A refined algorithm first initialises all labelled vertices as unsupported and scans the graph to identify all labelled vertices that are either isolated or directly connected to a vertex of the same label and classifies them as supported vertices. This refined algorithm then uses breadth-first-search to find all connected components that consist of only unlabelled vertices and for each component $Component$ stores a set of labelled vertices $N(Component)$ that are connected to vertices in $Component$. If multiple labelled vertices in $N(Component)$ have the same label, these vertices are supported because they connect to each other through a path that consists of only unlabelled vertices in $Component$. GraphBin2 removes the labels for all unsupported vertices because these labels may not be reliable. For example, the label of the unsupported vertex 1 is removed by GraphBin2 in Step 1 of Figure 1.

2.3 Step 2: Correct Labels of Inconsistent Vertices

After Step 1, each non-isolated labelled vertex v is supported by at least one vertex with the same label. The closer two vertices are in the assembly graph, the more likely they have the same label. For each vertex v, we introduce a *labelled score*, $S(v, x)$, for each label x by considering all vertices of label x that are directly connected to v or connected to v through a path that consists of only unlabelled vertices. A vertex t of label x contributes to $S(v, x)$ by $2^{-D(v,t)}$ where $D(v, t)$ is the shortest distance between v and t using only unlabelled vertices. This distance is measured by the number of edges in a path and $D(v, t) = 1$ if v and t are directly connected. Therefore, the *labelled score* $S(v, x)$ is the sum of contributions from all vertices of label x that are directly connected to v or connected to v through a path that consists of only unlabelled vertices. In Step 1 of Figure 1, vertex 17 contributes 1/2 to $S(18, blue)$ because $D(17, 18) = 1$ and vertex 8 contributes 1/8 to $S(18, green)$ because $D(8, 17) = 3$. The labelled score of $S(18, blue)$ is 2 to which all four blue vertices 17, 20, 23 and 24 contribute 1/2 respectively while $S(18, green) = 5/16$ to which vertex 8 contributes 1/8, vertex 15 contributes 1/8 and vertex 26 contributes 1/16.

A labelled vertex v of label x is defined as *inconsistent* if and only if the *labelled score* of its current label x times α is less than or equal to the *labelled score* of another label y where α is a parameter, *i.e.*, $\alpha \times S(v, x) \leqslant S(v, y)$. We have set $\alpha = 1.5$ in the default settings of GraphBin2. In Step 1 of Figure 1, vertex 18 in green is an inconsistent vertex because $1.5 \times S(18, green) = 1.5 \times 5/16 = 0.47$ is less than $S(18, blue) = 2$.

Again, GraphBin2 uses the breadth-first-search to check if a labelled vertex is *inconsistent*. GraphBin2 corrects the label of an inconsistent vertex v to another label that maximises the labelled score. For example, GraphBin2 corrects the label of vertex 18 from green to blue and corrects the label of vertex 22 from red to green (refer from Step 1 to Step 2 in Figure 1).

2.4 Step 3: Propagate Labels to Unlabelled Vertices

As existing contig-binning tools discard contigs due to their short lengths in the initial binning, many vertices are still unlabelled in the current assembly graph. In this step, we will propagate existing labels to the remaining unlabelled vertices using the assembly graph. There are two intuitions behind this label propagation process. Firstly, vertices that are closer to each other in the assembly graph are more likely to have the same label. Secondly, vertices with similar coverages are more likely to have the same label because contigs from the same genome usually have similar coverages [39, 12]. GraphBin2 uses both the connectivity and coverage information of the assembly graph to propagate the labels.

For each unlabelled vertex v with coverage $c(v)$ (*i.e.*, coverage of the contig that corresponds to the vertex), a candidate propagation action $(D(v,t), |c(v) - c(t)|, t, v)$ is recorded as a tuple where t is the nearest labelled vertex to v, $c(t)$ is the coverage of t and $D(v,t)$ is the shortest distance between v and t (as defined in Step 2). Given two candidate propagation actions, (d_1, c_1, t_1, v_1) and (d_2, c_2, t_2, v_2), GraphBin2 will execute (d_1, c_1, t_1, v_1) before (d_2, c_2, t_2, v_2), *i.e.,* propagating the label of t_1 to v_1 before propagating the label of t_2 to v_2, if $(d_1 < d_2)$ or $(c_1 < c_2$ and $d_1 = d_2)$. In other words, GraphBin2 puts more emphasis on the connectivity information than the coverage information because the edges in the assembly graph are expected to be more reliable than the coverage information on vertices, especially for vertices corresponding to short contigs (which are discarded by initial binning tools).

GraphBin2 first uses the breadth-first-search to compute all candidate propagation actions for unlabelled vertices and sort them into a ranked list according to the order defined above. At each iteration, GraphBin2 executes the first candidate propagation action and then updates the ranked list of candidate propagation actions. Note that one unlabelled vertex receives its label at each iteration and updating the ranked list of candidate propagation actions can be done efficiently by breadth-first-search from this unlabelled vertex. Please refer to the Supplementary Material Section A to see a step-by-step label propagation process from Step 2 to Step 3 in Figure 1. Note that this label propagation process in GraphBin2 improves on the label propagation algorithm in GraphBin by incorporating both the connectivity and coverage information in the assembly graph. So far, GraphBin2 does not generate multi-labelled vertices. In the next step, we will show how GraphBin2 uses the labelling, connectivity and coverage information together on the assembly graph to infer multi-labelled vertices.

2.5 Step 4: Infer Multi-Labelled Vertices

Contigs belonging to multiple genomes correspond to multi-labelled vertices in the assembly graph. What are the characteristics of shared contigs between multiple species? Firstly, a contig shared by multiple genomes may connect other contigs in these genomes. Secondly, the coverage of a contig shared by multiple genomes should be equal to the sum of coverages of these genomes in the ideal case. After label propagation, vertices of the same label are likely to form connected components in the assembly graph and multi-labelled vertices are likely to be located along the borders between multiple connected components where distinct labels meet and have a coverage similar to the sum of the average coverages of multiple components that they belong to.

GraphBin2 checks labelled vertices that are connected to vertices of multiple different labels. The average coverage of a connected component P is calculated by $\frac{\sum c(i) \times L(i)}{\sum L(i)}$ for each vertex i in the connected component P, where $c(i)$ is the coverage of the vertex i and $L(i)$ is the length of the contig corresponding to vertex i. Assume v is a labelled vertex v from a component P, the coverage of v is $c(v)$ and the average coverage of P is $c(P)$. When $c(v)$ is larger than $c(P)$ and v is connected to other components $P_1, P_2, \ldots, P_k$ with different labels, it is possible that v also belongs to one or more components (in addition to P). For example, if v belongs to P, P_i and P_j in the ground-truth, the coverage of v, $c(v)$, is expected to be close to the sum of average coverages of the above three components, $c(P) + c(P_i) + c(P_j)$. In fact, finding which components in $\{P_1, P_2, \ldots, P_k\}$ that v also belongs to (in addition to P) can be modelled as the following subset sum problem [9]. Given a set of positive numbers $\{c(P_1), c(P_2), \ldots, c(P_k)\}$, find a subset whose sum is or is closest to $c(v) - c(P)$. Then v will be assigned to the corresponding components in this subset as well as to P. Note that it is possible that the selected subset is empty and thus v only belongs to P.

In all of our experiments, the maximum number of different components that a vertex connects to in the assembly graph is less than 5. We use a brute-force way to enumerate all possible combinations of components and find out the combinations that best explain the observed coverages. For example, after Step 3 in Figure 1, vertex 3 in green connects to another red component. The coverage of vertex 3 is 108 while the average coverage of the green component is 95 and the average coverage of the red components is 19. Because the coverage of vertex 3 (108) is closer to the sum of average coverages of green and red components (95+19=114) compared to the average coverage of the green component (95), vertex 3 is assigned both green and red labels. Similarly, the coverage of vertex 25 (142) is closer to the sum of average coverages of green and blue components (95+49=144) compared to the average coverage of the green component (95). Hence, vertex 25 is assigned both green and blue labels. In the same assembly graph after Step 3 in Figure 1, vertex 14 in red does not gain any other labels because its own coverage is closest to the average coverage of the red component (19) compared to other possible combinations (*i.e.*, red+blue, red+green, green+blue and red+green+blue).

3 Experimental Setup

3.1 Datasets

3.1.1 Simulated Datasets

We simulated three metagenomic datasets according to the species found in the simMC+ dataset [38]. Three datasets were simulated each containing 5 species (referred as **Sim-5G**), 10 species (referred as **Sim-10G**) and 20 species (referred as **Sim-20G**) respectively. Paired-end reads were simulated using the tool InSilicoSeq [11] modelling a MiSeq instrument with 300bp mean read length. More details about the simulated datasets can be found in Table 1 and Supplementary Material Section B.

3.1.2 Real Datasets

We used the preborn infant gut metagenome, commonly known as the **Sharon** dataset [29] (NCBI accession number *SRA052203*). There are 18 Illumina (Illumina HiSeq 2000) runs available for this dataset. One run *SRR492184* is included as a representative dataset (referred as **Sharon-1**) and all the 18 Illumina runs are combined to form the **Sharon-All** dataset in our experiments. Further details can be found in Supplementary Material Section B.

3.2 Tools Used

To derive the assembly graph, there are two dominant assembly paradigms, de Bruijn graphs [24] and overlap-overlap-layout-consensus (or string graphs) [21]. We selected one representative tool from each paradigm to show the effectiveness of GraphBin2. To represent the de Bruijn graph paradigm, we used metaSPAdes[22] (from SPAdes version 3.13.0 [4]) with its default parameters to generate the assembly graph. As for the overlap-layout-consensus paradigm, we selected SGA (version 0.10.15) [30] to derive the assembly graph.

We used MaxBin2 (version 2.2.5) [38] with default parameters and SolidBin (version 1.3) [36] in `SolidBin-SFSmode` to obtain the initial binning results for our experiments. MaxBin2 and SolidBin are considered as hybrid contig-binning tools as they use both the composition and coverage information. They make use of tetranucleotide frequencies and

Table 1 Details about the simulated datasets.

Dataset	Species present	Genome size	Coverage	Abundance
Sim-5G	Acetobacter pasteurianus	2.9 Mb	115×	28%
	Aeromonas veronii	4.6 Mb	72×	28%
	Amycolatopsis mediterranei	10.4 Mb	26×	22%
	Arthrobacter arilaitensis	3.9 Mb	41×	13%
	Azorhizobium caulinodans	5.4 Mb	20×	9%
Sim-10G	Acetobacter pasteurianus	2.9 Mb	357×	25%
	Aeromonas veronii	4.6 Mb	225×	25%
	Amycolatopsis mediterranei	10.4 Mb	80×	20%
	Arthrobacter arilaitensis	3.9 Mb	128×	12%
	Azorhizobium caulinodans	5.4 Mb	62×	8%
	Bacillus cereus	5.3 Mb	58×	7%
	Bdellovibrio bacteriovorus	3.8 Mb	11×	1%
	Bifidobacterium adolescentis	2.1 Mb	20×	1%
	Brachyspira intermedia	3.4 Mb	11×	1%
	Campylobacter jejuni	1.7 Mb	21×	1%
Sim-20G	Acetobacter pasteurianus	2.9 Mb	705×	23%
	Aeromonas veronii	4.6 Mb	445×	23%
	Amycolatopsis mediterranei	10.4 Mb	157×	18%
	Arthrobacter arilaitensis	3.9 Mb	253×	11%
	Azorhizobium caulinodans	5.4 Mb	123×	7%
	Bacillus cereus	5.3 Mb	114×	7%
	Bdellovibrio bacteriovorus	3.8 Mb	22×	1%
	Bifidobacterium adolescentis	2.1 Mb	40×	1%
	Brachyspira intermedia	3.4 Mb	21×	1%
	Campylobacter jejuni	1.7 Mb	41×	1%
	Candidatus Pelagibacter ubique	1.3 Mb	54×	1%
	Chlamydia trachomatis	1.1 Mb	64×	1%
	Clostridium acetobutylicum	4.0 Mb	18×	1%
	Corynebacterium diphtheriae	2.5 Mb	28×	1%
	Cyanobacterium UCYN	1.5 Mb	47×	1%
	Desulfovibrio vulgaris	3.6 Mb	20×	1%
	Ehrlichia ruminantium	1.5 Mb	47×	1%
	Enterococcus faecium	3.0 Mb	24×	1%
	Erysipelothrix rhusiopathiae	1.8 Mb	39×	1%
	Escherichia coli	5.0 Mb	14×	1%

coverages of reads with different machine learning approaches to bin contigs. Note that both MaxBin2 and SolidBin only bin contigs which are longer than 1,000bp by default. We also compared GraphBin2 with its predecessor GraphBin [19]. The commands used to run all the assembly and binning tools can be found in Supplementary Material Section C.

3.3 Evaluation Criteria

Since the reference genomes of the simulated datasets were known, we used BWA-MEM [17] to align the contigs to their reference genomes to determine the ground truth species to which the contigs actually belonged to. If at least 50% of a contig aligns to a species, then a contig is considered to belong to this species. Note that a contig may be considered to belong to multiple species if multiple such alignments exist. To reduce the effect of random alignments between short contigs and multiple genomes, a contig is considered to belong to multiple species when its length is at least 1,000bp long. Furthermore, isolated contigs (corresponding vertices with zero degree in the assembly graph) were not considered for the ground-truth set of the datasets.

For the Sharon dataset, we considered the annotated contigs from 12 species which are available at `https://ggkbase.berkeley.edu/carrol/organisms` as references. A process similar to the simulated datasets were followed for the Sharon datasets to determine the origin species of contigs and contigs belonging to multiple species.

To evaluate the binning results of MaxBin2 [38], SolidBin [36], GraphBin [19] and GraphBin2, we used the metrics (1) precision, (2) recall and (3) F1-score which have been used in previous studies [2, 19, 35]. The binning result is denoted as a $K \times S$ matrix where K is the number of bins identified by the binning tool and S is the number of species available in the ground truth. In this matrix, the element a_{ks} denotes the number of contigs binned to the k^{th} bin and belongs to the s^{th} species. Note that contigs belonging to multiple species are not included in this matrix. $Unclassified$ denotes the number of contigs that are unclassified or discarded by the tool. Following are the definitions and equations that were used to calculate the precision, recall and F1-score.

$$Precision = \frac{\sum_k max_s\{a_{ks}\}}{\sum_k \sum_s a_{ks}} \tag{1}$$

$$Recall = \frac{\sum_s max_k\{a_{ks}\}}{(\sum_k \sum_s a_{ks} + Unclassified)} \tag{2}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3}$$

To evaluate the detection of multi-labelled vertices corresponding to contigs that may belong to multiple species, we used the criteria (1) sensitivity (also known as true positive rate or recall) which measures the proportion of actual positives that are correctly identified, (2) specificity (also known as true negative rate) which measures the proportion of actual negatives that are correctly identified and (3) balanced accuracy as follows.

$$Sensitivity = \frac{TP}{TP + FN} \tag{4}$$

$$Specificity = \frac{TN}{TN + FP} \tag{5}$$

$$Balanced\ accuracy = \frac{Sensitivity + Specificity}{2} \tag{6}$$

Here TP refers to the true positives (*i.e.*, the number of multi-labelled vertices correctly assigned with multiple labels), FP refers to the false positives (*i.e.*, the number of single-labelled vertices incorrectly assigned with multiple labels), FN refers to the false negatives (*i.e.*, the number of multi-labelled vertices incorrectly assigned with a single label) and TN refers to the true negatives (*i.e.*, the number of single labelled vertices correctly assigned with a single label). We use the balanced accuracy because the dataset is imbalanced; *i.e.*, the number of multi-labelled contigs is much smaller than the number of single-labelled contigs.

4 Results and Discussion

4.1 Binning Results

Table 2 Comparison of binning results of MaxBin2 [38], GraphBin [19] and GraphBin2 (on top of MaxBin2 results) using assembly graphs built by metaSPAdes [22]. The best values are highlighted in bold.

Dataset	No. of bins identified	Evaluation Criteria	**MaxBin2**	**GraphBin** with **MaxBin2** results	**GraphBin2** with **MaxBin2** results
Sim-5G	5	Precision	92.28%	**99.80%**	99.03%
		Recall	44.16%	97.08%	**99.03%**
		F1 score	59.74%	98.42%	**99.03%**
Sim-10G	10	Precision	90.24%	99.77%	**99.78%**
		Recall	38.21%	98.66%	**99.78%**
		F1 score	53.69%	99.21%	**99.78%**
Sim-20G	21	Precision	89.48%	**98.28%**	97.78%
		Recall	41.37%	94.06%	**97.71%**
		F1 score	56.59%	96.12%	**97.74%**
Sharon-1	5	Precision	75.46%	89.28%	**90.02%**
		Recall	31.59%	61.44%	**62.53%**
		F1 score	44.54%	72.79%	**73.80%**
Sharon-All	11	Precision	83.80%	90.02%	**90.09%**
		Recall	28.55%	82.04%	**83.25%**
		F1 score	42.58%	85.84%	**86.53%**

Table 3 Comparison of binning results of SolidBin [36], GraphBin [19] and GraphBin2 (on top of SolidBin results) using assembly graphs built by metaSPAdes [22]. The best values are highlighted in bold.

Dataset	No. of bins identified	Evaluation Criteria	**SolidBin**	**GraphBin** with **SolidBin** results	**GraphBin2** with **SolidBin** results
Sim-5G	5	Precision	91.94%	**99.40%**	99.03%
		Recall	44.36%	96.50%	**99.03%**
		F1 score	59.84%	97.93%	**99.03%**
Sim-10G	10	Precision	92.17%	**99.21%**	98.77%
		Recall	39.44%	98.99%	**99.55%**
		F1 score	55.24%	99.10%	**99.16%**
Sim-20G	10	Precision	17.51%	35.30%	**46.05%**
		Recall	8.80%	89.62%	**90.05%**
		F1 score	11.72%	50.65%	**60.94%**
Sharon-1	5	Precision	72.31%	83.98%	**86.93%**
		Recall	30.08%	86.99%	**92.95%**
		F1 score	42.49%	85.46%	**89.84%**
Sharon-All	9	Precision	78.30%	**82.98%**	81.13%
		Recall	22.63%	66.75%	**68.30%**
		F1 score	35.11%	73.99%	**74.16%**

Table 4 Comparison of binning results of MaxBin2 [38], GraphBin [19] and GraphBin2 (on top of MaxBin2 results) using assembly graphs built by SGA [30]. The best values are highlighted in bold.

Dataset	No. of bins identified	Evaluation Criteria	**MaxBin2**	**GraphBin** with **MaxBin2** results	**GraphBin2** with **MaxBin2** results
Sim-5G	5	Precision	93.01%	99.45%	**99.57%**
		Recall	2.70%	99.35%	**99.57%**
		F1 score	5.26%	99.40%	**99.57%**
Sim-10G	9	Precision	97.12%	93.03%	**98.08%**
		Recall	5.21%	89.73%	**94.70%**
		F1 score	9.89%	91.35%	**96.36%**
Sim-20G	20	Precision	**96.30%**	87.66%	94.39%
		Recall	4.03%	85.91%	**92.69%**
		F1 score	7.74%	86.78%	**93.53%**
Sharon-1	5	Precision	91.29%	85.90%	**93.48%**
		Recall	32.90%	76.67%	**80.47%**
		F1 score	48.36%	81.02%	**86.49%**
Sharon-All	8	Precision	63.32%	77.83%	**78.07%**
		Recall	15.85%	37.62%	**39.58%**
		F1 score	25.35%	50.73%	**52.53%**

Table 5 Comparison of binning results of SolidBin [36], GraphBin [19] and GraphBin2 (on top of SolidBin results) using assembly graphs built by SGA [30]. The best values are highlighted in bold.

Dataset	No. of bins identified	Evaluation Criteria	**SolidBin**	**GraphBin** with **SolidBin** results	**GraphBin2** with **SolidBin** results
Sim-5G	5	Precision	93.37%	99.29%	**99.62%**
		Recall	2.71%	99.29%	**99.54%**
		F1 score	5.27%	99.29%	**99.58%**
Sim-10G	9	Precision	85.20%	77.82%	**88.91%**
		Recall	5.05%	75.38%	**93.62%**
		F1 score	9.54%	76.58%	**91.20%**
Sim-20G	19	Precision	**86.25%**	77.07%	83.28%
		Recall	3.86%	64.92%	**77.31%**
		F1 score	7.39%	70.10%	**80.18%**
Sharon-1	4	Precision	94.79%	**97.19%**	96.53%
		Recall	35.78%	90.83%	**91.59%**
		F1 score	51.95%	93.90%	**93.99%**
Sharon-All	5	Precision	60.85%	**76.02%**	75.90%
		Recall	22.33%	47.48%	**47.84%**
		F1 score	32.67%	58.45%	**58.69%**

Table 2 and Table 4 denote the binning results of MaxBin2 [38] and the binning results of GraphBin [19] and GraphBin2 on top of MaxBin2 results for the metaSPAdes [22] assemblies and SGA [30] assemblies, respectively. Table 3 and Table 5 demonstrate the results of SolidBin [36], GraphBin [19] and GraphBin2 on top of SolidBin results for metaSPAdes assemblies and SGA assemblies, respectively. The results in these tables show that GraphBin2 achieves the best performance in most of the scenarios. Both GraphBin and GraphBin2 have shown significant improvements on recall compared to MaxBin2 and SolidBin. While MaxBin2 and SolidBin filter contigs with length shorter than 1,000bp, GraphBin and GraphBin2 are able to bin short contigs using assembly graphs built by either metaSPAdes or SGA. In a few scenarios, GraphBin2 improved on the recall with a bit of a compromise on the precision compared to the GraphBin because GraphBin removes ambiguous labels in the final step. Furthermore, the existence of weak edges (*i.e.*, edges that are not well supported from the data) can form false connections between contigs and can mislead the label propagation process.

4.2 Multi-Labelled Inference Results

One key novelty of GraphBin2 is the introduction of the multiple-labelled inference for contigs. Tables 6, 7, 8 and 9 demonstrate the performance of the GraphBin2 with its multi-labelled inference. It is evident that there is an increase in the number of multi-labelled contigs with the increasing complexity of the dataset.

Table 6 Multi-labelled inference results using GraphBin2 on top of MaxBin2 [38] results for the metaSPAdes assemblies.

Dataset	Ground truth	TP	FP	TN	FN	Sensitivity	Specificity	Balanced accuracy
Sim-5G	2	2	2	512	0	100.00%	99.61%	99.81%
Sim-10G	5	4	3	893	1	80.00%	99.67%	89.83%
Sim-20G	7	4	7	1,393	3	57.14%	99.50%	78.32%
Sharon-1	2	2	1	368	0	100.00%	99.73%	99.86%
Sharon-All	8	4	34	2,692	4	50.00%	98.75%	74.38%

Table 7 Multi-labelled inference results using GraphBin2 on top of SolidBin [36] results for the metaSPAdes assemblies.

Dataset	Ground truth	TP	FP	TN	FN	Sensitivity	Specificity	Balanced accuracy
Sim-5G	2	2	3	511	0	100.00%	99.42%	99.71%
Sim-10G	5	4	3	893	1	80.00%	99.67%	89.83%
Sim-20G	7	4	7	1,393	3	57.14%	99.50%	78.32%
Sharon-1	2	1	1	369	1	50.00%	99.73%	74.86%
Sharon-All	8	1	29	2,700	7	12.50%	98.94%	55.72%

GraphBin2 has assigned correct labels for most of the multi-labelled and single-labelled vertices (*i.e.*, TP+TN). The relatively poor true-positive rate on **Sharon-All** dataset may be due to the poor performance of the initial binning results of MaxBin2 and SolidBin. Moreover, the sequencing noise or contamination in the real metagenomic dataset may also affect the identification of multi-labelled vertices in the assembly graph. Furthermore, the **Sharon-All** dataset consisted of short reads of 100bp length compared to other datasets and resulted in a very fragmented assembly graph with a large number of nodes and edges.

Table 8 Multi-labelled inference results using GraphBin2 on top of MaxBin2 [38] results for the SGA assemblies.

Dataset	Ground truth	TP	FP	TN	FN	Sensitivity	Specificity	Balanced accuracy
Sim-5G	3	1	5	18,186	2	33.33%	99.97%	66.65%
Sim-10G	2	1	8	32,380	1	50.00%	99.98%	74.99%
Sim-20G	3	1	14	72,776	2	33.33%	99.98%	66.66%
Sharon-1	3	1	1	764	2	33.33%	99.87%	66.60%
Sharon-All	9	1	36	20,905	8	11.11%	99.83%	55.47%

Table 9 Multi-labelled inference results using GraphBin2 on top of SolidBin [36] results for the SGA assemblies.

Dataset	Ground truth	TP	FP	TN	FN	Sensitivity	Specificity	Balanced accuracy
Sim-5G	3	2	6	18,184	1	66.67%	99.97%	83.32%
Sim-10G	2	1	0	32,388	1	50.00%	100.00%	75.00%
Sim-20G	3	1	10	72,780	2	33.33%	99.99%	66.66%
Sharon-1	3	1	0	765	2	33.33%	100.00%	66.67%
Sharon-All	9	2	15	20,925	7	22.22%	99.93%	61.08%

4.3 Visualisation of the Assembly Graph

Figure 2 denotes the labelling of the contigs in the metaSPAdes assembly graph of the **Sim-5G** dataset at different stages as it undergoes the processing of GraphBin2. In Figure 2(a), we can see that some mis-binned contigs are identified (circled in red) as differently coloured contigs within components of a single colour. Figure 2(b) shows the refined assembly graph where GraphBin2 has removed labels of unsupported vertices and corrected labels of inconsistent vertices. After GraphBin2 propagates labels to the remaining unlabelled vertices, the assembly graph will be as denoted in Figure 2(c). Finally, GraphBin2 will detect multi-labelled vertices that correspond to contigs that may belong to multiple species as shown by the black coloured vertices in Figure 2(d).

4.4 Implementation

The source code for the experiments was implemented using Python 3.7.3 and run on a Darwin system with macOS Mojave 10.14.6, 16G memory and Intel Core i7 CPU @ 2.8 GHz with 4 CPU cores. In our experiments, we restrict the depth of the breadth-first-search in Steps 2-3 to be 5 to speed up GraphBin2. Moreover, we have set the parameter $\alpha = 1.5$ by default for GraphBin2. Furthermore, the process of inferring multi-labelled vertices was performed in parallel using multithreading (set to 8 threads by default in GraphBin2).

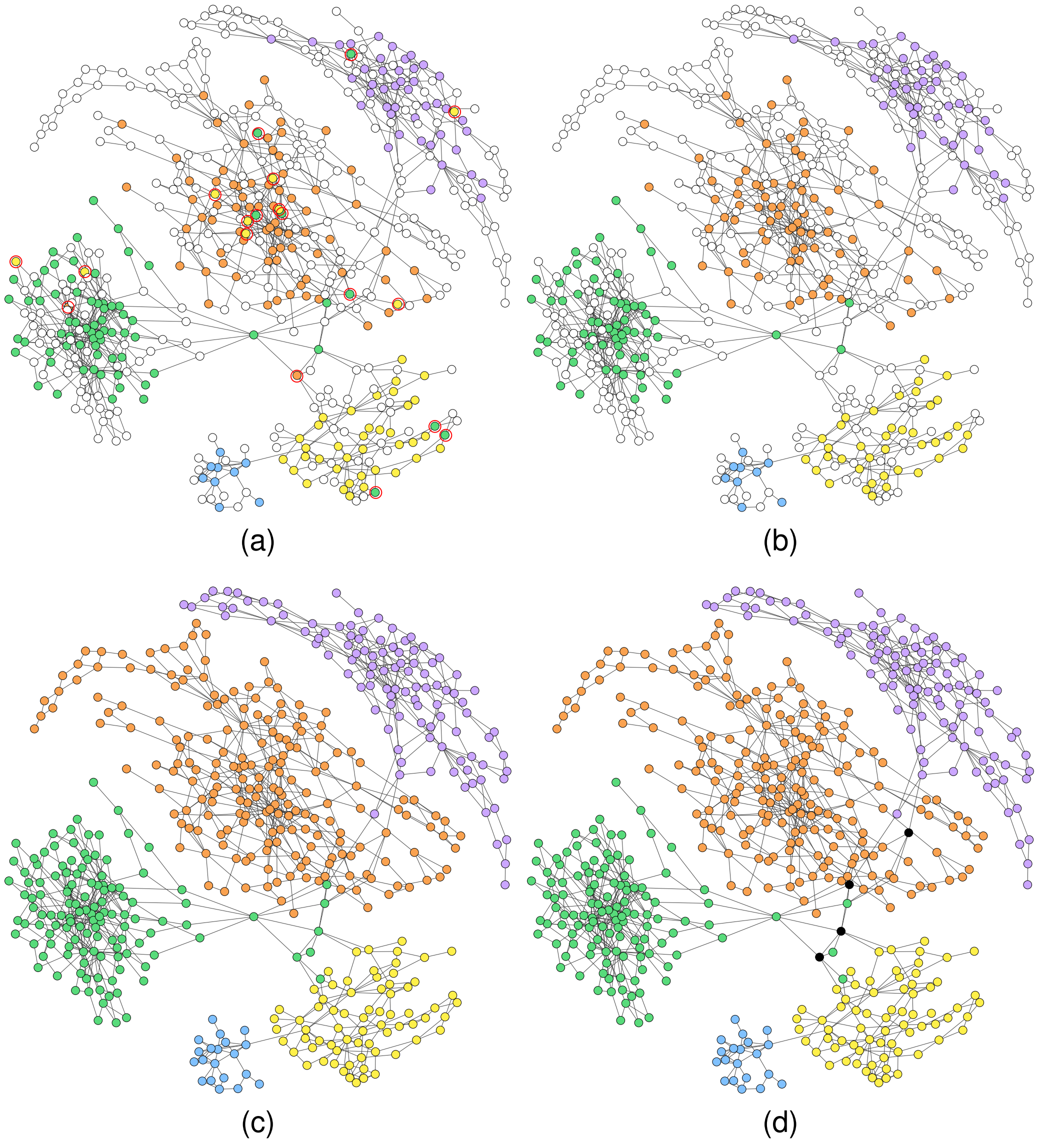


Figure 2 The labelling of the assembly graph of Sim-5G dataset based on (a) the initial MaxBin2 result (mis-binned contigs are circled in red), (b) after removing labels of unsupported vertices and correcting labels of inconsistent vertices, (c) after propagating labels of unlabelled vertices (d) after determining multi-labelled vertices (black coloured vertices) by GraphBin2.

4.5 Running Time and Memory Usage

Table 10 denotes the running times (wall time) and the peak memory used for the **Sharon-1** and **Sharon-All** datasets. MaxBin2 and GraphBin2 executed with 8 threads and SolidBin executed with a single thread. The running times for MaxBin2 and SolidBin only include the times taken to run the main software, excluding the times taken to build the composition and coverage profile files.

GraphBin2 took less than 12 minutes and less than 165 MB of memory to complete executing the **Sharon-All** dataset with 8 threads.

Table 10 Running times (wall time) and peak memory usage for binning using each tool. *s* denotes seconds, *m* denotes minutes and *MB* denotes megabytes.

Dataset	Assembly type	Criteria	MaxBin2	GraphBin2 with MaxBin2 result	SolidBin	GraphBin2 with SolidBin result
Sharon-1	metaSPAdes	Time	9s	5s	6s	5s
		Memory	1,389 MB	45 MB	290 MB	45 MB
	SGA	Time	12s	3s	15s	3s
		Memory	203 MB	33 MB	654 MB	33 MB
Sharon-All	metaSPAdes	Time	30s	10m 50s	2m 7s	11m 12s
		Memory	1,378 MB	163 MB	1,416 MB	163 MB
	SGA	Time	28s	1m 21s	2m 51s	1m 15s
		Memory	241 MB	50 MB	2,612 MB	50 MB

5 Conclusion

In this paper we presented a novel algorithm, GraphBin2, that incorporates the coverage information into the assembly graph as an improvement of GraphBin [19]. While GraphBin uses only the topology of the assembly graph to remove and propagate labels, GraphBin2 makes use of the coverage information on vertices to perform label propagation. Furthermore, GraphBin2 enables the detection of contigs that may belong to multiple species. The performance of GraphBin2 was evaluated against its predecessor and two other binning tools on top of contigs obtained from short-reads assembled using metaSPAdes [22] and SGA [30] which represent the two assembly paradigms; de Bruijn graphs and overlap-layout-consensus (string graphs). The results showed that GraphBin2 achieves the best binning performance in both simulated and real datasets. Moreover, GraphBin2 shows the potential to infer contigs shared by multiple species. Note that GraphBin2 could be in principle applied to long-read assemblies. In the future, we intend to extend the capabilities of GraphBin2 to explore the avenues at improving the detection of contigs shared by multiple species and further extend towards binning long reads directly using read-overlap graphs.

References

1 Jarno Alanko, Fabio Cunial, Djamal Belazzougui, and Veli Mäkinen. A framework for space-efficient read clustering in metagenomic samples. *BMC Bioinformatics*, 18(3):59, March 2017. doi:10.1186/s12859-017-1466-6.

2 Johannes Alneberg, Brynjar Smári Bjarnason, Ino de Bruijn, Melanie Schirmer, Joshua Quick, Umer Z. Ijaz, Leo Lahti, Nicholas J. Loman, Anders F. Andersson, and Christopher Quince. Binning metagenomic contigs by coverage and composition. *Nature Methods*, 11:1144–1146, September 2014. doi:10.1038/nmeth.3103.

3 Sasha K. Ames, David A. Hysom, Shea N. Gardner, et al. Scalable metagenomic taxonomy classification using a reference genome database. *Bioinformatics*, 29(18):2253–2260, July 2013. arXiv:http://oup.prod.sis.lan/bioinformatics/article-pdf/29/18/2253/17128159/btt389.pdf.

4 Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A. Gurevich, Mikhail Dvorkin, Alexander S. Kulikov, Valery M. Lesin, Sergey I. Nikolenko, Son Pham, Andrey D. Prjibelski, Alexey V. Pyshkin, Alexander V. Sirotkin, Nikolay Vyahhi, Glenn Tesler, Max A. Alekseyev,

and Pavel A. Pevzner. SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing. *Journal of Computational Biology*, 19(5):455–477, 2012. PMID: 22506599. doi:10.1089/cmb.2012.0021.

5 Tyler P. Barnum, Israel A. Figueroa, Charlotte I. Carlström, Lauren N. Lucas, Anna L. Engelbrektson, and John D. Coates. Genome-resolved metagenomics identifies genetic mobility, metabolic interactions, and unexpected diversity in perchlorate-reducing communities. *The ISME Journal*, 12(6):1568–1581, 2018. doi:10.1038/s41396-018-0081-5.

6 Joshua N. Burton, Ivan Liachko, Maitreya J. Dunham, and Jay Shendure. Species-level deconvolution of metagenome assemblies with hi-c–based contact probability maps. *G3: Genes, Genomes, Genetics*, 4(7):1339–1346, 2014. doi:10.1534/g3.114.011825.

7 Chon-Kit Kenneth Chan, Arthur L. Hsu, Saman K. Halgamuge, and Sen-Lin Tang. Binning sequences using very sparse labels within a metagenome. *BMC Bioinformatics*, 9(1):215, April 2008. doi:10.1186/1471-2105-9-215.

8 Brian Cleary, Ilana Lauren Brito, Katherine Huang, Dirk Gevers, Terrance Shea, Sarah Young, and Eric J. Alm. Detection of low-abundance bacterial strains in metagenomic datasets by eigengenome partitioning. *Nature Biotechnology*, 33:1053, September 2015. doi:10.1038/nbt.3329.

9 Garey, Michael R. and Johnson, David S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., USA, 1979.

10 Samuele Girotto, Cinzia Pizzi, and Matteo Comin. MetaProb: accurate metagenomic reads binning based on probabilistic sequence signatures. *Bioinformatics*, 32(17):i567–i575, August 2016. doi:10.1093/bioinformatics/btw466.

11 Hadrien Gourlé, Oskar Karlsson-Lindsjö, Juliette Hayer, and Erik Bongcam-Rudloff. Simulating Illumina metagenomic data with InSilicoSeq. *Bioinformatics*, 35(3):521–522, July 2018. doi:10.1093/bioinformatics/bty630.

12 Damayanthi Herath, Sen-Lin Tang, Kshitij Tandon, David Ackland, and Saman Kumara Halgamuge. Comet: a workflow using contig coverage and composition for binning a metagenomic sample with high precision. *BMC Bioinformatics*, 18(16):571, December 2017. doi:10.1186/s12859-017-1967-3.

13 Dongwan Kang, Feng Li, Edward S Kirton, Ashleigh Thomas, Rob S Egan, Hong An, and Zhong Wang. MetaBAT 2: an adaptive binning algorithm for robust and efficient genome reconstruction from metagenome assemblies. *PeerJ*, 7:e27522v1, February 2019. doi:10.7287/peerj.preprints.27522v1.

14 David Kelley and Steven Salzberg. Clustering metagenomic sequences with interpolated Markov models. *BMC Bioinformatics*, 11(1):544, 2010. doi:10.1186/1471-2105-11-544.

15 Daehwan Kim, Li Song, Florian P. Breitwieser, and Steven L. Salzberg. Centrifuge: rapid and sensitive classification of metagenomic sequences. *Genome Research*, 26(12):1721–1729, 2016. arXiv:http://genome.cshlp.org/content/26/12/1721.full.pdf+html.

16 Cedric C. Laczny, Christina Kiefer, Valentina Galata, Tobias Fehlmann, Christina Backes, and Andreas Keller. BusyBee Web: metagenomic data analysis by bootstrapped supervised binning and annotation. *Nucleic Acids Research*, 45(W1):W171–W179, May 2017. doi:10.1093/nar/gkx348.

17 Heng Li. Aligning sequence reads, clone sequences and assembly contigs with bwa-mem, 2013. arXiv:1303.3997.

18 Yunan Luo, Yun William Yu, Jianyang Zeng, Bonnie Berger, and Jian Peng. Metagenomic binning through low-density hashing. *Bioinformatics*, 35(2):219–226, July 2018. doi:10.1093/bioinformatics/bty611.

19 Vijini Mallawaarachchi, Anuradha Wickramarachchi, and Yu Lin. GraphBin: Refined binning of metagenomic contigs using assembly graphs. *Bioinformatics*, March 2020. btaa180. doi:10.1093/bioinformatics/btaa180.

20 Peter Menzel, Kim Lee Ng, and Anders Krogh. Fast and sensitive taxonomic classification for metagenomics with Kaiju. *Nature Communications*, 7:11257, April 2016. Article.

21 Eugene W. Myers. The fragment assembly string graph. *Bioinformatics*, 21(suppl_2):ii79–ii85, September 2005. doi:10.1093/bioinformatics/bti1114.

22 Sergey Nurk, Dmitry Meleshko, Anton Korobeynikov, and Pavel A. Pevzner. metaSPAdes: a new versatile metagenomic assembler. *Genome Research*, 27(5):824–834, 2017. doi:10.1101/gr.213959.116.

23 Rachid Ounit, Steve Wanamaker, Timothy J. Close, and Stefano Lonardi. CLARK: fast and accurate classification of metagenomic and genomic sequences using discriminative k-mers. *BMC Genomics*, 16(1):236, March 2015. doi:10.1186/s12864-015-1419-2.

24 Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001. doi:10.1073/pnas.171285098.

25 Christopher Quince, Alan W. Walker, Jared T. Simpson, Nicholas J. Loman, and Nicola Segata. Shotgun metagenomics, from sampling to analysis. *Nature Biotechnology*, 35(9):833–844, 2017. doi:10.1038/nbt.3935.

26 Christian S. Riesenfeld, Patrick D. Schloss, and Jo Handelsman. Metagenomics: Genomic analysis of microbial communities. *Annual Review of Genetics*, 38(1):525–552, 2004. PMID: 15568985. doi:10.1146/annurev.genet.38.072902.091216.

27 L Schaeffer, H Pimentel, N Bray, P Melsted, and L Pachter. Pseudoalignment for metagenomic read assignment. *Bioinformatics*, 33(14):2082–2088, February 2017. doi:10.1093/bioinformatics/btx106.

28 Karel Sedlar, Kristyna Kupkova, and Ivo Provaznik. Bioinformatics strategies for taxonomy independent binning and visualization of sequences in shotgun metagenomics. *Computational and Structural Biotechnology Journal*, 15:48–55, 2017. doi:10.1016/j.csbj.2016.11.005.

29 Itai Sharon, Michael J. Morowitz, Brian C. Thomas, Elizabeth K. Costello, David A. Relman, and Jillian F. Banfield. Time series community genomics analysis reveals rapid shifts in bacterial species, strains, and phage during infant gut colonization. *Genome Research*, 23(1):111–120, 2013. doi:10.1101/gr.142315.112.

30 Jared T. Simpson and Richard Durbin. Efficient de novo assembly of large genomes using compressed data structures. *Genome Research*, 22(3):549–556, 2012. doi:10.1101/gr.126953.111.

31 Marc Strous, Beate Kraft, Regina Bisdorf, and Halina Tegetmeyer. The Binning of Metagenomic Contigs for Microbial Physiology of Mixed Cultures. *Frontiers in Microbiology*, 3:410, 2012. doi:10.3389/fmicb.2012.00410.

32 Torsten Thomas, Jack Gilbert, and Folker Meyer. Metagenomics - a guide from sampling to data analysis. *Microbial Informatics and Experimentation*, 2(1):3, 2012. doi:10.1186/2042-5783-2-3.

33 Le Van Vinh, Tran Van Lang, Le Thanh Binh, and Tran Van Hoai. A two-phase binning algorithm using l-mer frequency on groups of non-overlapping reads. *Algorithms for Molecular Biology*, 10(1):2, January 2015. doi:10.1186/s13015-014-0030-4.

34 Jun Wang, Yuan Jiang, Guoxian Yu, Hao Zhang, and Haiwei Luo. BMC3C: binning metagenomic contigs using codon usage, sequence composition and read coverage. *Bioinformatics*, 34(24):4172–4179, June 2018. doi:10.1093/bioinformatics/bty519.

35 Ying Wang, Kun Wang, Yang Young Lu, and Fengzhu Sun. Improving contig binning of metagenomic data using d2S oligonucleotide frequency dissimilarity. *BMC Bioinformatics*, 18(1):425, September 2017. doi:10.1186/s12859-017-1835-1.

36 Ziye Wang, Zhengyang Wang, Yang Young Lu, Fengzhu Sun, and Shanfeng Zhu. SolidBin: improving metagenome binning with semi-supervised normalized cut. *Bioinformatics*, 35(21):4229–4238, April 2019. doi:10.1093/bioinformatics/btz253.

37 Derrick E. Wood and Steven L. Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biology*, 15(3):R46, 2014.

38 Yu-Wei Wu, Blake A. Simmons, and Steven W. Singer. MaxBin 2.0: an automated binning algorithm to recover genomes from multiple metagenomic datasets. *Bioinformatics*, 32(4):605–607, October 2015. doi:10.1093/bioinformatics/btv638.

39 Yu-Wei Wu, Yung-Hsu Tang, Susannah G. Tringe, Blake A. Simmons, and Steven W. Singer. Maxbin: an automated binning method to recover individual genomes from metagenomes using an expectation-maximization algorithm. *Microbiome*, 2(1):26, August 2014. doi:10.1186/2049-2618-2-26.

A Step-By-Step Example of Label Propagation in GraphBin2

Figure 3 shows how GraphBin2 propagates labels from Step 2 to Step 3 on the example assembly graph denoted in Figure 1. Figure 3(a) denotes the assembly graph after correcting labels of inconsistent vertices (after Step 2). In our example assembly graph that we have considered in Figure 1(a), the following candidate propagation actions will be executed in the given order.

(1) The candidate propagation action $(1, 0, 6, 1)$ is executed. Vertex 1 receives the red label from vertex 6 as shown in Figure 3(b).

(2) The candidate propagation action $(1, 0, 13, 14)$ is executed. Vertex 14 receives the red label from vertex 13 as shown in Figure 3(c).

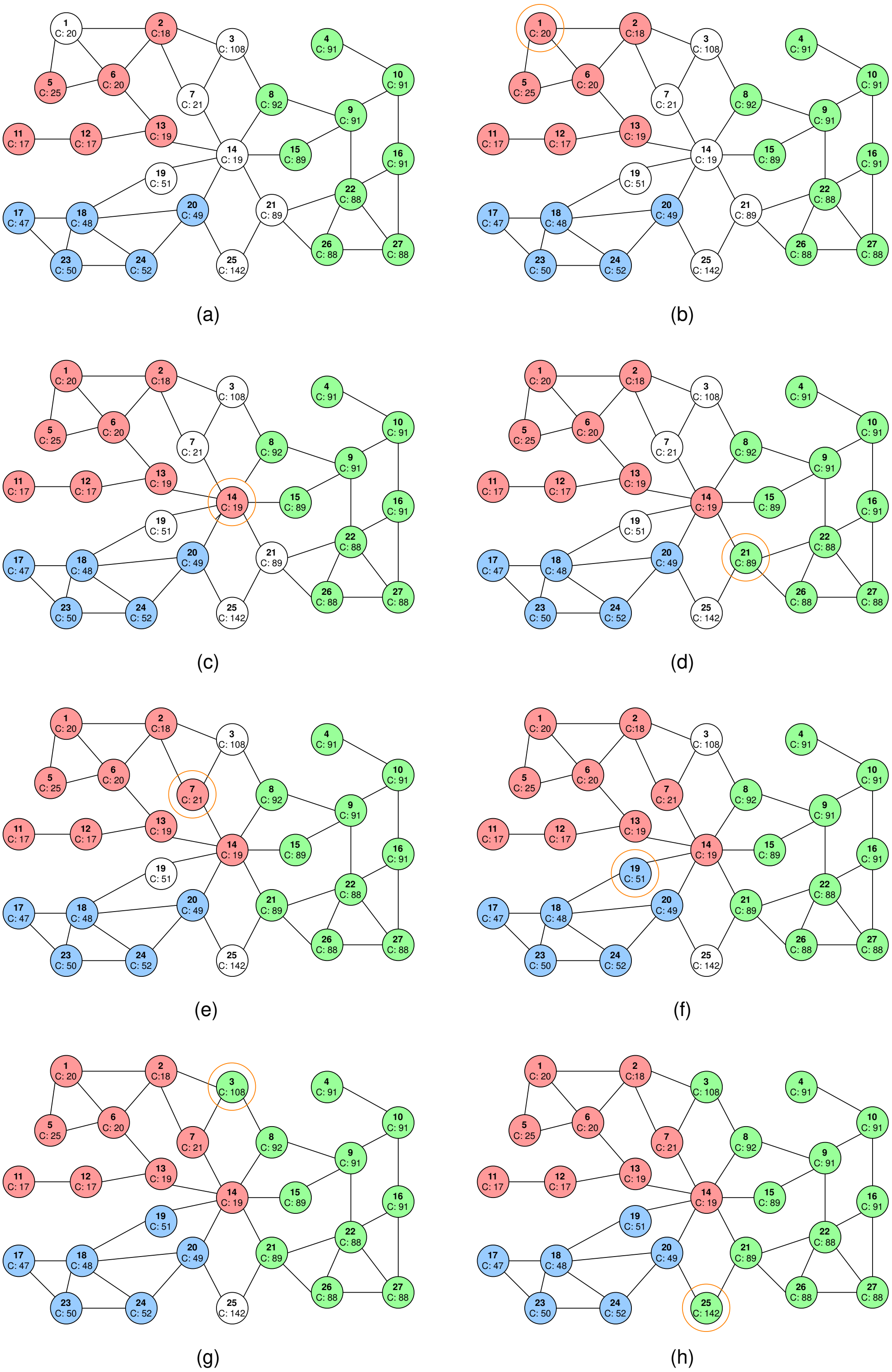
(3) The candidate propagation action $(1, 1, 22, 21)$ is executed. Vertex 21 receives the green label from vertex 22 as shown in Figure 3(d).

(4) The candidate propagation action $(1, 2, 14, 7)$ is executed. Vertex 7 receives the red label from vertex 14 as shown in Figure 3(e).

(5) The candidate propagation action $(1, 3, 18, 19)$ is executed. Vertex 19 receives the blue label from vertex 18 as shown in Figure 3(f).

(6) The candidate propagation action $(1, 16, 8, 3)$ is executed. Vertex 3 receives the green label from vertex 8 as shown in Figure 3(g).

(7) The candidate propagation action $(1, 53, 21, 25)$ is executed. Vertex 25 receives the green label from vertex 21 as shown in Figure 3(h).



(a) (b)

(c) (d)

(e) (f)

(g) (h)

Figure 3 Step-by-step illustration of how labels are propagated in Step 3 of the GraphBin2 Workflow on the example assembly graph.

B Details on the Datasets

Table 11 Information on the datasets used for the experiments.

Dataset	Assembler	Read length (bp)	Number of paired end reads	Total number of non-isolated contigs	Mean contig length (bp)	Number of species in ground truth
Sim-5G	metaSPAdes	300	2,000,000	516	51,723	5
	SGA	300	2,000,000	18,192	1,675	5
Sim-10G	metaSPAdes	300	6,999,998	900	47,279	10
	SGA	300	6,999,998	32,389	1,300	10
Sim-20G	metaSPAdes	300	15,000,001	1,404	48,021	20
	SGA	300	15,000,001	72,791	873	20
Sharon-1	metaSPAdes	100	14,869,863	371	17,144	12
	SGA	100	14,869,863	766	3,034	12
Sharon-All	metaSPAdes	100	135,493,567	2,730	7,689	12
	SGA	100	135,493,567	20,942	1,547	12

C Commands Used

C.1 Assembly Tools

metaSPAdes

```
spades --meta -1 Reads_1.fastq -2 Reads_2.fastq -o /path/output_folder -t 20
```

SGA

```
sga preprocess -o reads.fastq --pe-mode 1 Reads_1.fastq Reads_2.fastq
sga index -a ropebwt -t 16 --no-reverse reads.fastq
sga correct -k 41 --learn -t 16 -o reads.k41.fastq reads.fastq
sga index -a ropebwt -t 16 reads.k41.fastq
sga filter -x 2 -t 16 reads.k41.fastq
sga fm-merge -m 45 -t 16 reads.k41.filter.pass.fa
sga index -t 16 reads.k41.filter.pass.merged.fa
sga overlap -m 55 -t 16 reads.k41.filter.pass.merged.fa
sga assemble -m 95 reads.k41.filter.pass.merged.asqg.gz
```

C.2 Binning Tools

MaxBin2

```
perl MaxBin-2.2.5/run_MaxBin.pl -contig contigs.fasta -abund abundance.abund -thread
8 -out /path/output_folder
```

Note: abundance.abund is a tab separated file with contig ID and the coverage for each contig in the assembly. metaSPAdes provides the coverage of each contig in the contig identifier of the final assembly. We can directly extract these values to create the abundance.abund file. However, no such information is provided for contigs produced by SGA. Hence, reads should be mapped back to contigs in order to determine the coverage of SGA contigs.

SolidBin

```
python scripts/gen_kmer.py /path/to/data/contig.fasta 1000 4
sh gen_cov.sh
```

```
python SolidBin.py --contig_file /path/to/contigs.fasta --composition_profiles
/path/to/kmer_4.csv --coverage_profiles /path/to/cov_inputtableR.tsv --output
/output/result.tsv --log /output/log.txt --use_sfs
```

GraphBin

metaSPAdes version

```
python graphbin.py --assembler spades --graph /path/to/graph_file.gfa --paths
/path/to/paths_file.paths --binned /path/to/binning_result.csv --output
/path/to/output_folder
```

SGA version

```
python graphbin.py --assembler sga --graph /path/to/graph_file.asqg --binned
/path/to/binning_result.csv --output /path/to/output_folder
```

GraphBin2

metaSPAdes version

```
python graphbin2.py --assembler spades --graph /path/to/graph_file.gfa --contigs
/path/to/contigs.fasta --paths /path/to/paths_file.paths --binned
/path/to/binning_result.csv --output /path/to/output_folder
```

SGA version

```
python graphbin2.py --assembler sga --graph /path/to/graph_file.asqg --contigs
/path/to/contigs.fasta --abundance /path/to/abundance.abund --binned
/path/to/binning_result.csv --output /path/to/output_folder
```

Fast and Efficient Rmap Assembly Using the Bi-Labelled de Bruijn Graph

Kingshuk Mukherjee
Department of Computer and Information Science and Engineering, Herbert Wertheim College of Engineering, University of Florida, Gainesville, USA
kingufl@ufl.edu

Massimiliano Rossi
Department of Computer and Information Science and Engineering, Herbert Wertheim College of Engineering, University of Florida, Gainesville, USA
rossi.m@ufl.edu

Leena Salmela
Department of Computer Science, Helsinki Institute for Information Technology, University of Helsinki, Finland
leena.salmela@cs.helsinki.fi

Christina Boucher
Department of Computer and Information Science and Engineering, Herbert Wertheim College of Engineering, University of Florida, Gainesville, USA
christinaboucher@ufl.edu

Abstract

Genome wide optical maps are high resolution restriction maps that give a unique numeric representation to a genome. They are produced by assembling hundreds of thousands of single molecule optical maps, which are called Rmaps. Unfortunately, there exists very few choices for assembling Rmap data. There exists only one publicly-available non-proprietary method for assembly and one proprietary method that is available via an executable. Furthermore, the publicly-available method, by Valouev et al. (2006), follows the overlap-layout-consensus (OLC) paradigm, and therefore, is unable to scale for relatively large genomes. The algorithm behind the proprietary method, Bionano Genomics' Solve, is largely unknown. In this paper, we extend the definition of bi-labels in the paired de Bruijn graph to the context of optical mapping data, and present the first de Bruijn graph based method for Rmap assembly. We implement our approach, which we refer to as RMAPPER, and compare its performance against the assembler of Valouev et al. (2006) and Solve by Bionano Genomics on data from three genomes - *E. coli*, human, and climbing perch fish (*Anabas Testudineus*). Our method was the only one able to successfully run on all three genomes. The method of Valouev et al.(2006) only successfully ran on *E. coli* and Bionano Solve successfully ran on *E. coli* and human but not on the fish genome. Moreover, on the human genome RMAPPER was at least 130 times faster than Bionano Solve, used five times less memory and produced the highest genome fraction with zero mis-assemblies.

2012 ACM Subject Classification Theory of computation; Theory of computation → Design and analysis of algorithms

Keywords and phrases optical maps, de Bruijn graph, assembly

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.9

Supplementary Material Our software, RMAPPER is written in C++ and is publicly available under GNU General Public License at `https://github.com/kingufl/Rmapper`.

Funding *Kingshuk Mukherjee*: Supported by the National Science Foundation (NSF) IIS (Grant No. 1618814).
Massimiliano Rossi: Supported by the National Science Foundation (NSF) IIS (Grant No. 1618814).
Leena Salmela: Supported by Academy of Finland (grants 308030, 314170, and 323233).
Christina Boucher: Supported by the National Science Foundation (NSF) IIS (Grant No. 1618814).

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti; Article No. 9; pp. 9:1–9:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In 1993 Schwartz et al. developed *optical mapping* [24], a system for creating an ordered, genome wide high resolution restriction map of a given organism's genome. Since this initial development, genome wide optical maps have found numerous applications including discovering structural variations [12, 8], scaffolding and validating contigs for several large sequencing projects [9, 4], and detecting misassembled regions in draft genomes [27, 16, 20]. Thus, optical mapping has assisted in the assembly of a variety of species – including various prokaryote species [23, 31, 32], rice [33], maize [34], mouse [6], goat [7], parrot [9], and *amborella trichopoda* [4]. Bionano Genomics has enabled the automated generation of the data, enabling the data to become more wide-spread. For example, Bionano data was generated for 133 species sequenced for the Vertebrate Genomes Project.

Similar to sequencing, the protocol for producing optical mapping data, begins with many fragmented copies of the genome of interest. This redundancy allows overlap between the raw data and assembly into longer contiguous regions corresponding to the genome. With a selected enzyme, the fragments are nicked at each restriction site recognized by the enzyme. These nicked fragments are then photographed and analyzed in order to determine the length (in kbp) of the regions between nick sites. The result of this process are optical maps for all the fragments, which are referred to as *Rmaps*. For example, given a genome fragment `TTTTAACTGGGGGGGAACTTTTTTTTAACTTTTT` and an enzyme that recognizes the site `AACT` and cleaves in the middle, the resulting Rmap would be [6, 11, 11, 6]. Rmaps by themselves are not traditionally used for analysis – although, they can be [17, 8, 12] – and instead have to be assembled into longer contiguous optical maps corresponding to the genome. Hence, assembly of Rmaps refers to the problem of generating a consensus genome wide optical map from overlapping Rmaps.

Although optical mapping has been around for decades now, the problem of efficiently assembling the data largely remains open as there has been little work in this area - which is largely due to the challenges posed by the data itself. Rmap data has a number of errors that make it difficult to assemble – namely, there exists added and deleted cut sites and sizing error, resulting in extra fragments, merges in neighboring fragments and under or over-estimates of the length of a fragment. In the running example, the error free Rmap of [6, 11, 11, 6] could occur as [6, 22, 6] with error. Nonetheless, there exists two Rmap assembly methods: Gentig by Anantharaman et al. [1] and the assembler of Valouev et al. [29]. Developed in 1998, Gentig is the first Rmap assembly algorithm. It is based on a Bayesian model that seeks to maximize the *a posteriori* estimate of the consensus optical map produced by the assembly of Rmaps. It first computes the overlap between all pairs of Rmaps using dynamic programming, and then builds contigs by greedily merging the Rmaps based on alignment score. This process of merging contigs continues until all alignments above a certain score are merged. Valouev et al. [29] implemented an overlap-layout-consensus (OLC) assembly algorithm using their alignment algorithm [28], which also starts by calculating alignment between all pairs of Rmaps, and identifying all alignments that have score above a specified threshold. A graph is built, where Rmaps are represented as nodes, and the non-filtered alignments are represented as edges. The graph is refined by eliminating paths in the graph that are weakly supported. In other words, if two connected regions in the graph are joined by only a single path – or with multiple paths, but having one or more common intermediate nodes – then the graph is disconnected at these nodes. Further, an edge is removed if it is inconsistent with a higher scoring edge. Contigs are then generated by traversing this graph in a depth first manner. Bionano Genomics Inc. provides a proprietary assembly method, called Bionano Solve, however the source code is not publicly available and the algorithmic details are unknown due to the proprietary nature of the software.

The alternative to an OLC approach for assembly is Eulerian assembly that relies on building and traversing the de Bruijn graph. For simplicity, we give a constructive definition of the de Bruijn graph in the context of genome assembly. Given a set of sequences $R = \{r_1, \ldots, r_n\}$ and an integer k, the de Bruijn graph is constructed by creating a directed edge for each unique k length substring (k-mer) with the nodes labeled as the $k-1$ length prefix and $k-1$ length suffix of the k-mer, and then all nodes that have the same label are merged. The important aspect of Eulerian assembly is that it avoids having to find alignments between any pair of sequences, leading to an $\mathcal{O}(n)$ run-time. Since its introduction by Idury et al. [11] and Pevzner et al. [22], Eulerian assembly has become the most common paradigm for assembling short read sequencing data because it led to huge gains in performance over OLC approaches. Hence, applying a Eulerian approach to Rmap assembly would likely lead to similar improvements by removing the burden of finding all pairwise alignments between Rmaps. The challenge we face is constructing a de Bruijn graph with added and deleted cut-sites and sizing error. Eulerian assembly works on the premise that a k-mer will occur exactly without error frequently in the data. Even without the occurrence of added and deleted cut-sites, k-mers created from Rmap data are unlikely to be exact replicas due to sizing error. For example, [6, 11, 11, 6] and [5, 10, 11, 7] should likely be recognized as instances of the same k-mers in Rmap data. Thus, to overcome this challenge the de Bruijn graph has to be redefined to account for the inexactness of the data.

In this paper, we formulate and describe an Eulerian approach for de novo Rmap assembly, which heavily relies on redefining the de Bruijn graph to make it suitable for Rmap data. We accomplish this by extending the definition of a bi-label in the context of the paired de Bruijn graph that was introduced by Medvedev et al. [14]. We refer to our modified de Bruijn graph as *bi-labelled de Bruijn graph.* Next, we demonstrate how to efficiently build and store the de Bruijn graph using a two tier orthogonal-range search data structure. We implement this approach, leading to a novel Rmap assembler that we call RMAPPER. We compare the performance of our method with the assembler of Valouev et al., and Bionano Solve on three genomes of varying size: *E. coli*, human, climbing perch (a fish species from the Vertebrate Genomes Project). Our comparison demonstrates that RMAPPER was more than 130 times faster and used less than five times less memory than Solve, and was more than 2,000 times faster than Valouev et al. Consequently, RMAPPER was the only method able to scale to the largest Rmap dataset: climbing perch. It successfully assembled the 3.1 million Rmaps of the climbing perch genome into contigs that covered over 87% of the draft genome with zero mis-assemblies.

2 Background and definitions

2.1 Rmap Data and Genome Wide Optical Maps

From a computer science perspective, we can view an Rmap $R = [r_1, r_2, \ldots, r_{|R|}]$ as an ordered list of integers. Each number represents the length of the respective fragment. The *size* of an Rmap R denotes the number of fragments in R, which we denote as $|R|$. For example, say we have an enzyme that cleaves the DNA at the middle position of `AACT` and a genomic sequence `TTTTAACTGGGGGGGAACTTTTTTTTAACTTTTT`, then the Rmap will be $R = [6, 11, 11, 6]$ corresponding to the cleaved sequences [`TTTTAA, CTGGGGGGGAA, CTTTTTTTTAA, CTTTTT`].

2.2 Error Profile of Rmap Data

There are three types of errors that can occur in optical mapping: (1) missing cut sites which are caused by an enzyme not cleaving at a specific site, (2) additional cut sites which can occur due to random DNA breakage and (3) inaccuracy in the fragment size due to the inability of the system to accurately estimate the fragment size. Continuing again with the example above, an example of an additional cut site would be when the second fragment of R is split into two, e.g., $R' = [6, 5, 6, 11, 6]$, and an example of a missing cut site would be when the last two fragments of R are joined into a single fragment, e.g., $R' = [6, 11, 17]$. Lastly, an example of a sizing error would be if the size of the first fragment is estimated to be 7 rather than 6.

Several different probabilistic models have been proposed for describing the sizing error, and the frequency of added and missed cut-sites, including the models of Valouev *et al.* [28], Li *et al.* [13], and Chen et al. [5]. We briefly describe these models here but refer to the original papers for a full description. Both Valouev et al. and Chen et al. describe the observed fragment lengths as normal distribution with the mean being equal to the true length of the fragment and the standard deviation being a function of the true length, i.e. longer fragments exhibit larger standard deviation. In the model by Li *et al.* the sizing error uses a Laplace distribution as follows: if the observed and actual size of a fragment are o_i and r_i, respectively, then the sizing error, $o_i \sim r_i \times Laplace(\mu, \beta)$ where μ and β are parameters of the Laplace distribution and are functions of r_i. All studies model the probability of having a missed cut-site as a Bernoulli trial. Valouev et al. and Chen et al. predict a fixed probability for digestion of a cut-site while Li *et al.* model the probability of digestion as a function of lengths of the fragments flanking the cut-site. The likelihood of a missed cut-site decreases with the length of the fragment. All three models postulate additional or false cut-sites result from random breaks of the DNA molecule and hence model the number of false cuts per unit length of DNA as a Poisson distribution. Li et al. observed that false cuts occurred less frequently at the two ends of an Rmap.

2.3 Rmap Segments and k-mers

We define a *segment* $s_{p,q}$ of an Rmap starting at position p and ending at position q, as the $q - p + 1$ consecutive fragments starting from r_p, i.e., $[r_p, r_{p+1}, .., r_q]$. We define the *length* of a segment as the summation of all of its constituent fragments, i.e., $r_p + \cdots + r_q$. We denote the length of a segment $s_{p,q}$ as $\ell(s_{p,q})$. We note that the length of the Rmap R should not be confused with the number of fragments, which we denote as its size $|R|$.

In this paper, we extend the definition of a k-mer to the context of Rmap data as follows. Given an integer k, we define a k-mer as a segment of exactly k fragments, i.e., a sequence of k successive fragments of an Rmap. Following the example from above, the following two 3-mers exist in $R = [6, 11, 11, 6]$: $[6, 11, 11]$ and $[11, 11, 6]$.

2.4 Prefixes and Suffixes of Rmaps

Given an Rmap $R = [r_1, r_2, \ldots, r_{|R|}]$, we define the x-size *prefix* of R as $R = [r_1, r_2, \ldots, r_x]$, where x is at most $|R| - 1$. Conversely, we define the x-size *suffix* of R as $R = [r_{|R|-x+1}, \ldots, r_{|R|}]$, where x is at most $|R| - 1$.

3 The Bi-labelled de Bruijn Graph

In this section, we modify the traditional definition of the de Bruijn graph for Rmap data by first redefining the concept of a bi-label for Rmap data. The term bi-label was first introduced by Medvedev et al. [14] in the context of short read assembly to incorporate mate-pair data into assembly of paired-end reads. There the term bi-label refers to two k-mers separated by a specified genomic distance. The redefinition of the de Bruijn graph with this extra information was shown to de-tangle the resulting graph, making traversal more efficient and accurate. Here, we demonstrate that an equivalent paradigm can be effective for Rmap assembly.

3.1 Bi-labels

Given integers k and D, and Rmap R, we define a *bi-label* from an Rmap R, as a segment of R containing a pair of k-mers separated by the shortest segment that has a length of at least D. The following is a formal definition.

▶ **Definition 1.** *Given an Rmap $R = [r_1, r_2, ..., r_i, r_{i+1}, .., r_{|R|}]$, integers k and D, and a position i, we define the bi-label at position i to be $[s_k^1, r_p, \ldots, r_q, s_k^2]$, where $p = i + k$ and q is an index such that $\ell(s_{p,q-1}) < D \leq \ell(s_{p,q})$ and s_k^1 and s_k^2 are the k-mers starting at positions i and $q + 1$, respectively.*

Next, we refer to segment $s_{p,q}$ between s_k^1 and s_k^2 as the *skip segment*, and note that, unlike s_k^1 and s_k^2 which both have k fragments, this segment is only bounded by its length and can have any number of fragments. Thus, this accounts for added and deleted cut-sites since these errors do not impact the length of a segment. Figure 1 demonstrates how the skip-segment tolerates a deleted cut-site.

For example, given $k = 3$, $D = 25$, and $R = [7, 18, 13, 3, 15, 12, 4, 3, 6, 5, 13, 2]$, the bi-labels of R are $\Big([7, 18, 13]\Big|[3, 15, 12]\Big|[4, 3, 6]\Big)$, $\Big([18, 13, 3]\Big|[15, 12]\Big|[4, 3, 6]\Big)$ and $\Big([13, 3, 15]\Big|[12, 4, 3, 6]\Big|[5, 13, 2]\Big)$.

We are now going to define the prefix and suffix bi-labels.

▶ **Definition 2.** *Given integers D and k and bi-label b with k-mers $b^1 = [b_1^1, ..b_k^1]$ and $b^2 = [b_1^2, .., b_k^2]$ and skip segment b^s, we define the* prefix bi-label *of b as the bi-label with $(k-1)$-mers and skip-segment length at least D, where the first $(k-1)$-mer is the $(k-1)$-size prefix of b^1 i.e. $[b_1^1, ..b_{k-1}^1]$.*

Note that the second $(k-1)$-mer of the prefix bi-label is not necessarily the $(k-1)$-size prefix of b^2. We also require an equivalent definition for the suffix of a bi-label.

▶ **Definition 3.** *Given integers D and k and bi-label b with k-mers $b^1 = [b_1^1, ..b_k^1]$ and $b^2 = [b_1^2, .., b_k^2]$ and skip segment b^s, we define the* suffix bi-label *of b as the bi-label with $(k-1)$-mers and skip-segment length at least D, where the first $(k-1)$-mer is the $(k-1)$-size suffix of b^1 i.e. $[b_2^1, ..b_k^1]$.*

Figure 2 illustrate this concept of prefix and suffix bi-labels. Note that for two successive bi-labels from an Rmap, the prefix bi-label of the latter is the same as the suffix bi-label of the former as shown in Figure 2. This is a vital property that allows the de Bruijn graph constructed over bi-labels to be connected.

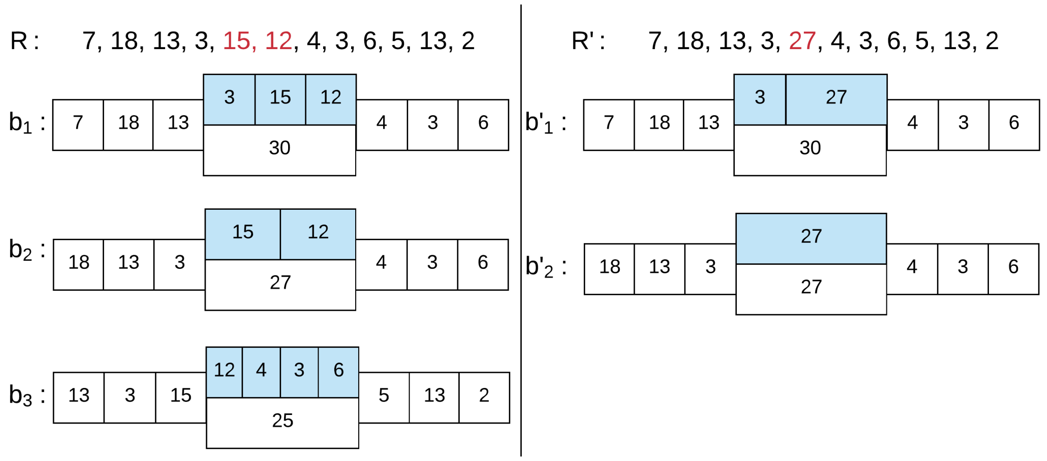

R : 7, 18, 13, 3, 15, 12, 4, 3, 6, 5, 13, 2
R' : 7, 18, 13, 3, 27, 4, 3, 6, 5, 13, 2
b1 : 7 18 13 | 3 15 12 | 30 | 4 3 6
b2 : 18 13 3 | 15 12 | 27 | 4 3 6
b3 : 13 3 15 | 12 4 3 6 | 25 | 5 13 2
b'1 : 7 18 13 | 3 27 | 30 | 4 3 6
b'2 : 18 13 3 | 27 | 27 | 4 3 6

Figure 1 All bi-labels for $k = 3$ and $D = 25$ of two Rmaps R and R', $\{b_1, b_2, b_3\}$ and $\{b'_1, b'_2\}$ respectively. Both Rmaps cover the same genomic location but R' has a missed cut-site in position 5 (shown in red). On each bi-label the fragments from the k-mers and the length of the skip segment are shown in white while the fragments of the skip segment are shown in blue. Despite the missed cut-site on R' bi-labels b_1 and b_2 are merged to b'_1 and b'_2 respectively according to our merge function.

3.2 Bi-label Proximity

One of the challenges with Rmap data is the fact that the fragments correspond to genomic distances and due to experimental error, the measured estimates for the same genomic fragment are different across different Rmaps representing the same genomic location. For example, $R = [5, 6, 7, 11, 5]$ and $R' = [6, 5, 6, 11, 6]$ likely correspond to the same k-mer but the numerical nature makes it such that they are not exactly equal. Thus, we need to define a criteria such that two bi-labels drawn from different Rmaps but corresponding to the same genomic locations can be identified and merged for the construction of the de Bruijn graph. Thus, to make the definition of a bi-label robust to sizing errors, we define conditions on both the difference of the individuals fragments of two bi-labels and the difference in the total lengths. Hence, we have the following definitions.

▶ **Definition 4.** *Given integers t_f, k and D, and two bi-labels a and b, we let the k-mers of a and b be $a^1 = [a^1_1, .., a^1_k]$ and $a^2 = [a^2_1, .., a^2_k]$ and $b^1 = [b^1_1, .., b^1_k]$ and $b^2 = [b^2_1, .., b^2_k]$, respectively. We define a and b to be* fragment proximal *if and only if $|a^1_i - b^1_i| \leq t_f$ and $|a^2_i - b^2_i| \leq t_f$ for all $i = 1, .., k$.*

Here t_f is an error-tolerance parameter that handles sizing errors on the fragments of the bi-label.

▶ **Definition 5.** *Given integers t_ℓ, k and D, and two bi-labels a and b, we let the k-mers of a and b be a^1 and a^2 and b^1 and b^2, respectively, and the skip segment of a and b be a^s and b^s, respectively. We define a and b to be* length proximal *if and only if $|\ell(a^1) - \ell(b^1)| \leq t_\ell$, $|\ell(a^2) - \ell(b^2)| \leq t_\ell$ and $|\ell(a^s) - \ell(b^s)| \leq t_\ell$.*

Here t_ℓ is another error-tolerance parameter that handles sizing errors on the segment lengths of the bi-label. These two definitions lead to our final definition that defines whether two bi-labels should be defined as equivalent in the de Bruijn graph.

▶ **Definition 6.** *Given integers k and D and two bi-labels a and b, we define them to be* proximal *if and only if they are fragment proximal and length proximal.*

This leads to our final definition, which is the set of bi-labels in which the bi-labelled de Bruijn graph is defined on.

▶ **Definition 7.** *Given a set of Rmaps $\{R_1, .., R_n\}$ and integers k and D, let B be the set of bi-labels from R. We define the* proximal reduced *set of bi-labels as the set B', where for each b in B there is a bi-label in B' that it is proximal to.*

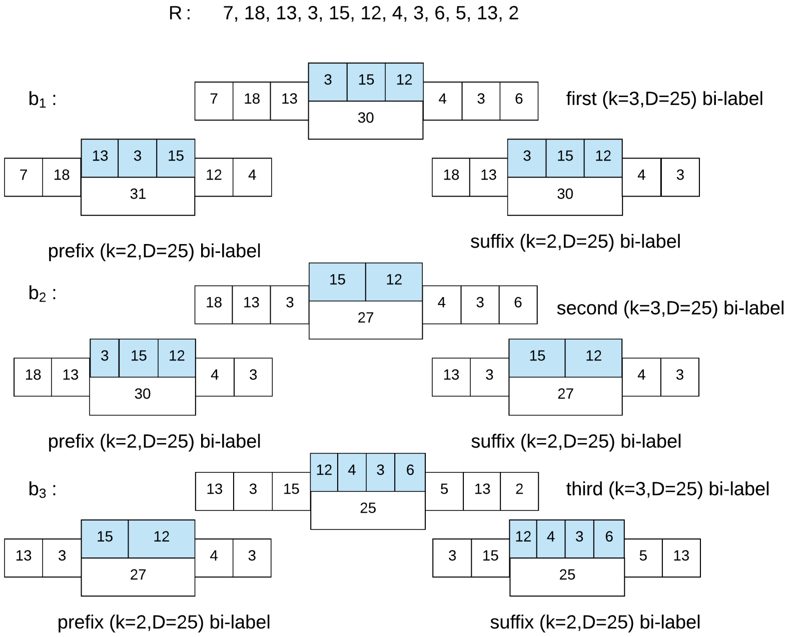

R : 7, 18, 13, 3, 15, 12, 4, 3, 6, 5, 13, 2
b₁ :
3 15 12
7 18 13 30 4 3 6
first (k=3,D=25) bi-label
13 3 15
7 18 31 12 4
prefix (k=2,D=25) bi-label
3 15 12
18 13 30 4 3
suffix (k=2,D=25) bi-label
b₂ :
15 12
18 13 3 27 4 3 6
second (k=3,D=25) bi-label
3 15 12
18 13 30 4 3
prefix (k=2,D=25) bi-label
15 12
13 3 27 4 3
suffix (k=2,D=25) bi-label
b₃ :
12 4 3 6
13 3 15 25 5 13 2
third (k=3,D=25) bi-label
15 12
13 3 27 4 3
prefix (k=2,D=25) bi-label
12 4 3 6
3 15 25 5 13
suffix (k=2,D=25) bi-label

Figure 2 All bi-labels for $k = 3$ and $D = 25$ of an Rmap R. On each bi-label the fragments from the k-mers and the length of the skip segment are shown in white while the fragments of the skip segment are shown in blue. For each bi-label we show the prefix and suffix bi-labels built with $k = 2$ and $D = 25$.

3.3 Definition of the Bi-labelled de Bruijn Graph

Given the above definitions, we are now ready to define the bi-labelled de Bruijn graph built on a set of proximal bi-labels extracted from Rmaps.

▶ **Definition 8.** *Given integers k and D and set of Rmaps $\{R_1, .., R_n\}$, let B be the set of proximal bi-labels extracted from R. We create a directed edge e for each bi-label b in B and label the incoming and outgoing nodes of e as the prefix bi-label of b and suffix bi-label of b, respectively. After all edges are formed, the graph undergoes a gluing operation. A pair of node bi-labels are glued into a single node if and only if they are proximal. We define the final graph obtained after gluing of nodes as the bi-labelled de Bruijn graph.*

4 Methods

In this section, we describe our method for building and traversing the bi-labelled de Bruijn graph from an Rmap dataset. Our method, which we refer to as RMAPPER, can be summarized into the following steps: extract and store bi-labels, find proximal bi-labels, build the bi-labelled de Bruijn graph, resolve tips and bubbles, and traverse the graph to build the contigs. We now describe each of these steps in detail.

4.1 Extract and Store all Bi-lablels

We first error correct the Rmap data using cOMet [18] and then extract and store all bi-labels from the error corrected Rmaps. We recall from Definition 6 that two bi-labels are proximal if they are both fragment proximal as well as length proximal for error-tolerance parameters t_f and t_ℓ. Therefore, we must store all the bi-labels in a manner that allows finding all proximal bi-lablels of a given bi-label efficiently. To accomplish this, we store all the bi-labels in a disjoint set of k-d trees [3] such that each pair of bi-labels in the same k-d tree are length proximal. For each bi-label, the $2k$ fragments of the k-mers of it are stored in the corresponding k-d tree, which will allow for efficiently finding all fragment proximal bi-labels of a given bi-label. Hence, the dimension of each k-d tree is $2k$.

More formally, we identify each k-d tree $\mathcal{K}_{a_1,a_2,a_3}$ by three positive integers a_1, a_2, and a_3, and insert a given bi-label b into $\mathcal{K}_{a_1,a_2,a_3}$ if the length of its two k-mers $\ell(b^1)$ and $\ell(b^2)$ are within the range $[a_1 \times t_\ell, \dots, (a_1+1) \times t_\ell - 1]$ and $[a_2 \times t_\ell, \dots, (a_2+1) \times t_\ell - 1]$ respectively and the length of the skip segment $\ell(b^s)$ is also within the range $[a_3 \times t_\ell, \dots, (a_3+1) \times t_\ell - 1]$. If such a tree does not exist then we create a new one with $\mathcal{K}_{a_1,a_2,a_3}$, where $a_1 = \lfloor \ell(b^1)/t_\ell \rfloor$, $a_2 = \lfloor \ell(b^2)/t_\ell \rfloor$ and $a_3 = \lfloor \ell(b^s)/t_\ell \rfloor$.

Next, for each bi-label in our set of k-d trees, we find and store pointers to all proximal bi-labels by performing an orthogonal range query. Given a bi-label b in $\mathcal{K}_{a_1,a_2,a_3}$, we let the k-mers of the bi-label b be $b^1 = [b^1_1, .., b^1_k]$ and $b^2 = [b^2_1, .., b^2_k]$. We perform a range query with $([b^1_1 \pm t_f], \dots, [b^1_k \pm t_f], [b^2_1 \pm t_f], \dots, [b^2_k \pm t_f])$ in the disjoint set of k-d trees to find all bi-labels whose first k-mer is equal to $[b^1_1 \pm t_f], \dots, [b^1_k \pm t_f]$ and whose second k-mers is equal to $[b^2_1 \pm t_f], \dots, [b^2_k \pm t_f]$. We add a pointer from b to each of these bi-labels. We repeat this for each bi-label. In particular, we perform the range query in all k-d trees where the proximal bi-labels can be found, i.e., all k-d trees $\mathcal{K}_{a'_1,a'_2,a_3}$ where for $m = min(kt_f, t_\ell)$ we have, $\lfloor (\ell(b^1) - m)/t_\ell \rfloor \le a'_1 \le \lfloor (\ell(b^1) + m)/t_\ell \rfloor$ and $\lfloor (\ell(b^2) - m)/t_\ell \rfloor \le a'_2 \le \lfloor (\ell(b^2) + m)/t_\ell \rfloor$.

We note that k-d trees support multi-dimensional orthogonal range-search queries in $\mathcal{O}(n^{(2k-1)/2k} + occ)$ time and $\mathcal{O}(n)$ space where n is the number of bi-labels in the tree, k is the k-mer value, and occ is the number of bi-labels that satisfy the constrains of the range-search query.

4.2 Graph Construction

We first filter all low frequency bi-labels, i.e., bi-labels that have a low number of proximal bi-labels. As illustrated in Figure 3, bi-labels that have low frequency typically arise from Rmap data that is highly erroneous. After filtering low frequency bi-labels, we build the bi-labelled de Bruijn graph by first building a proximal reduced set from the unfiltered bi-labels, then building all directed edges with labelled nodes from the reduced set, and finally merging nodes that have the same label. Using an efficient heuristic, we first greedily find the proximal reduced set of bi-labels by sorting the unfiltered bi-labels in descending order based on the number of proximal bi-labels found for them. From this sorted list of bi-labels B, we iteratively insert bi-labels into the reduced set B' unless the bi-label is proximal to a bi-label already in B'.

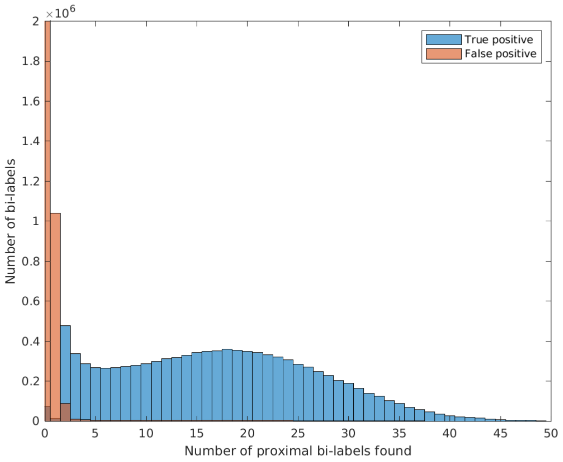


Figure 3 Histogram showing the precision of finding proximal bi-labels. For simulated human Rmap data, we found proximal bi-labels for all extracted bi-labels. We designate a proximal bi-label found to be a true positive if its true location in the genome is the same as the location of the bi-label to which it is proximal - and false positive otherwise. Next, we plotted a histogram showing the distribution of true positives and false positive proximal bi-labels for each bi-label. We show that high frequency bi-labels i.e. bi-labels for which we find more proximal bi-labels produce more precise proximal bi-labels. This justifies filtering low frequency bi-labels.

Next, we build a bi-labelled de Bruijn graph by creating a directed edge for each bi-label b' in B' and labeling the incoming and outgoing nodes as the prefix bi-label and suffix bi-label of b'. We store all the nodes and edges in a modified adjacency list format that contains three arrays: one array stores all node bi-labels, one array containing a list of pointers of the incoming nodes for each node, and lastly, one array containing a list of pointers of the outgoing nodes for each node. Thus, to insert b' into the graph, we first determine if the prefix and suffix bi-labels are contained in the node array and insert them if they are not contained in the list, and then insert an entry into the incoming and outgoing arrays with lists containing pointers to the prefix and suffix bi-labels. This graph representation will allow for the adjacency lists of two nodes to be efficiently merged if the bi-labels they represent are found to be proximal.

Lastly, we merge all nodes in the graph whose bi-labels are proximal to obtain the final bi-labelled de Bruijn graph. For merging the nodes, we again use a set of disjoint k-d trees as we did before for finding proximal bi-labels for the edge bi-labels. Hence, we extract all the node bi-labels and construct a set of k-d trees as before. Then for each node v in the node array, we query the corresponding k-d trees to find all nodes that are proximal to it using the same error tolerance parameters t_f and t_ℓ. Any node u that is found to be proximal to v is merged to v by removing u from the graph by updating the two adjacency lists such that the incoming and outgoing array entries storing pointers to u are updated to store pointers to v. This can be achieved in linear time. We repeat this until all proximal nodes have been merged. Figure 4 illustrates the construction of the bi-labelled de Bruijn graph for a pair of Rmaps.

4.3 Graph Cleaning and Traversal

Before traversing the graph, we first pre-process the bi-labelled de Bruijn graph to remove *tips* and *bubbles*, which are common in de Bruijn graphs. Since they limit the size of unary paths (i.e. paths in the graph that contain nodes with only a single outgoing edge) and

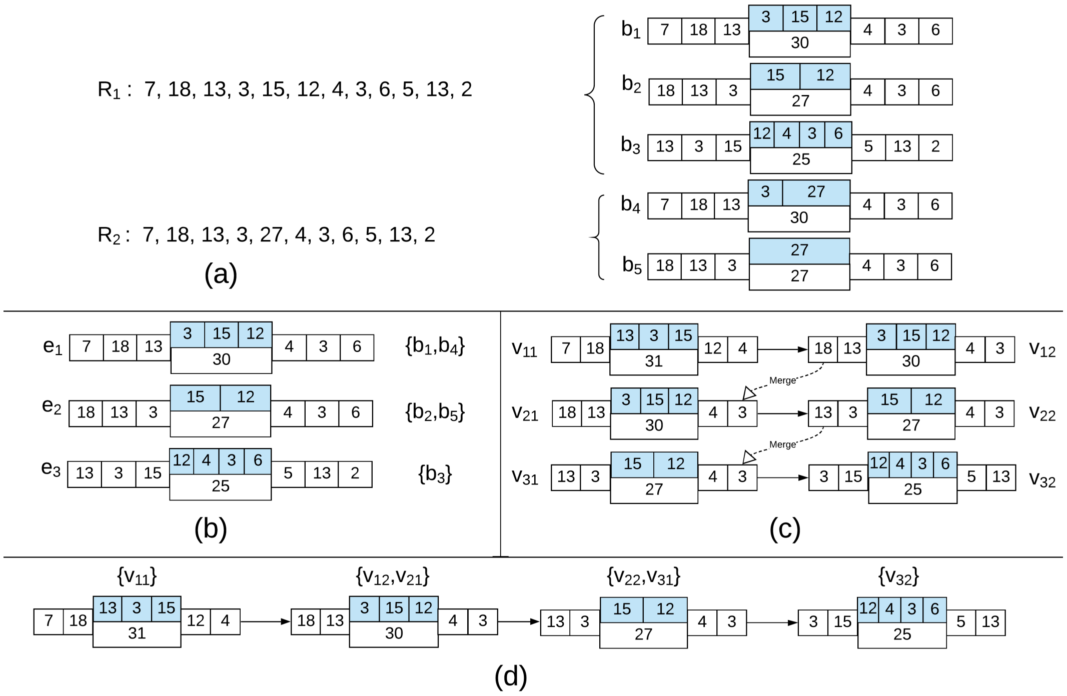

R1 : 7, 18, 13, 3, 15, 12, 4, 3, 6, 5, 13, 2
R2 : 7, 18, 13, 3, 27, 4, 3, 6, 5, 13, 2
(a)
b1 7 18 13 3 15 12 30 4 3 6
b2 18 13 3 15 12 27 4 3 6
b3 13 3 15 12 4 3 6 25 5 13 2
b4 7 18 13 3 27 30 4 3 6
b5 18 13 3 27 27 4 3 6
e1 7 18 13 3 15 12 30 4 3 6 {b1,b4}
e2 18 13 3 15 12 27 4 3 6 {b2,b5}
e3 13 3 15 12 4 3 6 25 5 13 2 {b3}
(b)
v11 7 18 13 3 15 31 12 4
18 13 3 15 12 30 4 3 v12
v21 18 13 3 15 12 30 4 3
13 3 15 12 27 4 3 v22
v31 13 3 15 12 27 4 3
3 15 12 4 3 6 25 5 13 v32
Merge
Merge
(c)
{v11}
{v12,v21}
{v22,v31}
{v32}
7 18 13 3 15 31 12 4
18 13 3 15 12 30 4 3
13 3 15 12 27 4 3
3 15 12 4 3 6 25 5 13
(d)

Figure 4 The construction of the bi-labelled de Bruijn Graph. (a) Two Rmaps R_1 and R_2 and the bi-labels extracted from them – $\{b_1, b_2, b_3\}$ from R_1 and $\{b_3, b_4\}$ from R_2 for $k = 3$ and $D = 25$. (b) Edges $\{e_1, e_2, e_3\}$ depict the proximal reduced set of bi-labels. Bi-labels $\{b_1, b_4\}$ are represented by e_1, bi-labels $\{b_2, b_5\}$ are represented by e_2 and bi-label $\{b_3\}$ forms e_3. We note that in this example no bi-labels are filtered for finding the proximal reduced set. (c) Nodes introduced into the graph. Each edge breaks into two nodes - one denoted by the prefix bi-label and the other by suffix bi-label of the edge. A directed edge is drawn from the former to the latter. (d) The final graph is formed by merging nodes v_{12} with v_{21} and merging v_{22} with v_{32}.

do not affect the accuracy of the assembly, it is common practice in short read assembly to resolve or remove these structures [2, 30, 26, 21]. Tips are produced when errors cause an otherwise unary path to branch at a node and create a short unary path that ends in a terminal node. Bubbles are created when bi-labels from the same genomic location are not merged and included in the graph as separate edges. This generates short unary paths that have the same starting node and the same ending node and are close in length.

Similar to existing short read assemblers, we identify all tips and bubbles that have length of at most a specified threshold by performing depth first search starting at each node with out-degree greater than one. Hence, if there exists a tip starting at a given node as well as a path of length longer than the specified threshold, then the tip is removed by deleting all of its edges starting at the branching node. Furthermore, if there exists a bubble starting at a given node, we remove one of the edges adjacent to the branching node. We note we do not remove an entire path from the graph to resolve a bubble – rather, we only disconnect them at the branching node. Following the work of Simpson et al. [26], we fix the maximum length of the paths in a bubble to twice the size of the bi-label.

After cleaning, our traversal algorithm extracts unitigs (i.e. contigs corresponding to unary paths) from the graph by performing a simple depth first traversal starting from each node with zero incoming edges. We terminate the traversal of a given path if a cycle is reached or a node with out-degree greater than one is reached.

5 Experiments

In this section, we compare the performance of RMAPPER, the assembler of Valouev *et al.* and Bionano Solve. We used the most recent version of Bionano Solve that is publicly available (version 3.5.1.). We performed all experiments on Intel E5-2698v3 processors with 192 GB of RAM running 64-bit Linux. Valouev and RMAPPER were ran on error corrected data. Bionano Solve was not because the input is required to be specified in their proprietary format. In addition, for larger genomes, we also ran RMAPPER by extracting bi-labels from both directions in an Rmap. We refer to this as RMAPPER2.0.

For all experiments we report the run time (CPU time), peak memory, maximum and mean contig size, genome fraction and number of mis-assembled contigs. We note that genome assembly evaluation tools such as QUAST [10] cannot be used on optical maps - hence, we design our own evaluation setup. To compute the genome fraction, we align all assembled contigs to the optical map reference genome using the alignment method of Valouev et al. [28]. The optical map reference genome is produced by *in silico* digesting the reference genome using the same restriction enzyme as used for producing the Rmaps. For all contigs that were successfully aligned, we designate their alignment locations on the reference genome as covered and report the percentage of the genome covered by at least one contig as the genome fraction. Any contig which is unable to be aligned by Valouev et al. is verified to be mis-assembled by aligning it to the reference genome using a second alignment software - Bionano's RefAligner. The Valouev method aligns an assembled contig to a contiguous stretch of the reference optical map that optimizes its alignment score and does not tolerate mis-assembled regions. Whereas, RefAligner allows split alignments. Hence, if the alignment outputted from RefAligner is uncontiguous then it is counted as a mis-assembly.

RMAPPER takes as input four parameters, namely the size k of the k-mers, the minimum distance D between the two k-mers in the bi-label, and the error tolerance parameter setting t_f and t_ℓ. The k-mer size depends on the error-rate of the Rmap data. When the frequency of added and missed cut-sites is high, the k-mer size needs to be set low so that a good percentage of k-mers are error-free. We node that the average error-rate of optical-map data typically lies between 14% to 16%. Considering that error-correcting the Rmaps brings the average error-rate below 10%, the k-mer size of 6 is the largest value such that the probability that an extracted k-mer will be error-free is at least 50%. Hence we use 6 as the default k-mer size in our experiments. The best combination of coverage, average length of contigs and run-time is achieved by fixing $t_\ell = 2000$. We experimented with the following values of $D = \{15\,000, 20\,000, 25\,000, 30\,000\}$ and the following values of $t_f = \{500, 1000, 1500\}$ and for each experiment, we choose the parameter setting that gives the best performance. A higher value of t_f is needed when the Rmap data still has significant sizing errors after error correction. A lower value of D is needed when the average Rmap size is small so that we can extract an adequate number of bi-labels from each Rmap.

5.1 Datasets

We performed experiments on both simulated and real Bionano datasets. We simulated data from both *E. coli* K-12 substr. MG1655 genome and the human reference genome GRCh38 (NCBI accession number GCF_000001405.26) with OMSim [15]. We used enzyme BspQI – a standard, commonly used restriction enzyme for optical mapping – and used the default error rate of OMSim, which is a 15% rate of deleted cut sites, and 1 added cut site per 100kbp. The resulting *E. coli* dataset contains 23 450 Rmaps with a mean of 42 fragments per Rmap. The Human dataset contains 377 894 Rmaps with a mean of 61 fragments per Rmap.

Lastly, we performed experiments using the Rmap dataset of the climbing perch (*Anabas testudineus*) genome generated for the Vertebrate Genomes Project, which consists of 3 121 480 Rmaps with mean of 28 fragments. A draft assembly of the genome is provided from the same source which was used to obtain the reference genome optical map.

5.2 Performance on E. coli

The results on *E. coli* Rmap dataset is summarized in Table 1. For this experiment we extracted bi-labels with $k = 6$ and $D = 15\,000$ and used error tolerance parameter setting $t_f = 500$ and $t_\ell = 2000$. RMAPPER took 342 seconds and peak memory of 274 Mb to assemble the data. The assembler produced two unitigs that are 529 and 522 fragments in length, which covered the reference from start to finish.

Table 1 Assembly results for E. coli Rmap data simulated by OMSim using enzyme BspQI. The dataset has 23,450 Rmaps of mean size of 42 fragments and coverage of 900x. The peak memory is given in gigabytes (GB). The run time is reported in second (s) minutes (m), hours (h) and days (d). RMAPPER was run with $k = 6$, $D = 15\,000$ and error tolerance parameter setting $t_f = 500$ and $t_\ell = 2000$. The contig with maximum length (Max) is reported in the number of fragments and the total genomic length in mega base pairs (Mbp). Similarly, the mean contig length (Mean) is also reported in the number of fragments and the total genomic length in mega base pairs. The genome fraction (GF) is the percentage of the genome that is covered by at least one contig. Lastly, the number of mis-assembled contigs (MA) is given.

Assembler	Run time	Peak Memory	No. of contigs	Max	Mean	GF(%)	MA
Valouev	8.5 d	0.48	5	102 (1.0 Mbp)	56 (0.5 Mbp)	48	0
Solve	48.1 h	1.18	1	631 (4.9 Mbp)	631 (4.9 Mbp)	100	0
RMAPPER	6 m	0.46	2	529 (4.6 Mbp)	526 (4.5 Mbp)	100	0

The Valouev assembler [29] took 204.8 hours to compute pairwise alignments between all pairs of Rmaps and an additional 30 minutes to assemble them into contigs. It produced 5 contigs with the longest contig of length 102 fragments (corresponding to a 1Mbp genomic span). We aligned the assembled contigs back to the reference and found the total genome coverage to be 48%. Bionano solve produced a high quality assembly, i.e., one contig that spanned 100% of the genome. The assembly took 48.14 hours of CPU time (59.75 minutes of wall time using 60 CPUs in parallel) and peak memory of 1.18 GB. The Valouev aligner reported alignments for all contigs, hence we report zero mis-assembled contigs for all three methods.

In summary, the quality of Bionano Solve and RMAPPER were comparable, yet RMAPPER was 480 times faster (6 minutes versus 2889 minutes) and used less than 500 Mb of memory.

5.3 Performance on Human

The results on the Human Rmap dataset are shown in Table 2. For this experiment we extracted bi-labels with $k = 6$ and $D = 25\,000$ and used error tolerance parameter setting $t_f = 1500$ and $t_\ell = 2000$. RMAPPER took 12.1 hours and peak memory of 7.9 GB to assemble the data whereas RMAPPER 2.0 took 22.2 hours and 18.8 GB of peak memory. RMAPPER produced 3134 contigs whereas RMAPPER 2.0 produced 2867 contigs. The maximum size unitig produced by RMAPPER and RMAPPER2.0 was 1380 and 1752 fragments in length, respectively. Lastly,. RMAPPER achieved a net coverage of 95.8% while RMAPPER2.0 was able to cover 96.7% of the genome - both with zero mis-assembled contigs.

Table 2 Assembly results for human Rmap data simulated by OMSim using enzyme BspQI. The dataset has 377 894 Rmaps of mean size of 61 fragments and coverage 80x. See Table 1 for a description of the assembly statistics and notation. As described in the text, RMAPPER2.0 extracts bi-labels from Rmaps in both forward and reverse directions.

Assembler	Run time	Peak Memory	No. of contigs	Max	Mean	GF(%)	MA
Valouev	> 360 d	n/a	n/a	n/a	n/a	n/a	n/a
Solve	122.4 d	94.8	169	14,133 (124.6 Mbp)	2,036 (16.4 Mbp)	93.8	4
RMAPPER	12.1 h	7.9	3865	1,380 (14.4 Mbp)	144 (1.4 Mbp)	95.8	0
RMAPPER 2.0	22.2 h	18.8	3524	1,752 (18.5 Mbp)	203 (2.0 Mbp)	96.7	0

The Valouev assembler did not produce any output after 360 CPU days so n/a is reported in Table 2. Bionano Solve produced comparably fewer but longer contigs to RMAPPER but had 4 mis-assembled contigs. In addition, it took approximately 2937 CPU hours (55 hours of wall time using 60 CPUs in parallel) and peak memory of 94.8 GB. It is also worth noting that Bionano Solve performs an elaborate scaffolding and stitching of contigs, which explains the relatively few number of contigs but higher mis-assembly rate. The scaffolding and stitching cannot be decoupled from the assembly since Bionano only distributed a single executable that runs both. The source code is not publicly available.

In summary, the Valouev assembler did not scale to the human genome, RMAPPER2.0 produced slightly longer contigs than RMAPPER, Bionano Solve produced the longest contigs but covered 93.8% of the genome and had 4 mis-assembled contigs. In addition, RMAPPER2.0 has the highest genome fraction, which is 96.7%. Lastly, RMAPPER and RMAPPER2.0 was 242 and 132 times faster than Solve, respectively, and used 5 times less memory.

5.4 Performance on Climbing Perch

The results on the climbing perch (*Anabas Testudineus*) Rmap dataset are shown in Table 3. For this experiment we extracted bi-labels with $k = 6$ and $D = 15\,000$ and used error tolerance parameter setting $t_f = 1500$ and $t_\ell = 2000$. RMAPPER took 7.5 hours and peak memory of 9.7 GB to assemble the data whereas RMAPPER 2.0 took 14.9 hours and 18.77 GB of peak memory. RMAPPER produced 1848 contigs whereas RMAPPER 2.0 produced 2489 contigs. The maximum size unitig produced by RMAPPER and RMAPPER 2.0 was 217 and 294 fragments in length, respectively. Lastly, RMAPPER achieved a genome fraction of 78.2%, while RMAPPER 2.0 was able to cover 87.6% of the genome. Both RMAPPER and RMAPPER2.0 produced zero mis-assemblies.

The Valouev assembler did not halt on this dataset after 360 CPU days so we do not report any results. Solve was also unable to assemble this genome as it halted with a fatal error message after 156 CPU days and using a peak memory of 16 GB. In summary, RMAPPER was only method able to assemble this genome. Although RMAPPER and RMAPPER2.0 both successfully assembled the genome with zero mis-assemblies, RMAPPER2.0 produced slightly longer contigs than RMAPPER and achieved a higher genome coverage.

Table 3 Assembly results for the Rmap dataset of the climbing perch genome generated for the Vertebrate Genomes Project, which consists of 3 121 480 Rmaps with mean of 28 fragments. The restriction enzyme used in the experiment is BspQI. See Table 1 for a description of the assembly statistics and notation. As described in the text, RMAPPER2.0 extracts bi-labels from Rmaps in both forward and reverse directions.

Assembler	Run time	Peak Memory	No. of contigs	Max	Mean	GF(%)	MA
RMAPPER	7.5 h	9.7	1848	217 (1.6 Mbp)	52 (0.4 Mbp)	78.2	0
RMAPPER2.0	14.9 h	18.8	2489	294 (2.4 Mbp)	65 (0.6 Mbp)	87.6	0

6 Conclusion and Future Work

Assembly of Rmap data is a fundamental problem in optical mapping that still remains in a nascent stage – as prior to this work, there was only a single other non-proprietary assembler. In this paper, we formulate and describe the first Eulerian approach for Rmap assembly by redefining the de Brujn graph to adapt it to Rmap data. We accomplish this by extending the definition of a bi-label introduced in the context of the paired-end de Bruijn graph by Medvedev et al. [14]. We refer to our modified de Bruijn graph as the bi-labelled de Bruijn graph and demonstrate how to efficiently build and store it using a two-tiered orthogonal range search data-structure.

We implement our approach and show its performance on multiple simulated and real datasets. Our experimental results show the only non-proprietary method (i.e. by Valouev et al. [29]) is unable to scale to the human genome, and that our method is at least 130 times faster than Bionano Solve and its memory usage is less than 20% of the memory usage of Bionano Solve. An important note about the comparison of the assemblers is that RMAPPER has a very simple traversal algorithm and does not use any sort of scaffolding. This is due to the fact that the main contribution of this work is formulating and solving the assembly of Rmaps. Bionano Solve has a scaffolding algorithm that cannot be decoupled from the assembly step since only an executable is available. Thus, the results really compare RMAPPER's unitigs with Solve's scaffolds, and RMAPPER are still comparable. This work does open the door for improving Rmap assembly by employing more involved graph traversal and/or adapting methods designed for scaffolding and stiching sequence contigs using optical mapping data [19, 25].

References

1 Thomas S. Anantharaman, Bud Mishra, and David C. Schwartz. Genomics via optical mapping iii: Contiging genomic DNA and variations (extended abstract). In *ISMB-99*, pages 18–27. AAAI Press, 1997.

2 Anton Bankevich et al. SPAdes: A New Genome Assembly Algorithm and its Applications to Single-Cell Sequencing. *J. Comput. Biol.*, 19(5):455–477, 2012.

3 Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

4 Srikar Chamala et al. Assembly and validation of the genome of the nonmodel basal angiosperm *amborella*. *Science*, 342(6165):1516–1517, 2013.

5 Ping Chen, Xinyun Jing, Jian Ren, Han Cao, Pei Hao, and Xuan Li. Modelling BioNano optical data and simulation study of genome map assembly. *Bioinformatics*, 34(23):3966–3974, 2018.

6 Deanna M. Church et al. Lineage-specific biology revealed by a finished genome assembly of the mouse. *PLoS Biol.*, 7(5):e1000112+, 2009.

7 Yang Dong et al. Sequencing and automated whole-genome optical mapping of the genome of a domestic goat (*Capra hircus*). *Nat. Biotechnol.*, 31:135–141, 2013.

8 Xian Fan, Jie Xu, and Luay Nakhleh. Detecting large indels using optical map data. In *Proc. of RECOMB-CG*, pages 108–127, 2018.

9 Ganeshkumar Ganapathy et al. *De novo* high-coverage sequencing and annotated assemblies of the budgerigar genome. *GigaScience*, 3:11, 2014.

10 Alexey Gurevich, Vladislav Saveliev, Nikolay Vyahhi, and Glenn Tesler. QUAST: quality assessment tool for genome assemblies. *Bioinformatics*, 29(8):1072–1075, 2013.

11 Ramana M. Idury and Michael S. Waterman. A new algorithm forDNA sequence assembly. *J. Comput. Biol*, 2(2):291–306, 1995.

12 Le Li et al. OMSV enables accurate and comprehensive identification of large structural variations from nanochannel-based single-molecule optical maps. *Genome Biol.*, 18(1):230, 2017.

13 Menglu Li et al. Towards a more accurate error model for BioNano optical maps. In *Proc of ISBRA*, pages 67–79, 2016.

14 Paul Medvedev, Son Pham, Mark Chaisson, Glenn Tesler, and Pavel Pevzner. Paired de Bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers. *J. Comput. Biol*, 18, 2011.

15 Giles Miclotte, Stéphane Plaisance, Stephane Rombauts, Yves Van de Peer, Pieter Audenaert, et al. OMSim: a simulator for optical map data. *Bioinformatics*, pages 2740–2742, 2017.

16 Martin D. Muggli, Simon J. Puglisi, Roy Ronen, and Christina Boucher. Misassembly detection using paired-end sequence reads and optical mapping data. *Bioinformatics*, 31(12):i80–i88, 2015.

17 Kingshuk Mukherjee, Bahar Alipanahi, Tamer Kahveci, Leena Salmela, and Christina Boucher. Aligning optical maps to de Bruijn graphs. *Bioinformatics*, 35(18):3250–3256, 2019.

18 Kingshuk Mukherjee, Darshan Washimkar, Martin D. Muggli, Leena Salmela, and Christina Boucher. Error correcting optical mapping data. *GigaScience*, 7, 2018.

19 Weihua Pan, Tao Jiang, and Stefano Lonardi. OMGS: optical map-based genome scaffolding. *J. Comput. Biol*, 27(4):519–533, 2020.

20 Weihua Pan and Stefano Lonardi. Accurate detection of chimeric contigs via BioNano optical maps. *Bioinformatics*, 35(10):1760–1762, 2018.

21 Yu Peng, Henry CM Leung, Siu-Ming Yiu, and Francis YL Chin. IDBA-UD: A *de novo* assembler for single-cell and metagenomic sequencing data with highly uneven depth. *Bioinformatics*, 28(11):1420–1428, 2012.

22 Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proc. Natl. Acad. Sci.*, 98(17):9748–9753, 2001.

23 Susan Reslewic et al. Whole-genome shotgun optical mapping of *Rhodospirillum Rubrum*. *Appl. Environ. Microbiol.*, 71(9):5511–5522, 2005.

24 David C. Schwartz, Xiaojun Li, Luis I Hernandez, Satyadarshan P. Ramnarain, Edward J. Huff, and Yu-Ker Wang. Ordered restriction maps of *saccharomyces cerevisiae* chromosomes constructed by optical mapping. *Science*, 262:110–114, 1993.

25 Jennifer M. Shelton, Michelle C. Coleman, Nic Herndon, Nanyan Lu, Ernest T. Lam, Thomas Anantharaman, Palak Sheth, and Susan J. Brown. Tools and pipelines for BioNano data: molecule assembly pipeline and fasta super scaffolding tool. *BMC Genomics*, 16(1):734, 2015.

26 Jared T. Simpson, Kim Wong, Shaun D. Jackman, Jacqueline E. Schein, Steven J. M. Jones, and Inanç Birol. ABySS: a parallel assembler for short read sequence data. *Genome Res.*, 19(6):1117–1123, 2009.

27 Brian Teague et al. High-resolution human genome structure by single-molecule analysis. *Proc. Natl. Acad. Sci.*, 107(24):10848–10853, 2010.

28 Anton Valouev et al. Alignment of optical maps. *J. Comp. Biol.*, 13(2):442–462, 2006.

29 Anton Valouev, David C. Schwartz, Shiguo Zhou, and Michael S. Waterman. An algorithm for assembly of ordered restriction maps from single dna molecules. *Proc. Natl. Acad. Sci.*, 103(43):15770–15775, 2006.

30 Daniel R. Zerbino and Ewan Birney. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res.*, 18(5):821–829, 2008.

31 Shiguo Zhou et al. A whole-genome shotgun optical map of *Yersinia pestis* strain KIM. *Appl. Environ. Microbiol.*, 68(12):6321–6331, 2002.

32 Shiguo Zhou et al. Shotgun optical mapping of the entire *leishmania major* Friedlin genome. *Mol. Biochem. Parasitol.*, 138(1):97–106, 2004.

33 Shiguo Zhou et al. Validation of rice genome sequence by optical mapping. *BMC Genom.*, 8(1):278, 2007.

34 Shiguo Zhou et al. A Single Molecule Scaffold for the Maize Genome. *PLoS Genetics*, 5:e1000711, 2009.

Economic Genome Assembly from Low Coverage Illumina and Nanopore Data

Thomas Gatter
Bioinformatics Group, Department of Computer Science, University of Leipzig, Germany
Interdisciplinary Center of Bioinformatics, University of Leipzig, Germany
thomas@bioinf.uni-leipzig.de

Sarah von Löhneysen
Bioinformatics Group, Department of Computer Science, University of Leipzig, Germany
Interdisciplinary Center of Bioinformatics, University of Leipzig, Germany
lsarah@bioinf.uni-leipzig.de

Polina Drozdova
Institute of Biology, Irkutsk State University, Russia
drozdovapb@gmail.com

Tom Hartmann
Bioinformatics Group, Department of Computer Science, University of Leipzig, Germany
Interdisciplinary Center of Bioinformatics, University of Leipzig, Germany
tom@bioinf.uni-leipzig.de

Peter F. Stadler
Bioinformatics Group, Department of Computer Science, University of Leipzig, Germany
Interdisciplinary Center of Bioinformatics, University of Leipzig, Germany
Max-Planck-Institute for Mathematics in the Sciences, Leipzig, Germany
Institut for Theoretical Chemistry, University of Vienna, Austria
Facultad de Ciencias, Universidad National de Colombia, Bogotá, Colombia
Santa Fe Institute, NM, USA
studla@bioinf.uni-leipzig.de

Abstract

Ongoing developments in genome sequencing have caused a fundamental paradigm shift in the field in recent years. With ever lower sequencing costs, projects are no longer limited by available raw data, but rather by computational demands. The high complexity of eukaryotic genomes in concordance with increasing data sizes creates unique demands on methods to assemble full genomes. We describe a new approach to assemble genomes from a combination of low-coverage short and long reads. `LazyB` starts from a bipartite overlap graph between long reads and restrictively filtered short-read unitigs, which are then reduced to a long-read overlap graph G. Instead of the more conventional approach of removing tips, bubbles, and other local features, `LazyB` stepwisely extracts subgraphs whose global properties approach a disjoint union of paths. First, a consistently oriented subgraph is extracted, which in a second step is reduced to a directed acyclic graph. In the next step, properties of proper interval graphs are used to extract contigs as maximum weight paths. These are translated into genomic sequences only in the final step. A prototype implementation of `LazyB`, entirely written in python, not only yields significantly more accurate assemblies of the yeast and fruit fly genomes compared to state-of-the-art pipelines but also requires much less computational effort. Our findings demonstrate a new low-cost method that enables the assembly of even large genomes with low computational effort.

2012 ACM Subject Classification Theory of computation → Discrete optimization; Applied computing → Computational genomics

Keywords and phrases Nanopore sequencing, Illumina sequencing, genome assembly, spanning tree, unitigs, anchors

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.10

Supplementary Material The `LazyB` prototype is available at `https://github.com/TGatter/LazyB`.

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti; Article No. 10; pp. 10:1–10:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding *Thomas Gatter*: DFG STA 850/19-2 within SPP 1738; German Federal Ministery of Education and Research 031A538A, de.NBI-RBC.
Polina Drozdova: RSF / Helmholtz Association 18-44-06201; German Academic Exchange Service (DAAD).
Tom Hartmann: German Federal Ministry of Education and Research (BMBF 031L0164C).

1 Introduction

The assembly of genomic sequences from high throughput sequencing data has turned out to be a difficult computational problem in practice. Recent approaches combine cheap short-read data (typically using Illumina technology) with long reads produced by PacBio or Nanopore technologies. Although the short-read data are highly accurate and comparably cheap to produce, they are insufficient even at (very) high coverage due to repetitive elements. Long-read data, on the other hand, are comparably expensive and have much higher error rates. HiFi PacBio reads derived from repeat sequencing of circularized elements rival short read accuracy but at vastly increased costs.

Several assembly techniques have been developed recently for *de novo* assembly of large genomes from high-coverage (50× or greater) PacBio or Nanopore reads. Recent state-of-the-art methods employ a hybrid assembly strategy using Illumina reads to correct errors in the longer PacBio reads prior to assembly. For instance, the 32 Gb axolotl genome was produced in this manner [26].

Traditional assembly strategies can be classified into two general categories [21]. The Overlap-layout-consensus (OLC) assembly model attempts to find all pairwise matches between reads, using sequence similarity as a metric for overlaps. A general layout is constructed and post-processed in various ways. Most notably, overlaps can be transformed into assembly graphs such as string graphs. This method is flexible to read length and can be adapted to the diverse error models of different sequencing technologies. However, finding all overlaps is very expensive, especially for increasing read sizes.

In de Bruijn graph based strategies, reads are deconstructed to fixed length k-mers, representing nodes with edges between them for each $k-1$ overlap. Ideally, a de Bruijn graph represents exactly one Eulerian path per chromosome, although this property is generally violated in practice even by light sequencing errors. With the help of specialized hashing strategies k-mers can be efficiently stored and constructed. Thus, de Bruijn graphs require much less memory than OLC strategies. An overall speed up can be attributed to the absence of an all-*vs*-all comparison step. However, as k has to be chosen smaller than read size, contiguity information is lost. With increasing error rates in reads, de Bruijn graphs tend to become less useful, as k-mers become also less accurate.

Long read only and hybrid assembly strategies also largely align to these two categories, although some more unique methods have emerged over the years. `Canu` [18] and `Falcon` [5] implement classic OLC, albeit both error-correct long reads before creating a string graph. MinHash filters can significantly reduce the costs of comparisons, but overall complexity remains high. `Wtdbg2` [28] also follows OLC, but utilizes de Bruijn like graphs based on sparse k-mer mapping for comparison. It avoids all-*vs*-all mapping by matching reads that share k-mers under the assumption that even under high error rates correct pairs share more k-mer than those with spurious matches. `Shasta` [29] implements a full de Bruijn graph strategy by transforming k-mers into a run-length encoding that is more robust to sequencing errors in long reads. Newer versions of `Canu` also implement a similar encoding [27].

Classic de Bruijn methods have been adapted to combine both long and short reads into a hybrid assembly. Long reads can serve as "bridging elements" in the same way as mate pairs to resolve paths in (short read) assembly graphs [2, 33].

Under the assumption that short-read assemblies are cheap and reliable, various workflows have been proposed to integrate both kinds of data also for OLC like approaches. As a general goal, these programs aim to avoid the costly all-*vs*-all comparison to create the assembly graph by various heuristics. `MaSuRCA` [36] attempts to join both long and short reads into longer super-reads by chaining unique k-mers, as such creating fewer reads to compare for overlap. `WENGAN` [7] first creates full short-read contigs that are then scaffolded by synthetic mate pairs generated out of the long reads. `Flye` [17], even more uniquely, assembles intentionally erroneous contigs that are concatenated to a common sequence. Self-mapping then reveals repeats that can be resolved much like in a traditional assembly graph. In `HASLR` [11] an assembly graph like structure is defined combing both short and long reads. Short reads are assembled into contigs that, after k-mer filtering to remove repeats, are aligned to long reads. In the resulting backbone graph, short-read contigs serve as nodes that are connected by an edge if they map onto the same long read. While different to e.g. string graphs, standard tip and bubble removal algorithms are applied to remove noise. Contigs are extracted as paths. `TULIP` [13] implements a very similar strategy, however, does not assemble short reads into full contigs. Instead, the gaps between mate-pairs are closed if possible with sufficiently rare k-mers, resulting in relative short but unique seeds that serve in the same capacity. In both cases, consensus construction of the resulting sequence is trivial. Edges define fixed regions on groups of long reads that can be locally aligned for each edge along a path.

`DBG2OLC` [35] is methodologically most closely related to `LazyB`, however, both approaches differ in various key features (which becomes obvious over the course of this paper). `DBG2OLC` assembles short reads to full contigs with the advise to avoid repeat resolving techniques such as gap closing or scaffolding as they introduce too many errors. Contigs are then aligned against long reads. Each long read implies a neighborhood of contigs. Mappings are corrected prior to graph construction via consistency checks over all neighborhoods for each contig, i.e., contigs are required to map in the same order on all long reads. This technique can help to remove both spuriously matched contigs and chimeric long reads, but requires adequate coverage to allow for effective voting. Notably, here, long reads serve as nodes, with edges representing contigs mapping to both. Nodes that map a subset of contigs of another node are removed as they are redundant. The resulting graph can be error corrected by classic tip and bubble removal, after which paths are extracted as contigs, following the edge with the best overlap at each step.

`LazyB` implements an alternative approach to assembling genomes from a combination of long-read and short-read data. We avoid the expensive direct all-*vs*-all comparison of the error-prone long-read data, the difficult mapping of individual short reads against the long reads, and the conventional techniques to error-correct de Bruijn or string graphs. As we shall see, this is not only possible but also adds the benefit of producing rather good assemblies with surprisingly low requirements on the coverage of both short and long reads. Our methods lends itself in particular to the exploratory assembly of large numbers of species.

2 Strategy

Instead of a "total data" approach, we identify "anchors" that are nearly guaranteed to be correct and use an, overall, greedy-like workflow to obtain very large long-read contigs. To this end, the initial overlap graph is oriented and then edited in several steps to graph classes approaching the desired union of paths. The strategy of `LazyB` is outlined in Fig. 1.

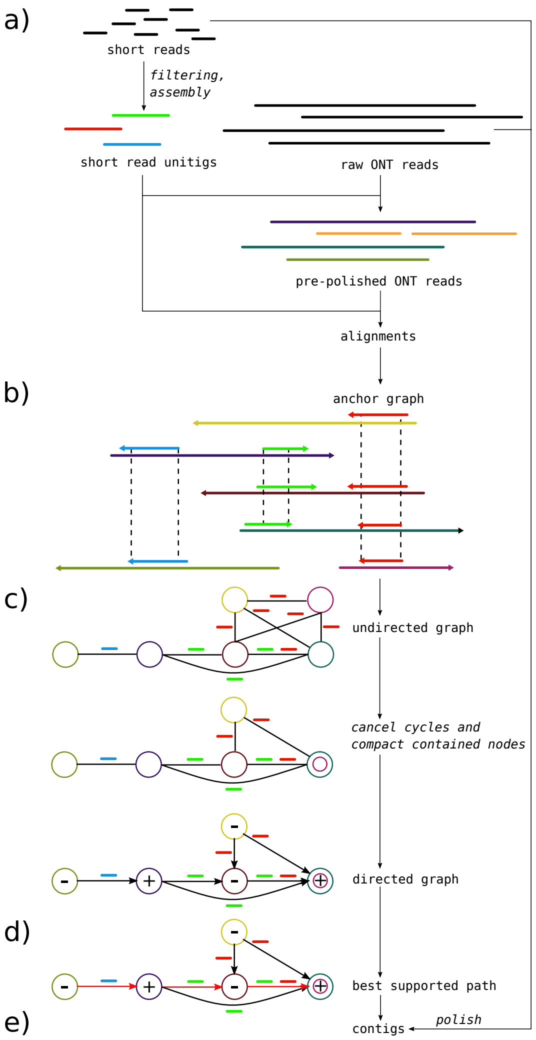


Figure 1 Overview of the `LazyB` assembly pipeline. a) Short Illumina reads are filtered to represent only near unique k-mers and subsequently assembled into unambiguous unitigs. Long Nanopore reads (ONT) can be optionally scrubbed to include only regions consistent to at least one other read. For larger data sets scrubbing can be handled on subsets efficiently. Mapping unitigs against Nanopore reads yields unique "anchors" between them b). An undirected graph c) is created by adding Nanopore reads as nodes and edges between all pairs of reads sharing an "anchor". Each edge is assigned a *relative orientation*, depending on whether the "anchor" maps in the same direction on both Nanopore reads. Cycles with a contradiction in orientation have to be removed before choosing a node at random and directing the graph based on its orientation. As Nanopore reads that are fully contained within another do not yield additional data, they can be collapsed. Contigs are extracted as maximally supported paths for each connected component d). Support in this context is defined by the number of consistent overlaps transitive to each edge. Final contigs e) can be optionally polished using established tools.

The key idea to obtain the overlap graph is to start from a collection $\mathcal{S} := \{s_i\}$ of pre-assembled, high-quality sequences that are unique in the genome. These serve as "anchors" to determine overlaps among the long reads $\mathcal{R} := \{r_j\}$. In practice, $\mathcal{S}$ can be obtained by assembling Illumina data with fairly low coverage to the level of unitigs only. The total genomic coverage of $\mathcal{S}$ only needs to be large enough to provide anchors between overlapping long reads, and it is rigorously filtered to be devoid of repetitive and highly similar sequences.

Mapping a short read $s \in \mathcal{S}$ against the set $\mathcal{R}$ of long reads implies (candidate) overlaps $r_1 - r_2$ between two long reads (as well as their relative orientation) whenever an s maps to both r_1 and r_2. Thus we obtain a directed overlap graph G of the long reads without an all-*vs*-all comparison of the long reads.

A series of linear-time filtering and reduction algorithms then prunes first the underlying undirected overlap graph and then the directed version of the reduced graph. Its connected components are reduced to near-optimal directed acyclic graphs (DAGs) from which contigs are extracted as best-supported paths. In the following sections we describe the individual steps in detail. In comparison to `DBG2OLC` we avoid global corrections of short-read mappings, but instead rely on the accuracy of assembled unitigs and a series of local corrections. For this, we utilize previously unreported properties of the class of alignment graphs used by both tools. This allows `LazyB` to operate reliably even on very low coverage. Variations of the dataset dependent assembly options have little impact on the outcome. In contrast of complicated setup of options required for tools such as `DBG2OLC`, `LazyB` comes with robust defaults.

3 Theory and Methods

3.1 Preprocessing

A known complication of both PacBio and Nanopore technologies are chimeric reads formed by the artificial joining of disconnected parts of the genome [23] that may cause misassemblies [34]. Current methods dealing with this issue heavily rely on raw coverage [22] and hence are of little use for our goal of a low-coverage assembler. In addition, start- and end-regions of reads are known to be particularly error-prone. We pre-filter low quality regions, but only consider otherwise problematic reads later at the level of the overlap graph.

Short-read (Illumina) data are preprocessed by adapter clipping and trimming. A set $\mathcal{S}$ of high quality fragments is obtained from a restricted assembly of the short-read data. The conventional use case of assembly pipelines aims to find a minimal set of contigs in trade-off to both correctness and completeness. For our purposes, however, completeness is of little importance and fragmented contigs are not detrimental to our workflow, as long as their lengths stay above a statistical threshold. Instead, correctness and uniqueness are crucial. We therefore employ three filtering steps: (1) Using a k-mer profile, we remove all k-mers that are much more abundant than the expected coverage since these are likely part of repetitive sequences. This process can be fully automated (see Appendix B).(2) In order to avoid ambiguities only branch-free paths are extracted from the assembly graph. Moreover, a minimal path length is required for secure anchors. The de Bruijn based assembler `ABySS` [30] allows to assemble up to unitig stage, implementing this goal. Since repeats in general lead to branch-points in the de Bruijn graph, repetitive sequences are strongly depleted in unitigs. While in theory, every such assembly requires a fine tuned k-mer size, a well known factor to be influential on assembly quality, we found overall results to be mostly invariant of this parameter. To test this, we systematically varied the k-mer-size for `ABySS`. Nevertheless, we found little to no effect on the results of `LazyB` (Fig. 2). As assembly stops at unitigs, error rates and genome coverage stay within a narrow range as long as the unitigs are long enough. (3) Finally, the set $\mathcal{R}$ of long reads is mapped against the unitig set. At present we use `minimap2` [20] for this purpose. Regions or whole unitigs significantly exceeding the expected coverage are removed from $\mathcal{S}$ because they most likely are repetitive or at least belong to families of very similar sequences such as multi-gene families. Please note that all repetitive elements connected to a unique region within a single long read may still be correctly assembled (see Appendix C).

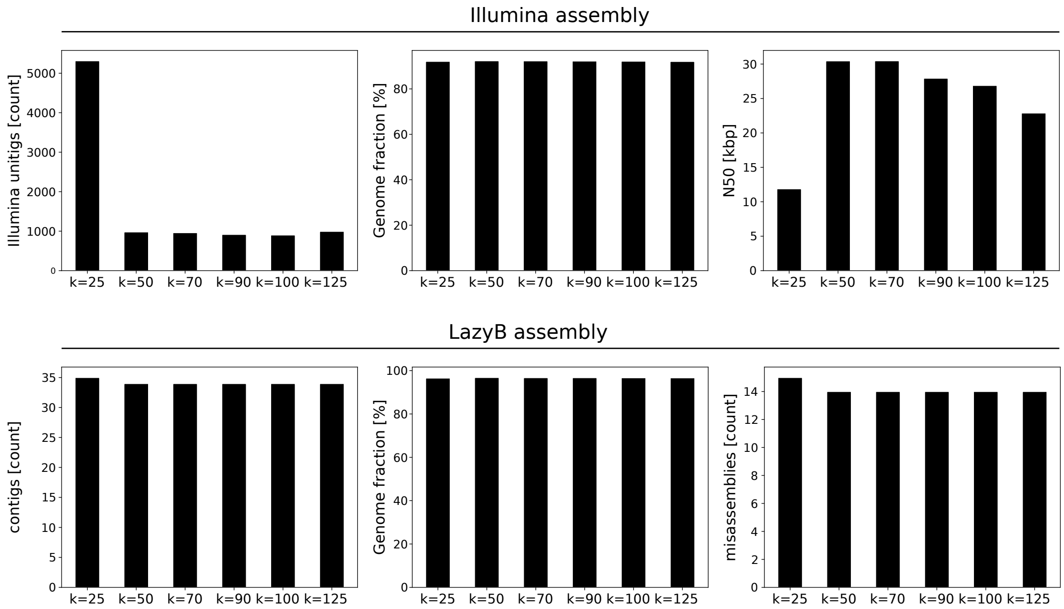


Figure 2 Assembly statistics as a function of the k-mer size used to construct unitigs from the short-read data for yeast. Top: Illumina unitigs (left: number of unitigs; middle: fraction of the reference genome covered; right: N50 values); bottom: final LazyB assembly at ~11× long reads (left: number of unitigs; middle: fraction of the reference genome covered; right: number of mis-assemblies).

3.2 Overlap Graph for Long Reads

As a result we obtain a set of *significant matches* $\mathcal{V} := \{(s,r) \in \mathcal{S} \times \mathcal{R} \mid \delta(s,r) \geq \delta_*\}$ whose matching score $\delta(s,r)$ exceeds a user-defined threshold δ_*. The *long-read overlap graph* G has the vertex set $\mathcal{R}$. Conceptually, two long reads overlap, i.e., there should be an undirected edge $r_1r_2 \in E(G)$ if and only if there is an $s \in \mathcal{S}$ such that $(s,r_1) \in \mathcal{V}$ and $(s,r_2) \in \mathcal{V}$. In practice, however, we employ a more restrictive procedure:

For distinct long reads $r_1, r_2 \in \mathcal{R}$ with $(s,r_1),(s,r_2) \in \mathcal{V}$ the sequence intervals on s that match intervals on r_1 and r_2 are denoted with $[i,j]$ and $[k,l]$, respectively. The intersection $[i,j] \cap [k,l]$ is the interval $[\max\{i,k\}, \min\{j,l\}]$ if $k \leq j$ and the empty interval otherwise. Note that if $[i,j] \cap [k,l]$ is not empty, then it corresponds to a direct match of r_1 and r_2. The expected bit score for the overlap is estimated as

$$\omega(s,r_1,r_2) := \begin{cases} 0 & \text{if } [i,j] \cap [k,l] = \varnothing; \\ \frac{1}{2}(\min\{j,l\} - \max\{i,k\} + 1)\left(\frac{\delta(s,r_1)}{(j-i+1)} + \frac{\delta(s,r_2)}{(l-k+1)}\right) & \text{otherwise.} \end{cases} \tag{1}$$

For a given edge $r_1r_2 \in E(G)$ there may be multiple significant matches, mediated by a set of unitigs $\mathcal{S}_{r_1r_2} := \{s \in \mathcal{S} \mid (s,r_1),(s,r_2) \in \mathcal{V}\}$. In ideal data they are all consistent with respect to orientation and co-linear location. In real data, however, this may not be the case.

For each significant match $(s,r) \in \mathcal{V}$ we define the *relative orientation* $\theta(s,r) \in \{+1,-1\}$ of the reading directions of the short-read scaffold s relative to the long read r. The relative reading direction of the long reads (as suggested by s) is thus $\theta_s(r_1,r_2) = \theta(s,r_1) \cdot \theta(s,r_2)$.

The position of a significant match (s,r) defined on the unitig s on interval $[i,j]$ corresponds to an interval $[i',j']$ on the long read r that is determined by the alignment of s to r. Due to the large number of randomly distributed InDels in the Nanopore data,

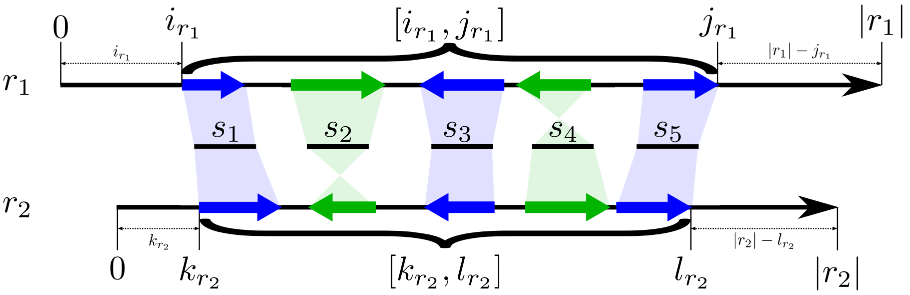


Figure 3 Construction of the overlap of two long reads r_1 and r_2 (long black arrows) from all unitigs $\mathcal{S}_{r_1 r_2} := \{s_1, ..., s_5\}$ (short black bars) that match to both r_1 and r_2. A significant match (s, r) of $s \in \mathcal{S}_{r_1 r_2}$ on $r \in \{r_1, r_2\}$ is illustrated by blue and green thick arrows on r. The relative orientation of (s, r) is indicated by the direction of its arrow, that is, $\theta(s, r) = +1$ (resp. $\theta(s, r) = -1$) if its arrow points to the right (resp. left). The subsets $\mathcal{S}^1_{r_1 r_2} := \{s_1, s_3, s_5\}$ (unitigs with blue significant matches) and $\mathcal{S}^2_{r_1 r_2} := \{s_2, s_4\}$ (unitigs with green significant matches) of $\mathcal{S}_{r_1 r_2}$ are both inclusion-maximal and consists of pairwise consistent unitigs. The set $\mathcal{S}^1_{r_1 r_2}$ maximizes $\Omega(r_1, r_2)$ and thus determines the overlap. It implies $\theta(r_1, r_2) = +1$. Moreover, i_{r_1} (resp. j_{r_1}) is the minimal (resp. maximal) coordinate of significant matches of unitigs from $\mathcal{S}^1_{r_1 r_2}$ on r_1. The corresponding coordinates on r_2 are k_{r_2} and l_{r_2}, respectively. The spanning intervals $[i_{r_1}, j_{r_1}]$ and $[k_{r_2}, l_{r_2}]$ define the overlap of r_1 and r_2. In this example we have $i_{r_1} > k_{r_2}$ and $|r_1| - j_{r_1} > |r_2| - l_{r_2}$, implying that r_2 extends r_1 neither to the left or right and thus, edge $r_1 r_2$ is contracted in G.

the usual dynamic programming alignment strategies fail to produce accurate alignments. This is also the case for `minimap2` [20], our preliminary choice, as it only chains short, high quality matches into larger intervals. Although more accurate alignments would of course improve the local error rate of the final assembled sequence, we expect very little impact on the overall assembly that is not effected in large by small local errors. We therefore record only the matching intervals and use a coordinate transformation τ_r that estimates the position $\tau_r(h) \in [i', j']$ for some $h \in [i, j]$ by linear interpolation:

$$\tau_r(h) := \begin{cases} j' - (j - h)\frac{j'-i'+1}{j-i+1} & \text{if } j - h \le h - i; \\ i' + (h - i)\frac{j'-i'+1}{j-i+1} & \text{if } j - h > h - i. \end{cases} \tag{2}$$

The values of $\tau_r(h)$ are rounded to integers and used to determine intersections of matches. We write $[i, j]_r := [\tau_r(i), \tau_r(j)]$ for the interval on r corresponding to an interval $[i, j]$ of s.

▶ **Definition 1.** *Two unitigs s, s' in $\mathcal{S}_{r_1 r_2}$ are* consistent *if (i) $\theta_s(r_1, r_2) = \theta_{s'}(r_1, r_2)$, (ii) the relative order of $[i^s, j^s]_{r_1}$, $[k^{s'}, l^{s'}]_{r_1}$ on r_1 and $[i^s, j^s]_{r_2}$, $[k^{s'}, l^{s'}]_{r_2}$ on r_2 is the same.*

For distinct long reads $r_1, r_2 \in \mathcal{R}$, Definition 1 enables us to determine $m \ge 1$ subsets $\mathcal{S}^1_{r_1 r_2}, ..., \mathcal{S}^m_{r_1 r_2}$ of $\mathcal{S}_{r_1 r_2}$ such that each is maximal with respect to inclusion and contains only unitigs that are pairwise consistent with respect to r_1 and r_2. In addition, we may require that the difference between the distances of consecutive corresponding intervals on r_1 and r_2, respectively, is sufficiently similar. Computing the set $\mathcal{S} \in \{\mathcal{S}^1_{r_1 r_2}, ..., \mathcal{S}^m_{r_1 r_2}\}$ that maximizes the total bit score $\sum_{s \in \mathcal{S}} \omega(s, r_1, r_2)$ amounts to a chaining problem that can be solved in quadratic time by dynamic programming [25]. An edge $r_1 r_2$ is inserted into G if the optimal total bit score $\Omega(r_1, r_2) := \sum_{s \in \mathcal{S}} \omega(s, r_1, r_2)$ exceeds a user-defined threshold. The *signature* $\theta(r_1, r_2)$ of the edge $r_1 r_2 \in E(G)$ is the common value $\theta_s(r_1, r_2)$ for all $s \in \mathcal{S}$.

For each edge $r_1 r_2 \in E(G)$ we determine $s, s' \in \mathcal{S}$ such that $\tau_{r_1}(i^s)$ is the minimal and $\tau_{r_1}(j^{s'})$ is the maximal coordinate of the matching intervals on r_1. Hence, the interval $[i^s, j^{s'}]_{r_1}$ spans all matching intervals on r_1. The corresponding pair of coordinates,

$\tau_{r_2}(k^s)$ and $\tau_{r_2}(l^{s'})$, spans the matching intervals on r_2. In particular, the interval $[k^s, l^{s'}]_{r_2}$ (resp. $[l^{s'}, k^s]_{r_2}$) spans both matching intervals on r_2 if $\theta(r_1, r_2) = 1$ (resp. $\theta(r_1, r_2) = -1$). For the sake of a clear notation, let $[i_{r_1}, j_{r_1}] := [i^s, j^{s'}]_{r_1}$ and $[k_{r_2}, l_{r_2}]$ be the "spanning" interval on r_2, i.e., either $[k_{r_2}, l_{r_2}] := [k^s, l^{s'}]_{r_2}$ or $[k_{r_2}, l_{r_2}] := [l^{s'}, k^s]_{r_2}$. Intervals $[i_{r_1}, j_{r_1}]$ and $[k_{r_2}, l_{r_2}]$ specify the known overlapping regions between r_1 and r_2, see also Fig. 3 for an illustration. If $\theta(r_1, r_2) = +1$ then r_1 *extends* r_2 *to the left* if $i_{r_1} > k_{r_2}$ and *to the right* if $|r_1| - j_{r_1} > |r_2| - l_{r_2}$. For $\theta(r_1, r_2) = -1$ the corresponding conditions are $i_{r_1} > |r_2| - k_{r_2}$ and $|r_1| - j_{r_1} > l_{r_2}$, respectively. If r_1 does not extend r_2 to either side then r_1 is completely contained in r_2 and does not contribute to the assembly. If r_1 extends r_2 on both sides, r_2 is fully contained, respectively. In both cases we contract the edge between r_1 and r_2 in G. Otherwise, if r_1 extends r_2 to the left and r_2 extends r_1 to the right we record $r_1 \rightarrow r_2$ and accordingly, if r_2 extends r_1 to the left and r_1 extends r_2 to the right we note $r_1 \leftarrow r_2$.

The result of this construction is a long-read-overlap graph G whose vertices are the non-redundant long reads and whose edges r_1r_2 record (1) the relative orientation $\theta(r_1, r_2)$, (2) the bit score $\Omega(r_1, r_2)$, (3) the local direction of extension, and (4) the overlapping interval.

3.3 Consistent Orientation of Long Reads

For perfect data it is possible to consistently determine the reading direction of each read relative to the genome from which it derives. This is not necessarily the case in real-life data. The relative orientation of two reads is implicitly determined by the relative orientation of overlapping reads, i.e., by the signature $\theta(r_1, r_2)$ of the edge $r_1r_2 \in E(G)$. To formalize this idea we consider a subset $D \subseteq E(G)$ and define the *orientation* of D as $\theta(D) := \prod_{r_1r_2 \in D} \theta(r_1, r_2)$. For a disjoint union of two edge sets D and D' we therefore have $\theta(D \uplus D') = \theta(D')\theta(D)$ and, more generally, their symmetric different $D \oplus D'$ satisfies $\theta(D \oplus D') = \theta(D)\theta(D')$ since the edges in $D \cap D'$ appear twice in $\theta(D)\theta(D')$ and thus contribute a factor $(\pm 1)^2 = 1$.

▶ **Definition 2.** *Two vertices $r_1, r_2 \in V(G)$ are* orientable *if $\theta(P) = \theta(P')$ holds for any two paths P and P' connecting r_1 and r_2 in G. We say that G is* orientable *if all pairs of vertices in G are orientable.*

▶ **Lemma 3.** *G is orientable if and only if every cycle C in G satisfies $\theta(C) = 1$.*

Proof. Let r, r' be two vertices of G and write $\mathcal{C}(r, r')$ for the set of all cycles that contain r and r'. If $r = r'$ or $\mathcal{C}(r, r') = \emptyset$, then r and r' are orientable by definition. Now assume $r \neq r'$, $\mathcal{C}(r, r') \neq \emptyset$, and consider a cycle $C \in \mathcal{C}(r, r')$. Clearly, C can be split into two edge-disjoint path C_1 and C_2 both of which connect r and r'. If r and r' are orientable, then $\theta(C_1) = \theta(C_2)$ and thus $\theta(C) = 1$. If r and r' are not orientable, then there is a pair of path P_1 and P_2 connecting r and r' such that $\theta(P_1) = -\theta(P_2)$. Since $P_1 \oplus P_2 = \biguplus_{i=1}^{k} C_i$ is an edge-disjoint union of cycles C_i we have $-1 = \theta(P_1)\theta(P_2) = \prod_{i=1}^{k} \theta(C_i)$ and thus there is least one cycle C_i with $\theta(C_i) = -1$ in G. ◀

The practical importance of Lemma 3 is the implication that only a small set of cycles needs to be considered since every graph G with c connected components has a cycle basis comprising $|E| - |V| - c$ cycles. Particular cycles bases, known as *Kirchhoff bases*, are obtained from a spanning tree T of G as the set $\mathcal{B}$ of cycles C_e consisting of the edge $e \in E \setminus T$ and the unique path in T connecting the endpoints of e. Every cycle C of G can then be written as $C = \bigoplus_{e \in C \setminus T} C_e$, see e.g. [15].

▶ **Theorem 4.** *Let $\mathcal{B}$ be a cycle basis of G. The graph G is orientable if and only if $\theta(C) = 1$ for all $C \in \mathcal{B}$.*

Proof. The theorem follows from Lemma 3 and the fact that every cycle C in G can be written as an $\oplus$-sum of basis cycles, i.e., $\theta(C) = 1$ for every cycle in C if and only if $\theta(C') = 1$ for every basis cycle $C' \in \mathcal{B}$. ◀

Theorem 4 suggests the following, conservative heuristic to extract an orientable subgraph from G:

(1) Construct a maximum weight spanning tree T_G of G by using the Ω-scores as edge weights. Tree T_G can easily be obtained using, e.g., Kruskal's algorithm [19].

(2) Construct a Kirchhoff cycle basis $\mathcal{B}$ from T_G.

(3) For every cycle $C \in \mathcal{B}$, check whether $\theta(C) = -1$. If so, find the Ω-minimum weighted edge $\hat{e} \in C$ and remove it from $E(G)$ and (possibly) from T_G if $\hat{e} \in E(T_G)$. Observe that if $\hat{e} \notin E(T_G)$, then T_G stays unchanged. If $\hat{e} \in E(T_G)$, then the removal of $\hat{e}$ splits T_G into two connected components. We restrict G to the connected components of T_G.

This procedure yields a not necessarily connected subgraph G' and a spanning forest $T_G \cap E(G')$ for G'.

▶ **Lemma 5.** *Let G be an undirected graph and let G'' be a connected component of the residual graph G' produced by the heuristic steps (1)-(3). Then (i) G'' is orientable and (ii) $T_G \cap E(G'')$ is an Ω-maximal spanning tree of G''.*

Proof. Removal of an edge e from a spanning tree T of G partitions T into two components with vertex sets V_1 and V_2. Let $G_1 = G[V_1]$ and $G_2 = G[V_2]$ be the corresponding induced subgraphs of G. The cut in G induced by e is $E(G) \setminus (E(G_1) \cup E(G_2))$. Clearly $T_1 = T \cap E(G_1)$ and $T_2 = T \cap E(G_2)$ are spanning trees of G_1 and G_2, respectively. The restrictions $\mathcal{B}_1$ and $\mathcal{B}_2$ of the Kirchhoff basis $\mathcal{B}$ to cycles with non-tree edges $e \in E(G_1)$ or $e \in E(G_2)$ form a Kirchhoff basis of G_1 and G_2, respectively. Now consider G_1' is obtained from G_1 by removing a set of non-tree edges, then $\mathcal{B}_1'$ obtained from $\mathcal{B}_1$ by removing the cycles with these non-tree edges is a cycle basis of G_1' and T_1 is still a spanning tree of G_1. The Ω-weights of $T_1 = T \cap E(G_1)$ and $T_2 = T \cap E(G_2)$ must be maximal, since otherwise a heavier spanning tree T of G could be constructed by replacing T_1 or T_2 by a heavier spanning tree of G_1 or G_2. The arguments obviously extend to splitting G_i by cutting at an edge of T_i. Since the heuristic removes all non-tree edges e with $\theta(C_e) = -1$, Theorem 4 implies that each component G'' is orientable. When removing $e \in T$, the corresponding cut edges are removed, the discussion above applies and thus $T \cap E(G'')$ is Ω-maximal. ◀

From here on, we denote a connected component of G' again by G and write T_G for its maximum Ω-weight spanning tree, which by Lemma 5 is just the restriction of the initial spanning tree to G. We continue by defining an orientation φ for the long reads. To this end, we pick an arbitrary $r_* \in V(G)$ and set $\varphi(r_*) := +1$. For each $r \in V(G)$ we set $\varphi(r) := \prod_{e \in \mathrm{path}(r_*,r)} \theta(e)$, where path$(r_*, r)$ is the unique path connecting r_* and r in T_G. We can now define an equivalent graph $\tilde{G}$ with the same vertices and edges as G and orientations $\tilde{\theta}(e) = +1$ for $e \in T_G$ and $\tilde{\theta}(e) := \varphi(x_e)\varphi(y_e)$ for all non-tree edges $e = x_e y_e \notin T_G$. We note that the vertex orientations can be computed in $\mathcal{O}(|\mathcal{R}|)$ time along T_G. Since $\theta(C_e) = \tilde{\theta}(e)$ for every $C_e \in \mathcal{B}$, we can identify the non-orientable cycles in linear time.

3.4 Reduction to a DAG

We next make use of the direction of extension of long read r_1 and r_2 defined by the mutual overhangs in the case that $r_1 r_2$ is an edge in G. We write $\vec{G}$ for the directed version of a connected component G of the residual graph G' constructed above. For each edge $r_1 r_2 \in E(G)$ we create the corresponding edge $e \in E(\vec{G})$ as

$$e := \begin{cases} r_1 r_2 & \text{if } \varphi(r_1) = +1 \text{ and } r_1 \rightarrow r_2 \text{ or } \varphi(r_1) = -1 \text{ and } r_1 \leftarrow r_2; \\ r_2 r_1 & \text{if } \varphi(r_1) = +1 \text{ and } r_1 \leftarrow r_2 \text{ or } \varphi(r_1) = -1 \text{ and } r_1 \rightarrow r_2. \end{cases} \tag{3}$$

In perfect data, $\vec{G}$ is a directed interval graph. Recall that we have contracted edges corresponding to nested reads (i.e., intervals). Therefore, $\vec{G}$ is a proper interval graph or indifference graph. Thus there is an ordering $\prec$ of the vertices (long reads) that satisfies the *umbrella property* [12]: $r_1 \prec r_2 \prec r_3$ and $r_1 r_3 \in E(\vec{G})$ implies $r_1 r_2, r_2 r_3 \in E(\vec{G})$. A "normal interval representation" and a linear order $\prec$ of the reads, can be computed in $\mathcal{O}(|\mathcal{R}|)$ time [24]. Again, we cannot use these results directly due to the noise in the original overlap graph.

First we observe that $\vec{G}$ should be acyclic. Our processing so far, however, does not guarantee acyclicity since $\vec{G}$ still may contain some spurious edges due to unrecognized repetitive elements. The obvious remedy is to remove a (weight-)minimal set of directed edges. This Feedback Arc Set problem, however, is NP-complete, see [3] for a recent overview. We therefore resort to a heuristic that makes use of our expectations on the structure of $\vec{G}$: In general we expect multiple overlaps of correctly placed reads, i.e., r is expected to have several incoming edges from its predecessors and several outgoing edges exclusively to a small set of succeeding reads. In contrast, we expect incorrect edges to appear largely in isolation. This suggest to adapt Khan's topological sorting algorithm [14]. In its original version, it identifies a source u, i.e., a vertex with in-degree 0, appends it to the list W of ordered vertex and then deletes all its out-edges. It stops with "fail" when no source can be found before the sorting is complete, i.e., W does not contain all vertices of the given graph, indicating that a cycle has been encountered. In our setting we need to identify the best approximation to create a new source in this case. Denote by $N_+(W)$ denotes the out-neighborhood of the already sorted set W. The set $K := (V \smallsetminus W) \cap N_+(W)$ of not yet sorted out-neighbors of W are the candidates for the next source. For each $u \in K$ we distinguish incoming edges xu from $x \in W$, $x \in K$, and $x \in V \smallsetminus (W \cup K)$ and consider two cases:

(1) There is a $u \in K$ without an in-edge xu from some other $x \in K$. Then we choose among these the vertex $\hat{u}$ with the largest total Ω-weight incoming from W because $\hat{u}$ then overlaps with most of the previously sorted reads.

(2) If for each $u \in K$ there is an in-edge xu from some other $x \in K$, then the candidate set K forms a strongly connected digraph. In this case we choose the candidate $\hat{u} \in K$ with the largest difference of Ω-weights incoming from W and K, i.e., $\hat{u} := \arg\max_{u \in K} \sum_{w \in W} \Omega(w,u) - \sum_{k \in K \smallsetminus \{u\}} \Omega(k,u)$.

In either case we remove from $\vec{G}$ the edges incoming from $V \smallsetminus W$ into $\hat{u}$ and proceed. If multiple sources are available we always pick the one with largest Ω-weight incoming from W. As a consequence, incomparable paths in $\vec{G}$ are sorted contiguously. The result of the modified Kahn algorithm is a directed acyclic graph $\overrightarrow{G}$.

3.5 Golden Paths

For perfect data, $\overrightarrow{G}$ (and already $\vec{G}$) has a single source and a single sink vertex, corresponding to the left-most and right-most long reads r' and r'', respectively. Furthermore, every directed path connecting r' and r'' is a *golden path*, that is, a sequence of overlapping intervals that covers the entire chromosome. Even more stringently, every read $r \neq r', r''$ has at least one predecessor and at least one successor in $\overrightarrow{G}$. The acyclic graph $\overrightarrow{G}$ therefore has a unique topological sorting, i.e., its vertices are totally ordered. As before, we cannot expect that $\overrightarrow{G}$ has these properties for real-life data.

Ploidy in eukaryotes may constitute a valid exception to this assumption, as differences in chromosomes ideally also cause diverging structures. However, given the high error rate of long reads, low sequence variation can only be differentiated in very high coverage scenarios; these explicitly are not targeted by `LazyB`. High accuracy short read assemblies originating from different alleles thus can be expected to match equally well to the same long reads given their low quality. Therefore, also ploidy variation will normally be merged to a single consensus. Accordingly, we did not detect any mayor duplication issues in the human, fly, or yeast.

A transitive reduction H° of some directed graph H is a subgraph of H with as few edges as possible such that two vertices x and y are connected by a directed path in H° if and only if they are connected by a directed path in H. It is well-known that each acyclic digraph has a unique transitive reduction [1, Thm. 1]. This property enables us to call an edge e of an acyclic digraph H *redundant* if $e \notin E(H^\circ)$.

Consider a proper interval graph H, an induced subgraph F of H, and recall that H is an acyclic digraph. Since H satisfies the umbrella property, every redundant edge $uw \in E(H)$ is part of some triangle. We also observe that F has a unique topological sorting and its *triangle reduction* $F^\triangle$, obtained by removing all edges $uw \in E(F)$ for which there is a vertex v with $uv, vw \in E(F)$, is a path. In fact, $F^\triangle$ is an induced path in the triangle reduction $H^\triangle$ of H.

This deduction suggests to identify maximal paths in the triangle reduction $\overrightarrow{G}^\triangle$ of the directed acycling graph $\overrightarrow{G}$ as contigs. Since the topological sorting along any such path is unique, it automatically identifies any redundant non-triangle edges along a path.

On imperfect data $\overrightarrow{G}^\triangle$ differs from a unique golden path by bubbles, tips, and crosslinks (see Appendix A). Tips and bubbles predominantly are caused by edges that are missing e.g. due to mapping noise between reads that belong to a shared contig region. Hence, any path through a bubble or superbubble yields essentially the same assembly of the affected region and thus can be chosen arbitrarily, whereas tips may prematurely end a contig. Node-disjoint alternative paths within a (super-)bubble start and end in the neighborhood of the original path. Tips either originate or end in neighborhood of the chosen path.

Crosslinks represent connections between two proper contigs by spurious overlaps, caused, e.g., by repetitive elements that have escaped filtering. As crosslinks can occur at any positions, a maximal path may not necessarily follow the correct connection and thus may introduce chimeras into the assembly. As a remedy we measure how well an edge e fits into a local region that forms an induced proper interval graph. Recall that the out-neighborhood of each vertex in a proper interval graph induces a transitive tournament. For real data, however, the subgraph $\overrightarrow{G}[N_+(r)]$ induced by the out-neighbors of r may in general violate this expectation. The problem of finding the maximum transitive tournament in an acyclic graph is NP-hard [8]. An approximation can be obtained, however, using the fact that a transitive tournament has a unique directed Hamiltonian path. Finding a longest path in a DAG only requires linear time. Thus candidates for transitive tournaments in $\overrightarrow{G}[N_+(r)]$ can be retrieved efficiently as the maximal path P_{rq} in $\overrightarrow{G}[N_+(r)]$ that connects r with an endpoint q, i.e., a vertex without an outgoing edge within $\overrightarrow{G}[N_+(r)]$. Clearly, it suffices to consider the maximum path problem in the much sparser DAG $\overrightarrow{G}^\triangle[N_+(r)]$. The induced subgraph

$\overrightarrow{G}^{\triangle}[P_{rq}]$ with the largest edge set $H_r := E(\overrightarrow{G}^{\triangle}[P_{rq}])$, i.e., $q := \arg\max_p |E(\overrightarrow{G}^{\triangle}[P_{rp}])|$, serves as approximation for the maximal transitive tournament and is used to define the *interval support* of an edge $e \in E(\overrightarrow{G})$ as

$$\nu(e) := \sum_{r \in V(\overrightarrow{G}): e \in H_r} (|H_r| - d(r,e) - 1)\,. \tag{4}$$

Here, $d(r,e)$ is the minimal number of edges in the unique path from r to e in the path formed by the edges in H_r. The interval support can be interpreted as the number of triangles that support e as lying within an induced proper interval graph. It suffices to compute $\nu(e)$ for $e \in E(\overrightarrow{G}^{\triangle})$. We observed empirically that determining the best path with respect to $\nu(e)$ (rather than weight Ω of the spanning tree edges) results in contigs with a better solution quality. Taken together, we arrive at the following heuristic to iteratively extract meaningful paths (see also Appendix D):

i) Find the longest path $\mathbf{p} = r_1, \ldots, r_n$ in $\overrightarrow{G}^{\triangle}$ such that at every junction, we choose the incoming and outgoing edges e with maximal interval support $\nu(e)$.

ii) Add the path $\mathbf{p}$ to the contig set if it is at least two nodes long and neither the in-neighborhood $N_-(r_1)$ nor the out-neighborhood $N_+(r_n)$ are marked as previously visited in $\overrightarrow{G}$. Otherwise, we have found a tip if one of $N_-(r_1)$ or $N_+(r_n)$ was visited before and a bubble if both were visited. Such paths are assumed to have arisen from more complex crosslinks and can be added to the contig set if they exceed a user-defined minimum length.

iii) The path $\mathbf{p}$ is marked visited in $\overrightarrow{G}$ and all corresponding nodes and edges are deleted from $\overrightarrow{G}^{\triangle}$.

iv) The procedure terminates when $\overrightarrow{G}^{\triangle}$ is empty.

As the result, we obtain a set of paths, each defining a contig.

3.6 Consensus Sequence

The final step is the retrieval of a consensus sequence for each path $\mathbf{p}$. This step is more complicated than usual due to the nature of our initial mappings. While we enforce compatible sets of unitigs for each pair of long reads, a shared unitig between edges does not necessarily imply the same genomic coordinate. (i) Unitigs can be long enough that we gain triples $r_i, r_{i+1}, r_{i+2} \in V(\mathbf{p})$ such that an $s \in \mathcal{S}_{r_i r_{i+1}} \cap \mathcal{S}_{r_{i+1} r_{i+2}}$ exists but r_i and r_{i+2} share no interval on s. Such triples can occur chained. (ii) Unitigs of genomic repeats may remain in the data. Such unitigs may introduce pairwise distinct edges e_i, e_j, e_k that appear in this order, denoted by $e_i \prec e_j \prec e_k$, along the path $\mathbf{p}$ such that $s \in \mathcal{S}_{e_i}$ and $s \in \mathcal{S}_{e_k}$ but $s \notin \mathcal{S}_{e_j}$, therefore creating disconnected occurrences of s. (iii) Similarly, proximal repeats may cause inversions in the order of two unitigs $s, s' \in \mathcal{S}_{e_i} \cap \mathcal{S}_{e_k}$, w.l.o.g $e_i \prec e_k$. This scenario cannot appear on neighboring edges, as the shared node has a unique order of s and s'. Hence, either s or s' must be missing in an intermediary edge e_l due to the consistency constraints in the original graph, resulting in a situation as described in (ii). (iv) Finally, true matches of unitigs may be missing for some long reads due to alignment noise, which may also yield a situation as in (ii).

To address (i), we collect all instances of a unitig in the path independent of its context. We create an undirected auxiliary graph U_s with a vertex set$V(U_s) := \{e \in E(\mathbf{p}) \mid s \in \mathcal{S}_e\}$. We add edges for all edge-pairs that share an overlap in s. Any clique in this graph then represents a set of edges that share a common interval in s. We assign each edge a unique cluster index c_s^e, according to a minimal size clique decomposition. As finding a set of maximal cliques is NP-hard, we instead resort to a $\mathcal{O}(|V|/(\log|V|)^2)$ heuristic [4]. We address (ii-iv) with the help of a second index g_s^e, where $g_s^{e_i} \neq g_s^{e_k}$ for two edges e_i, e_k if and only if an edge e_j exists such that $e_i \prec e_j \prec e_j$ and $s \notin \mathcal{S}_{e_j}$.

Finally, we can now create a multigraph M consisting of vertex triples $\{(s, c_s^e, g_s^e) \mid s \in \mathcal{S}_e$ with $e \in E(\mathbf{p})\}$. We add edges $(s, c_s^e, g_s^e) \to (s', c_s'^e, g_s'^e)$ if and only if $s \prec s'$ on an edge e and no element s'' exists such that $s \prec s'' \prec s'$. The resulting graph is cycle free and thus uniquely defines the positions of all unitigs. Nodes represent the sequence of the common interval on the unitig s as attributed to the clique c_s^e. Edges represent the respective sequence of long reads between s and s', or a negative offset value if unitigs overlap. We take an arbitrary node in M and set its interval as the reference point. Positions of all other nodes are progressively built up following a topological order in this graph. If multiple edges exist between two nodes in this process a random but fixed edge is chosen to estimate the distance between nodes. As now all sequence features are embedded in the same coordinate system, an arbitrary projection of the sequence is set as the reference contig, retaining unitigs were possible due to their higher sequence quality. At the same time, we can map the features of each long read to their respective position in this newly constructed reference. This information can be directly fed into consensus based error correction systems such as racon [32].

4 Experimental Results

To demonstrate the feasibility of our assembly strategy we applied LazyB to publicly available datasets (see Appendix E) [9, 16, 31] for three well studied model organisms, baker's yeast (*S. cerevisiae*, genome size 12 Mb), fruit fly (*D. melanogaster*, genome size 140 Mb) and human (*H. sapiens*, genome size 3 Gb). The data were downsampled to approximately 5× and 10× nanopore coverage for long reads, respectively, and Illumina coverage sufficient for short-read anchors. We compare results to the most widespread competing assembler Canu [18], also highlighting the disadvantage of long read only strategies, DBG2OLC's [35] implementing the most closely related concept, as well as the recent competitor HASLR [11] based on also a similar strategy. For comparison, we also provide the statistics for short-read only assemblies created with ABySS [30] on the same sets of reads used to create the "anchors" to show the advantage of hybrid assembly even at a low coverage of long reads. Quality was assessed via alignment to a reference genome by the QUAST tool [10]; see Table 1. LazyB produced consistently better results than Canu, increasing genomic coverage at a lower contig count. Due to our inclusion of accurate short-read unitigs, overall error counts are also significantly lower. Most notably, Canu was unable to properly operate at the 5× mark for both data sets. Only insignificant portions of yeast could be assembled, accounting for less than 15% of the genome. Canu completely failed for fruit fly, even after adapting settings to low coverage. Even at 5×, LazyB already significantly reduces the number of contigs compared to the respective short-read assemblies, while retaining a reasonably close percentage of genome coverage. At only 10× coverage for fruit fly, we were able to reduce the contig count 10-fold at better error rates. For human, LazyB manages at 39-fold decrease of the number contigs, albeit at a loss of greater 10% coverage. This difference appears to be a consequence of the high fragmentation of unitigs in the abundant repeat regions of the genome, rendering them too unreliable as anchors. Results are indeed in line with unitig coverage. While HASLR produced the fewest mis-assemblies, it creates significantly more and shorter contigs that cover a much smaller fraction of the genome. As a consequence it has the least favorable N50 values of all tools. For fruit fly at 10×, it results in four times as many contigs and

Table 1 Assessment of assembly qualities for `LazyB`, `Canu`, and short-read only assemblies for two model organisms. `LazyB` outperforms `Canu` in all categories, while significantly reducing contig counts compared to short-read only assemblies. While `HASLR` is more accurate, it covers significantly lower fractions of genomes at a higher contig count and drastically lower N50. While `DBG2OL` produces few contigs at a high N50 for higher coverage cases, it calls significantly more mis-assemblies. Mismatches and InDels are given per 100kb. Accordingly, errors in `LazyB`'s unpolished output constitute < 1% except for human. Column descriptions: X coverage of sequencing data, **compl**eteness of the assembly. **#ctg** number of contigs, **#MA** number of mis-assemblies (breakpoints relative to the reference assembly) **M**is**M**atches and **InDels** relative to the reference genomes. N50 of **correctly assembled** contigs (minimal length of a correctly assembled contig needed to cover 50% of the genome, also named NGA50; omitted when < 50% is correctly recalled).

Org.	X	Tool	compl.[%]	#ctg	#MA	MM	InDels	N50
yeast	~5×	`LazyB`	90.466	127	9	192.56	274.62	118843
		`Canu`	14.245	115	5	361.47	2039.15	-
		`HASLR`	64.158	111	1	14.87	34.86	60316
		`DBG2OLC`	45.645	53	20	2066.64	1655.92	-
	~11×	`LazyB`	97.632	33	15	193.73	300.20	505126
		`Canu`	92.615	66	15	107.00	1343.37	247477
		`HASLR`	92.480	57	1	7.89	33.91	251119
		`DBG2OLC`	97.689	38	25	55.06	1020.48	506907
	~80×	`Abyss`	95.247	283	0	9.13	1.90	90927
fruit fly	~5×	`LazyB`	71.624	1879	68	446.19	492.43	64415
		`Canu`	-	-	-	-	-	-
		`HASLR`	24.484	1407	10	31.07	58.96	-
		`DBG2OLC`	25.262	974	141	1862.85	969.26	-
	~10×	`LazyB`	80.111	596	99	433.37	486.28	454664
		`Canu`	49.262	1411	275	494.66	1691.11	-
		`HASLR`	67.059	2463	45	43.83	84.89	36979
		`DBG2OLC`	82.52	487	468	739.47	1536.32	498732
	~45×	`Abyss`	83.628	5811	123	6.20	8.31	67970
human	~10×	`LazyB`	67.108	13210	2915	1177.59	1112.84	168170
	~43×	Unitig	69.422	4146090	252	93.07	13.65	338
	~43×	`Abyss`	84.180	510315	2669	98.53	25.03	7963

covers 10% less of the genome, with a 12 times lower N50. While an improvement to `Canu`, it also struggles on datasets with low Nanopore coverage. `DBG2OLC` shows the greatest promise compared to our own method, but similarly fails to operate well on very low coverage datasets. For yeast at 5×, less then 50% the genome can be reconstructed. In fruit fly even less then 25% can be assembled at about 2 times the error rate of `LazyB`. At 10×, `DBG2OLC` reconstruct a similar proportion of the genome, albeit at high error rates. While it produces about 100 fewer contigs for fruit fly, this achievement is offset by over 350 (4.7 times more) mis-assemblies.

The resource footprint of `LazyB` is small enough to run on an off-the-shelf desktop machine or even a laptop. The total effort is, in fact, dominated by the computation of the initial unitig set from the short reads. We expect that an optimized re-implementation of `LazyB` will render its resource consumption negligible. Compared to the competing `Canu` assembler, the combination of `ABySS` and the `python`-prototype of `LazyB` is already more than a factor of 60 faster. In terms of memory, given precomputed unitigs `LazyB` also requires 3 – 18 times less RAM than Canu, see Table 4. Most notably, we were able to assemble the human genome within only 3 days, while Canu could not be run within our resource constraints. HASLR shows a similar distribution of running times between tasks, overall operating slightly faster. We could not process our human test set with HASLR. A human DBG2OLC assembly can be estimated to take several weeks without manual parallelization for a single set of parameters, with authors recommending several possible alternatives for optimization. We therefore include only the results for LazyB here, and leave a more detailed comparison of the performance for very complex genomes for a proper follow-up experiment.

5 Discussion and Outlook

We demonstrated here the feasibility of a new strategy for sequence assembly with low coverage long-read data. Already the non-optimized prototype LazyB, written entirely in python, not only provides a significant improvement of the assembly but also requires much less time and memory than state-of-the-art tools. This is achieved by avoiding both a correction of long reads and an all-*vs*-all comparison of the long reads. Instead, we use rigorously filtered short-read unitigs as anchors to sparsifying the complexity of full string-graphs construction. LazyB then uses a series of fast algorithms to consistently orient this sparse overlap graph, reduce it to a DAG, and sort it topologically, before extracting contigs as maximum weight paths. This workflow relies on enforcing properties of overlap graphs that have not been exploited in this manner in competing sequence assembly methods.

The prototype implementation leaves several avenues for improvements. We have not attempted here to *polish* the sequence but only to provide a common coordinate system defined on the long reads into which the short-reads unitigs are unambiguously embedded to yield high-quality parts of the LazyB-assembly. The remaining intervals are determined solely by long-read data with their high error rate. Multiple edges in the multigraph constructed in the assembly step correspond to the same genome sequence, hence the corresponding fragments of reads can be aligned. This is also true for alternative paths between two nodes. This defines a collection of alignments distributed over the contig, similar to the situation in common polishing strategies based on the mapping of (more) short-read data or long reads to a preliminary assembly. Preliminary tests with off-the-shelf tools such as racon [32], however, indeed improve sequence identity but also tend to introduce new translocation breakpoints. We suspect this is the consequence of InDels being much more abundant than mismatches in Nanopore data, which is at odds with the Needleman–Wunsch alignments used by polishing tools.

A prominent category of mis-assemblies within the LazyB contigs are inherited from chimeric reads. This therefore suggests an iterative approach: Subsampling the long-read set will produce more fragmented contigs, but statistically remove chimeric reads from the majority of replicate assemblies. Final contigs are constructed in a secondary assembly step by joining intermediary results. It might appear logical to simply run LazyB again to obtain a "consensus" assembly, where intermediary contigs play the role of longer reads with mapped anchors. In preliminary tests, however, we observed that this results in defects that depend on the sampling rate. The question of how to properly design the majority calling to construct a consensus assembly remains yet to be answered.

References

1 Alfred V. Aho, Michael R. Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1:131–137, 1972. doi:10.1137/0201008.

2 Dmitry Antipov, Anton Korobeynikov, Jeffrey S McLean, and Pavel A Pevzner. hybridSPAdes: an algorithm for hybrid assembly of short and long reads. *Bioinformatics*, 32:1009–1015, 2016. doi:10.1093/bioinformatics/btv688.

3 Ali Baharev, Hermann Schichl, and Arnold Neumaier. An exact method for the minimum feedback arc set problem. Technical report, University of Vienna, 2015.

4 Ravi Boppana and Magnús M. Halldórsson. Approximating maximum independent sets by excluding subgraphs. *BIT Numerical Mathematics*, 32:180–196, 1992. doi:10.1007/BF01994876.

5 Chen-Shan Chin, Paul Peluso, Fritz J Sedlazeck, Maria Nattestad, Gregory T Concepcion, Alicia Clum, Christopher Dunn, Ronan O'Malley, Rosa Figueroa-Balderas, Abraham Morales-Cruz, Grant R. Cramer, Massimo Delledonne, Chongyuan Luo, Joseph R. Ecker, Dario Cantu, David R. Rank, and Michael C. Schatz. Phased diploid genome assembly with single-molecule real-time sequencing. *Nature methods*, 13:1050–1054, 2016. doi:10.1038/nmeth.4035.

6 Yun Sung Cho, Hyunho Kim, Hak-Min Kim, Sungwoong Jho, JeHoon Jun, Yong Joo Lee, Kyun Shik Chae, Chang Geun Kim, Sangsoo Kim, Anders Eriksson, et al. An ethnically relevant consensus korean reference genome is a step towards personal reference genomes. *Nature Comm.*, 7:13637, 2016. doi:10.1038/ncomms13637.

7 Alex Di Genova, Elena Buena-Atienza, Stephan Ossowski, and Marie-France Sagot. WENGAN: Efficient and high quality hybrid *de novo* assembly of human genomes. Technical Report 840447, bioRxiv, 2019. doi:10.1101/840447.

8 Kunal Dutta and C. R. Subramanian. Induced acyclic tournaments in random digraphs: sharp concentration, thresholds and algorithms. *Discussiones Mathematicae Graph Theory*, 34:467–495, 2014. doi:10.7151/dmgt.1758.

9 Francesca Giordano, Louise Aigrain, Michael A. Quail, Paul Coupland, James K. Bonfield, Robert M. Davies, German Tischler, David K. Jackson, Thomas M. Keane, Jing Li, Jia-Xing Yue, Gianni Liti, Richard Durbin, and Zemin Ning. *De novo* yeast genome assemblies from MinION, PacBio and MiSeq platforms. *Scientific Reports*, 7:1–10, 2017. doi:10.1038/s41598-017-03996-z.

10 Alexey Gurevich, Vladislav Saveliev, Nikolay Vyahhi, and Glenn Tesler. QUAST: quality assessment tool for genome assemblies. *Bioinformatics*, 29:1072–1075, 2013. doi:10.1093/bioinformatics/btt086.

11 Ehsan Haghshenas, Hossein Asghari, Jens Stoye, Cedric Chauve, and Faraz Hach. HASLR: Fast hybrid assembly of long reads. Technical Report 921817, bioRxiv, 2020. doi:10.1101/2020.01.27.921817.

12 Pinar Heggernes, Daniel Meister, and Charis Papadopoulos. A new representation of proper interval graphs with an application to clique-width. *Electronic Notes in Discrete Mathematics*, 32:27–34, 2009. doi:10.1016/j.endm.2009.02.005.

13 Hans J. Jansen, Michael Liem, Susanne A. Jong-Raadsen, Sylvie Dufour, Finn-Arne Weltzien, William Swinkels, Alex Koelewijn, Arjan P. Palstra, Bernd Pelster, Herman P. Spaink, et al. Rapid de novo assembly of the European eel genome from nanopore sequencing reads. *Scientific reports*, 7:7213, 2017. doi:10.1038/s41598-017-07650-6.

14 Arthur B. Kahn. Topological sorting of large networks. *Communications of the ACM*, 5:558–562, 1962. doi:10.1145/368996.369025.

15 Telikepalli Kavitha, Christian Liebchen, Kurt Mehlhorn, Dimitrios Michail, Romeo Rizzi, Torsten Ueckerdt, and Katharina A. Zweig. Cycle bases in graphs: characterization, algorithms, complexity, and applications. *Computer Science Review*, 3:199–243, 2009. doi:10.1016/j.cosrev.2009.08.001.

16 Hui-Su Kim, Sungwon Jeon, Changjae Kim, Yeon Kyung Kim, Yun Sung Cho, Jungeun Kim, Asta Blazyte, Andrea Manica, Semin Lee, and Jong Bhak. Chromosome-scale assembly

comparison of the korean reference genome KOREF from PromethION and PacBio with Hi-C mapping information. *GigaScience*, 8:giz125, 2019. doi:10.1093/gigascience/giz125.

17 Mikhail Kolmogorov, Jeffrey Yuan, Yu Lin, and Pavel A Pevzner. Assembly of long, error-prone reads using repeat graphs. *Nature Biotech.*, 37:540–546, 2019. doi:10.1038/s41587-019-0072-8.

18 Sergey Koren, Brian P Walenz, Konstantin Berlin, Jason R. Miller, Nicholas H. Bergman, and Adam M. Phillippy. Canu: scalable and accurate long-read assembly via adaptive *k*-mer weighting and repeat separation. *Genome Research*, 27:722–736, 2017. doi:10.1101/gr.215087.116.

19 Joseph B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956. doi:10.1090/S0002-9939-1956-0078686-7.

20 Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34:3094–3100, 2018. doi:10.1093/bioinformatics/bty191.

21 Zhenyu Li, Yanxiang Chen, Desheng Mu, Jianying Yuan, Yujian Shi, Hao Zhang, Jun Gan, Nan Li, Xuesong Hu, Binghang Liu, Bicheng Yang, and Wei Fan. Comparison of the two major classes of assembly algorithms: overlap–layout–consensus and de-bruijn-graph. *Briefings Funct. Genomics*, 11:25–37, 2012. doi:10.1093/bfgp/elr035.

22 Pierre Marijon, Rayan Chikhi, and Jean-Stéphane Varré. yacrd and fpa: upstream tools for long-read genome assembly. Technical Report 674036, bioRxiv, 2019. doi:10.1101/674036.

23 Samuel Martin and Richard M. Leggett. Alvis: a tool for contig and read ALignment VISualisation and chimera detection. Technical Report 663401, BioRxiv, 2019. doi:10.1101/663401.

24 George B. Mertzios. A matrix characterization of interval and proper interval graphs. *Applied Mathematics Letters*, 21:332–337, 2008. doi:10.1016/j.aml.2007.04.001.

25 Burkhard Morgenstern. A simple and space-efficient fragment-chaining algorithm for alignment of DNA and protein sequences. *Applied Mathematics Letters*, 15:11–16, 2002. doi:10.1016/S0893-9659(01)00085-4.

26 Sergej Nowoshilow, Siegfried Schloissnig, Ji-Feng Fei, Andreas Dahl, Andy WC. Pang, Martin Pippel, Sylke Winkler, Alex R. Hastie, George Young, Juliana G. Roscito, Francisco Falcon, Dunja Knapp, Sean Powell, Alfredo Cruz, Han Cao, Bianca Habermann, Michael Hiller, Elly M. Tanaka, and Eugene W. Myers. The axolotl genome and the evolution of key tissue formation regulators. *Nature*, 554:50–55, 2018. doi:10.1038/nature25458.

27 Sergey Nurk, Brian P. Walenz, Arang Rhie, Mitchell R. Vollger, Glennis A. Logsdon, Robert Grothe, Karen H. Miga, Evan E. Eichler, Adam M. Phillippy, and Sergey Koren. Hicanu: accurate assembly of segmental duplications, satellites, and allelic variants from high-fidelity long reads. *bioRxiv*, 2020. doi:10.1101/2020.03.14.992248.

28 Jue Ruan and Heng Li. Fast and accurate long-read assembly with wtdbg2. *Nature Methods*, 17:155–158, 2020. doi:10.1038/s41592-019-0669-3.

29 Kishwar Shafin, Trevor Pesout, Ryan Lorig-Roach, Marina Haukness, Hugh E Olsen, Colleen Bosworth, Joel Armstrong, Kristof Tigyi, Nicholas Maurer, Sergey Koren, Fritz J. Sedlazeck, Tobias Marschall, Simon Mayes, Vania Costa, Justin M. Zook, Kelvin J. Liu, Duncan Kilburn, Melanie Sorensen, Katy M. Munson, Mitchell R. Vollger, Evan E. Eichler, Sofie Salama, David Haussler, Richard E. Green, Mark Akeson, Adam Phillippy, Karen H. Miga, Paolo Carnevali, Miten Jain, and Benedict Paten. Efficient *de novo* assembly of eleven human genomes using PromethION sequencing and a novel nanopore toolkit. Technical Report 715722, BioRxiv, 2019. doi:10.1101/715722.

30 Jared T. Simpson, Kim Wong, Shaun D. Jackman, Jacqueline E. Schein, Steven JM. Jones, and Inanç Birol. ABySS: a parallel assembler for short read sequence data. *Genome Research*, 19:1117–1123, 2009. doi:10.1101/gr.089532.108.

31 Edwin A. Solares, Mahul Chakraborty, Danny E. Miller, Shannon Kalsow, Kate Hall, Anoja G. Perera, JJ. Emerson, and R. Scott Hawley. Rapid low-cost assembly of the *Drosophila*

melanogaster reference genome using low-coverage, long-read sequencing. *G3: Genes, Genomes, Genetics*, 8:3143–3154, 2018. doi:10.1534/g3.118.200162.

32 Robert Vaser, Ivan Sović, Niranjan Nagarajan, and Mile Šikić. Fast and accurate de novo genome assembly from long uncorrected reads. *Genome Research*, 27:737–746, 2017. doi:10.1101/gr.214270.116.

33 Ryan R. Wick, Louise M. Judd, Claire L. Gorrie, and Kathryn E. Holt. Unicycler: resolving bacterial genome assemblies from short and long sequencing reads. *PLOS Computational Biology*, 13:e1005595, 2017. doi:10.1371/journal.pcbi.1005595.

34 Ryan R. Wick, Louise M. Judd, and Kathryn E. Holt. Deepbinner: Demultiplexing barcoded Oxford Nanopore reads with deep convolutional neural networks. *PLOS Computational Biology*, 14:e1006583, 2018. doi:10.1371/journal.pcbi.1006583.

35 Chengxi Ye, Christopher M Hill, Shigang Wu, Jue Ruan, and Zhanshan Sam Ma. DBG2OLC: efficient assembly of large genomes using long erroneous reads of the third generation sequencing technologies. *Scientific reports*, 6:31900, 2016. doi:10.1038/srep31900.

36 Aleksey V Zimin, Guillaume Marçais, Daniela Puiu, Michael Roberts, Steven L Salzberg, and James A Yorke. The MaSuRCA genome assembler. *Bioinformatics*, 29:2669–2677, 2013. doi:10.1093/bioinformatics/btt476.

A Definitions of Alignment Graph Defects

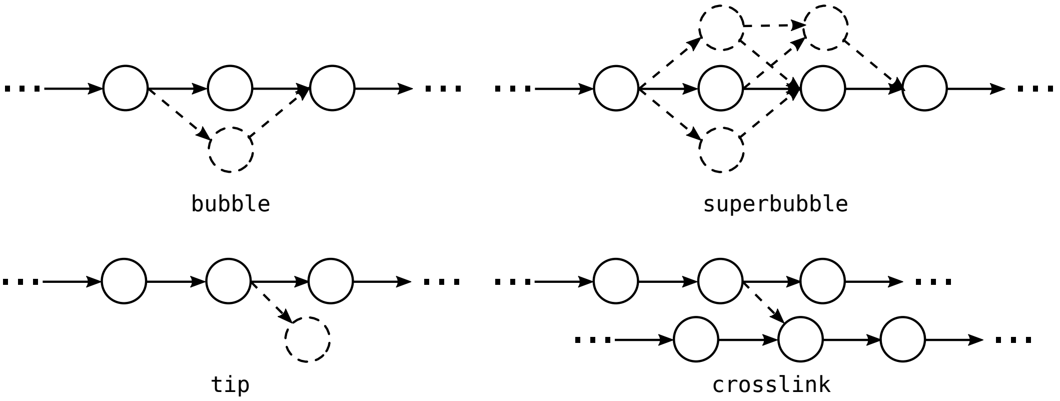


Figure 4 Examples of assembly graph defects in $\overrightarrow{G}^{\triangle}$.

On ideal data, $\overrightarrow{G}^{\triangle}$ would consist of a unique golden path. For real data, however, it also also harbors bubbles, tips, and crosslinks. We briefly define these types of imperfections here. Given two nodes $s, t \in \overrightarrow{G}^{\triangle}$, an $s - t$ path is a path starting in s and ending in t. A *simple bubble* consists of two vertex disjoint $s - t$ paths. This construct can be extended to *super-bubbles*, defined as a set of $s - t$ paths, exactly including all nodes reachable from s without passing t and *vice versa*. Bubbles and superbubbles are primarily the result of unrecognized overlaps. *Tips* are "side branches" that do not reconnect with the dominating paths and thus have distinct end-points. *Crosslinks*, finally, are connecting edges between two golden paths. As tips themselves may also be subject to mild noise, and crosslinks may occur near the start- or end-sites of the true paths, both are not always easily distinguished. Hence, we apply the heuristic filtering steps described in the main text.

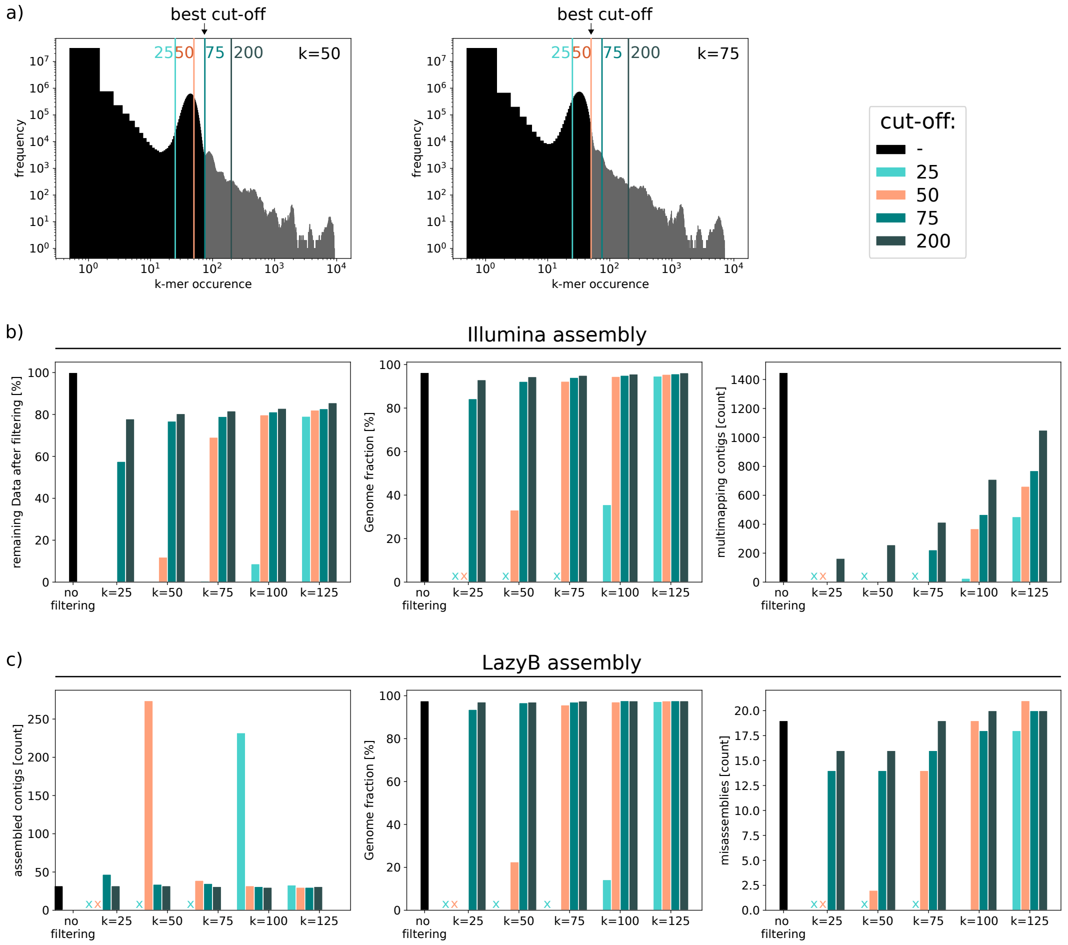


Figure 5 Assembly statistics of yeast as a function of the k-mer size and maximal occurrence cut-off used to remove very frequent k-mers from short reads prior to unitig assembly. a) k-mer profiles for k=50 bp and k=75 bp. Cut-offs restrict short reads to different degrees. Note logarithmic axes. b) Illumina unitigs (left: percentage of remaining short-read data; middle: fraction of the reference genome covered; right: number of unitigs mapping multiple times to reference). c) final LazyB assembly left: number of unitigs; middle: fraction of the reference genome covered; right: number of mis-assemblies). x: not enough data to assemble.

B Influence of Short Read Filtering

The strategies for filtering short-read data have a larger impact than the choice of the k-mer size for unitig assembly (Fig. 5). This is not surprising given that both chimeric unitigs and unitigs that harbor repetitive DNA elements introduce spurious edges into G and thus negatively influence the assembly. In order to exclude short reads that contain highly frequent k-mers, the maximal tolerated occurrence has to be set manually and is dependent on the k-mer size. Setting the cut-off right next to the main peak in the profiles has turned out to be a good estimate. After assembling short reads, unitigs are mapped to long reads and a coverage profile over the length of every unitig is calculated. Unitigs with maximal coverage above interquartile range $IQR \times 1.5 + Q3$ are considered outliers. However, regions below coverage threshold (Q3) spanning more than 500 bp can be "rescued". This filter step effectively reduces ambiguous regions, especially when no previous filtering is applied (Fig. 6). Combining both short-read filter improves the assembly quality; see Table 2.

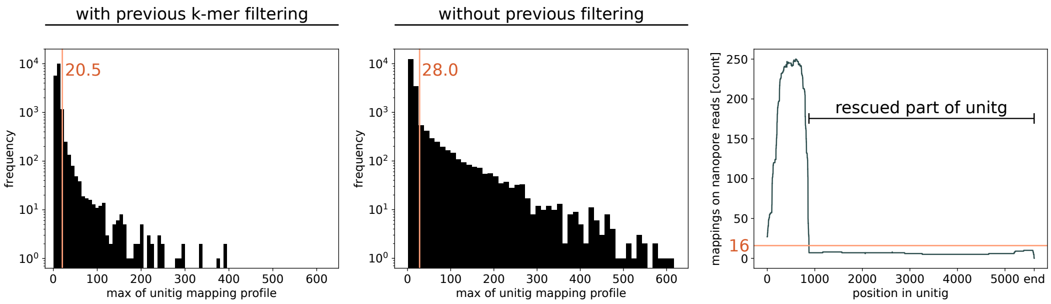


Figure 6 Exclusion of unitigs based on very high mapping coverage. Thresholds are IQR×1.5+Q3. Shown are maximal values of coverage profiles for unitigs assembled with (left) and without (middle) previous k-mer filtering. Note the logarithmic axes. right: exemplary profile; only the high-coverage peak is excluded. Threshold is Q3.

Table 2 Impact of short-read filtering strategies on LazyB assembly quality in fruit fly. Column descriptions: **compl**eteness of the assembly, **#ctg** number of contigs, **#MA** number of misassemblies (breakpoints relative to the reference assembly).

Filter strategy	compl.[%]	#ctg	#MA
no filter	82.81	457	302
***k*-mer filter**	80.66	567	104
unitig filter	80.71	563	108
***k*-mer and unitig filter**	80.11	596	99

C Validation of Minimap2 Anchor Alignments

Classic alignment tools, even those specifically advertised for this purpose, rely on scoring schemes that cannot accurately represent the high InDel profiles of long-read data. Instead, they rely on seeds of high quality matches that are then chained with high error tolerance. Currently, minimap2 is one of the most commonly used tools for this purpose. Since we do not have a gold set of perfect data, we can only roughly estimate the influence of this heuristic on the LazyB alignment quality in a related experiment. Specifically, we test consistency of anchor alignments on pairs of long reads to direct alignments of both reads for fruit fly. Consistency is validated at the level of relative orientation, the offset indicated by both alignment methods, the portion of overlap that can be directly aligned and whether direct alignment of the long reads is possible at all. Different relative orientations were observed only in very small numbers. Changes in the offset by more then 5% of the longer read length are equally rare (Fig. 7). However, requiring a direct alignment of at least 75% of the overlap region marks 4.6% of the anchor links as incorrect. Removing those has a negative effect on the final LazyB assembly and in particular tends to break correct contigs apart; see Table 3. In our test set 7.7% of direct alignments of two anchor-linked long reads gave no result. In these cases, expected overlaps are rather short (Fig. 7). We therefore tested whether the assembly could be improved by excluding those connections between long reads for which no alignment could be calculated despite the presence of an overlap of at least 1 kbp (3.7%). We found, however, that this procedure also causes the loss of correct edges in G.

Summarizing, we observe three facts: (1) The overwhelming number of pairs is consistent and therefore true. (2) Removing inconsistent edges from the assembly not only does not improve the results but results are worse on average. (3) While we can manually identify some incorrect unitig matches, the mappings produced by minimap2 are too inconsistent

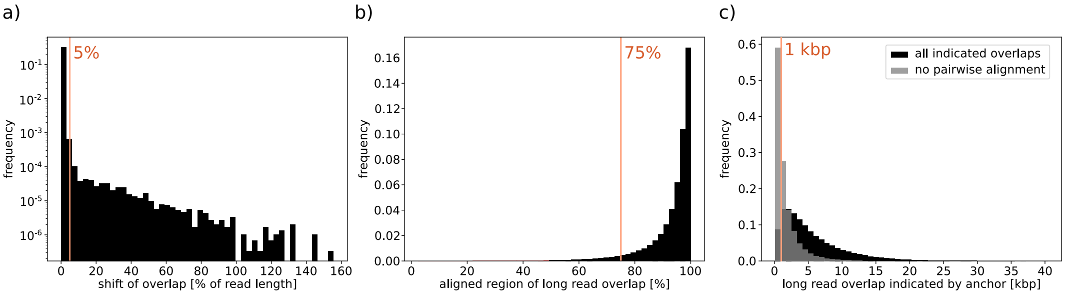


Figure 7 Consistency test of anchor-linked long-read overlaps to direct alignments of both reads on fruit fly. a) Frequencies of shifted offsets (% of the longer read); changes up to 5% are tolerated; note logarithmic axis. b) Frequencies of the percentage at which the direct alignment covers the overlap. A minimum of 75% is set for consistency. c) Long read pairs where no direct alignment is possible tend to have shorter anchor-indicated overlaps. Connections that cannot be confirmed via direct alignments despite an expected overlap of at least 1 kbp are excluded.

for proper testing. Since we have no proper methods to identify such false positives we also cannot properly estimate the number of false negatives, i.e., missing matches in the graph $\vec{G}$, e.g. by computing a transitive completion.

Overall, our main results together with (1) indicate that a high level trust in the anchors mapping in warranted. We also conclude that `minimap2` is sufficient for our purposes. However, the data also suggest that the assembly would profit from a more accurate handling of the alignments.

Table 3 Assessment of different parameters to verify long-read overlaps and their impact on `LazyB` assembly quality on fruit fly. Overlaps are indicated by anchors and evaluated by pairwise long-read alignments. They are considered valid if: the relative direction suggested by the anchor matches that of the pairwise alignment (direction); the offset is sufficiently similar for both methods (offset); at least 75% of the overlap is found as direct alignment (incomplete mapping); the overlap indicated by the anchor is less than or equal to 1 kbp or a pairwise alignment is possible (no mapping). Column descriptions: **compl**eteness of the assembly, **#ctg** number of contigs, **#MA** number of mis-assemblies (breakpoints relative to the reference assembly).

Varification parameters	**compl.**[%]	**#ctg**	**#MA**
direction	80.13	608	111
direction + offset	80.08	622	103
direction + offset + incomplete mapping	80.04	1263	121
no mapping	80.15	801	113

D Alternative Heuristic for Maximum Induced Transitive Tournament

We found that $\nu(e)$ provides a better heuristic than the initial bit scores $\Omega(e)$ for the extraction of the paths. Most plausibly, one is interested transitive tournaments as an indication for the correct assembly path. Since this is a computationally difficult problem, we described in the main text a heuristic based on longest paths in $\vec{G}^{\triangle}$, that is equivalent for perfect data. Here we briefly sketch an alternative heuristic operating directly on the edges of $\vec{G}$.

Here, we denote by $N_{+}(r)$ and $N_{-}(r)$ the set of out- and in-edges of r in $\vec{G}$. We note that the out-neighbors of r form an induced proper interval graph if and only if they are an acyclic tournament (AT). With noisy data, we therefore ask for the maximal AT $\vec{K}_r$ with source r.

Unfortunately, this task is NP-hard even in acyclic graphs [8]. We therefore resort to a heuristic making use of the topological order in $N_+(r)$ inherited from the topological order of $\overrightarrow{G}$ and process nodes in increasing order starting with r: (i) Initialize a list $\mathcal{L}$ of candidates with the pair (H, A) where $H = \{r\}$ and $A = N_+(r)$. (ii) For each candidate $(H, A) \in \mathcal{L}$ consider the $r_i \in A$ in topological order and append (H'_i, A'_i) to $\mathcal{L}$, where $H' = H \cup \{r_i\}$ and $A' = A \cap N_+(r_i)$. (iii) Select the candidate with maximum cardinality $|H_r|$.

E Evaluation of Real Data Sets

We re-used data sets from previously published benchmarks of Nanopore assemblies. For yeast (*S. cerevisiae*) we used Nanopore sets ERR1883389 for lower coverage, ERR1883399 for higher coverage, and short-reads set ERR1938683, all from bioproject PRJEB19900 [9]. For comparison we use the reference genome R64.2.1 of strain S288C from the SGD. For the fruit fly (*D. melanogaster*) we used the Oxford Nanopore and Illumina raw data of bioproject PRJNA433573 [31], and the FlyBase reference genome 6.30 (`http://www.flybase.org`). On Human we use SRX6356866-8 on bioproject PRJNA549351 [16] for long reads and SRA292482 [6] for short reads. We compare against reference GRCh38.p13. `QUAST` [10] is a specialized tool to evaluate the quality of assemblies. We report statistics without further processing. Table 4 summarizes the resource requirements for the assembly of the yeast, fruit fly, and human data set.

Table 4 Assessment of running times for `LazyB` and `Canu`. Resources for `LazyB` are given in three steps: 1) `ABySS` unitig assembly; 2) Mapping of unitigs to long reads and 3) `LazyB` itself. Step 1) is often not needed as short-read assemblies are available for many organisms. Resources are only compared for yeast and fruit fly, as `Canu` cannot be run for human in sensible time and resource-constraint on our machine. As all tools except `LazyB` and `DBG2OL` are parallelized, running times are given as the sum of time spent by all CPUs. `ABySS` greatly dominates the `LazyB` pipeline. Nevertheless, `LazyB` is faster on a factor of > 60 to `Canu` and ≈ 3 to `DBG2OL`.

Organism	**X**	**Tool**	**Runtime** (dd:hh:mm:ss)	**RAM** (MB)
yeast		`ABySS`	00:00:11:03	2283
	~5×	Mapping	00:00:00:05	540
		`LazyB`	00:00:00:30	136
		`ABySS` + Mapping + `LazyB`	00:00:11:38	2283
		`Canu`	00:10:23:55	2617
		`HASLR`	00:00:06:44	4922
		`DBG2OL`	00:00:31:46	1141
	~11×	Mapping	00:00:00:15	1544
		`LazyB`	00:00:01:46	362
		`ABySS` + Mapping + `LazyB`	00:00:13:04	2283
		`Canu`	00:13:44:16	6779
		`HASLR`	00:00:08:09	4922
		`DBG2OL`	00:00:51:13	1264
fruit fly		`ABySS`	00:02:32:39	25344
	~5×	Mapping	00:00:02:43	6433
		`LazyB`	00:00:08:33	613
		`ABySS` + Mapping + `LazyB`	00:02:43:55	25344
		`Canu`	02:13:51:39	7531
		`HASLR`	00:01:30:33	5531
		`DBG2OL`	00:07:58:22	6151
	~10×	Mapping	00:00:06:11	9491
		`LazyB`	00:00:11:57	2241
		`ABySS` + Mapping + `LazyB`	00:02:50:47	25344
		`Canu`	07:04:08:28	7541
		`HASLR`	00:01:43:21	5553
		`DBG2OL`	02:07:32:01	17171

A Graph-Theoretic Barcode Ordering Model for Linked-Reads

Yoann Dufresne
Department of Computational Biology, C3BI USR 3756 CNRS, Institut Pasteur, Paris, France
yoann.dufresne@pasteur.fr

Chen Sun
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, USA
chensunx@gmail.com

Pierre Marijon
Center for Bioinformatics, Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
pmarijon@mmci.uni-saarland.de

Dominique Lavenier
IRISA, Inria, Université de Rennes, France
lavenier@irisa.fr

Cedric Chauve
Department of Mathematics, Simon Fraser University, Burnaby, Canada
LaBRI, Université de Bordeaux, France
cedric.chauve@sfu.ca

Rayan Chikhi
Department of Computational Biology, C3BI USR 3756 CNRS, Institut Pasteur, Paris, France
rayan.chikhi@pasteur.fr

Abstract

Considering a set of intervals on the real line, an interval graph records these intervals as nodes and their intersections as edges. Identifying (i.e. merging) pairs of nodes in an interval graph results in a multiple-interval graph. Given only the nodes and the edges of the multiple-interval graph without knowing the underlying intervals, we are interested in the following questions. Can one determine how many intervals correspond to each node? Can one compute a walk over the multiple-interval graph nodes that reflects the ordering of the original intervals? These questions are closely related to linked-read DNA sequencing, where barcodes are assigned to long molecules whose intersection graph forms an interval graph. Each barcode may correspond to multiple molecules, which complicates downstream analysis, and corresponds to the identification of nodes of the corresponding interval graph. Resolving the above graph-theoretic problems would facilitate analyses of linked-reads sequencing data, through enabling the conceptual separation of barcodes into molecules and providing, through the molecules order, a skeleton for accurately assembling the genome. Here, we propose a framework that takes as input an arbitrary intersection graph (such as an overlap graph of barcodes) and constructs a heuristic approximation of the ordering of the original intervals.

2012 ACM Subject Classification Applied computing → Bioinformatics

Keywords and phrases DNA sequencing, graph algorithms, linked-reads, interval graphs, cliques

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.11

Supplementary Material `https://gitlab.pasteur.fr/ydufresne/linkedreadsmoleculeordering`

Funding *Cedric Chauve*: Natural Sciences and Engineering Research Council of Canada Discovery Grant RGPIN-03986.
Rayan Chikhi: INCEPTION project (PIA/ANR-16-CONV-0005).

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti; Article No. 11; pp. 11:1–11:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements The authors are grateful to Paul Medvedev and Jean-Stéphane Varré for preliminary work and discussions and to Marthe Bonamy for discussions about interval graphs models.

1 Introduction

A well-known limitation of short-read sequencing is that it does not provide long-range information, which is crucial to many biological endeavors, such as genome assembly and structural variant identification. There have been several sequencing technologies developed to overcome this limitation, such as matepair libraries, Hi-C, and long reads (PacBio & Oxford Nanopore). Another family of approaches is linked-read sequencing, which includes 10XGenomics Chromium, stLFR [28], CPTv2-seq [32] and TELL-seq [8]. In these approaches, DNA is cloned and cut into large molecules (10-100 kbp), which are then isolated (physically in 10X, or virtually using beads) and sheared into shorter fragments. A barcode is attached to each short fragment for identification of its originating molecule. Importantly, barcodes do not uniquely identify molecules: several molecules are typically labeled with the same barcode. The number of different barcodes differ from 150k for CPTv2 to 2 billions for TELL-seq. Fragments are then sequenced using a standard short-read protocol (e.g Illumina).

Linked-reads have been used to assemble genomes [29], detect complex structural variants [16], and more recently assemble metagenomes [4]. A common challenge faced by most linked-read methods is that in order to make use of the linking information, the reads within each barcode should be first separated into their constituent molecules. More formally, for each read r, we would like to find the identifier $mi(r)$ of its originating molecule, given as input an observed identifier $b(mi(r))$, where $b(x)$ associates a barcode identifier to a molecule x. Note that the image of b (all barcodes) is significantly smaller than its domain (all molecules), hence b can be viewed as a non-invertible hash function. Currently, this problem is being tackled, one way or another, as part of any method using linked-read data. Switching from a read-centric view to a molecule-centric view opens the possibility of using methodology similar to long-read overlap graphs. Finding an ordering of barcodes that reflects the underlying order of molecules would indeed greatly facilitate and decrease errors during the scaffolding stage of genome assembly. As noted by the authors of the ARCS scaffolder [31], different molecules having the same barcode can induce false joins in a scaffolding algorithm, resulting in misassemblies.

Linked-read mapping tools such as `longranger` or `ema` [25] are able to infer molecules by clustering mapping locations of reads from the same barcode. While such reference-based algorithms are often applicable, they do not replace the need for *de novo* algorithms. The quality of reference-based algorithms is related to the quality of the assembly, since clusters cannot be identified across different contigs. When the genome or metagenome references are in a draft state, molecules will frequently span multiple contigs, preventing their identification. Moreover, in many situations the reference is simply unavailable.

To the best of our knowledge, the barcode ordering problem has not been previously studied, and the assignment of molecule identifiers to reads without a reference has only been previously studied in [11], where it was referred to as *barcode deconvolution*. The authors first constructed a bipartite graph between reads and barcodes. An edge (r, b) was added when a k-mer of read r was found in another read of barcode b. Then a second graph was constructed with reads as nodes, and edges indicating whether two reads were connected to sufficiently many common barcodes in the bipartite graph. Finally, the second graph was clustered and each cluster reflected reads from the same molecule. This algorithm was implemented in a software called `Minerva`. We note that Minerva only assigns molecules identifiers to a fraction of the reads. In our tests on a simulated *E. coli* dataset, Minerva reported results for 12% of the reads, inferring around 50% of the true number of molecules.

This raises the question of whether reference-free inference of molecules is fundamentally unsolvable in the setting of linked-read data, or whether an adequate technique has just not yet been found. Surely there exist corner cases where the problem is either impossible, e.g. in an hypothetical situation where all molecules are assigned to the same barcode, or trivial, when each molecule is assigned to a different barcode. As we will see in Section 2, while there exist previous works in graph theory (e.g. in a setting corresponding to all barcodes containing 2 molecules each), the general setting does not appear to have previously been studied.

In this paper, we establish theoretical grounds for studying the feasibility of inferring molecule without a reference genome. We will not directly tackle the problem of assigning molecule identifiers to reads (as Minerva does), but instead we look at two problems which can be reduced, in the complexity sense, to molecule inference:

1. **Molecule counting**: count the number of molecules assigned to each barcode
2. **Molecule ordering**: reconstruct a total (or partial) order of molecules as a sequence of barcodes

Both problems, if solved accurately, can provide useful information for barcode deconvolution (molecule counting) and genome scaffolding (molecule ordering). Staying at the level of barcodes and molecules instead of reads will allow to thoroughly establish expectations on whether molecule inference is at all feasible, and how various parameters (e.g. number of molecules, how many molecules per barcode, etc) influence its difficulty.

We first present the commononalities between the barcodes ordering problem, and the previously-known concepts of interval graphs and multiple-interval graphs. We then introduce the notion of barcode graph, which models overlaps between molecules across different barcodes. We discuss its link with well-known graph classes leading to the conclusion that solving the molecules ordering problem for a barcode graph is likely difficult. Next we introduce another graph structure, the local clique-pairs graph, inspired of approaches used to realise an interval graph. By identifying maximal cliques in the barcode graph, which are then paired into structures that we call local clique-pairs, we show that the local clique-pairs graph captures a strong signal related to the ordering of the barcodes according to their underlying molecules. We apply this technique to synthetic interval graphs, as well as barcode graphs constructed from simulated molecules from a real genome, and show that on synthetic interval graphs we are able to accurately count the number of molecules per barcode, and reconstruct an approximate but accurate molecule ordering on barcodes. Finally, we demonstrate how to construct a barcode graph directly from linked-read sequencing data.

2 Models and Methods

We consider the problem of sequencing a single long DNA molecule (e.g. a chromosome) using linked reads. We assume that the sequencing data were obtained by sequencing n fragments (called *molecules* from now) from the chromosome, each molecule being assigned a barcode, where several molecules can be assigned the same barcode; for a molecule m we denote by $b(m)$ its barcode. We denote by $\mathcal{B}$ the barcode alphabet and by $|\mathcal{B}| = \mu$ its size, i.e. the total number of observed barcodes; for a barcode b we denote by $m(b)$ the molecules it labels (the barcode size). Let $F = \max_{b\in\mathcal{B}} |m(b)|$. Finally, we assume that no two molecules do start at the same coordinate, which implies that molecules can be totally ordered by their start coordinates.

2.1 Barcode graphs and families of interval graphs.

The sequenced molecules can be seen as intervals along the real line if the sequenced chromosome is linear, or arcs around a circle if it is circular; their *intersection graph* is the graph whose vertices are the n molecules and two vertices are linked by an edge if the corresponding intervals do intersect. Intersection graphs of intervals on the real line (resp. arcs around a circle) form the class of *interval graphs* (resp. *circular-arc graphs*). It is well-known that deciding if a graph is an interval graph or an arc-circular graph can be done in linear time [6, 23], and many algorithmic problems that are computationally hard in general graphs are tractable in these graph classes [15].

However, the result of the sequencing experiment with linked reads does not provide direct knowledge of the sequenced molecules and of their intersections, as the reads originating from molecules having the same barcode b are all labeled by b and, as discussed in introduction, the problem of separating reads with the same barcodes into clusters corresponding to molecules is non-trivial. Nevertheless, we assume here first that it is possible to infer, from the barcoded reads if, for a given pair of barcodes b_1, b_2 there exists molecules m_1 and m_2 such that $b(m_1) = b_1$, $b(m_2) = b_2$ and and m_1 and m_2 do intersect: we then say that barcodes b_1 and b_2 do *intersect*. We assume here moreover that we do not observe two intersecting molecules m_1 and m_2 such that $b(m_1) = b(m_2)$[1].

▶ **Definition 1.** *The exact barcode graph of a set of barcoded molecules is the graph with vertex set $\mathcal{B}$ and edges between pairs of intersecting barcodes.*

In the case of a linear chromosome, exact barcode graphs generalize the class of interval graphs and form another well-studied graphs class, *multiple-interval graphs* [12]. Moreover if we assume that each barcode labels exactly f molecules, exact barcode graphs form the class of f-interval graphs; finally, under the additional assumption that all sequenced molecules have exactly the same length, exact barcode graphs are equivalent to the class of *unit f-interval graph.* We are not aware of any study of the equivalent graph classes for circular chromosomes, i.e. arcs around a circle, and from now on we concentrate on the case of linear chromosomes. We describe below the formulation of several algorithmic problems related to barcode graphs and how they translate into problems on the aforementioned graph classes. Note that an exact barcode graph can be a multi-graph (a graph where multiple edges may have the same endpoints) in the case where there exist molecules m_1, m_2, m_3, m_4 with $b(m_1) = b(m_3)$, $b(m_2) = b(m_4)$ and m_1, m_2 (resp. m_3, m_4) do intersect.

Recognizing exact barcode graphs. The link with unit f-interval graph, although it assumes an unrealistic uniformity in the sequencing process (uniform molecules length and uniform number of molecules per barcode) sheds a light on the computational hardness of analyzing barcoded sequencing data. Indeed, recognizing 2-interval graphs is NP-complete [30], while the complexity of recognizing unit f-interval graphs is still open, the only positive recognition result being for depth-2 unit f-interval graphs [18], corresponding to the case where no chromosome base is covered by more than two molecules, an unrealistic assumption for sequencing experiments. To the best of our knowledge, given a graph on a barcode alphabet whose edges represent possible molecules intersections, deciding if it is an exact barcode graph, even in the setting of molecules of uniform length and barcodes of uniform size, is open.

[1] We justify this assumption as such molecules could be seen as a single molecule defined by the union of m_1 and m_2; moreover, simulations with realistic sequencing parameters show that this situation occurs rarely and most often with molecules that share a small intersection.

Realizing an exact barcode graph. A barcode sequence is a sequence $b_1 \dots b_n$ over the barcode alphabet. Given a barcode graph BG, a barcode sequence *realizes* BG if every edge of BG can be assigned to two barcodes of the sequence in such a way that if b_j is covered by an edge between b_i and b_k (i.e. $i < j < k$) then there are also edges between b_i and b_j and between b_j and b_k. The molecules ordering problem applied to an exact barcode graph BG is then equivalent to finding a barcode sequence realizing BG. This problem is tractable in the case of interval graphs ($F = 1$); note that if intervals lengths are also fixed, then the problem becomes NP-complete [24], while it solvable in polynomial time if additionally the intersection lengths are provided [19]. We are not aware of similar tractability results for multiple-interval graphs. However, existing algorithms to realize interval graphs are mainly based on the property that such a realization can be obtained by a sequence of overlapping maximal cliques. While maximal cliques are easy to find in an interval graph, it is not the case in multiple-interval graph, as the problem of finding the maximum clique in multiple-interval graphs is NP-complete, even for unit 2-interval graphs [13], although approximation and parameterized algorithms do exist [7, 12]. Moreover a structural property of interval graphs that is important toward the realization through maximal cliques, the existence of a vertex whose neighbourhood is a clique, does not hold for multiple interval graphs [2]. Finally, it is easy to see that a maximal clique of size c in an exact barcode graph might not correspond to a set of c pairwise intersecting molecules. This leads us to conjecture that realizing an exact barcode graph is difficult.

Handling inexact barcode graphs. Constructing an exact barcode graph implies to detect intersecting barcodes from sequenced barcoded reads and it is thus likely unrealistic to expect perfectly obtaining such a graph from sequencing data. It follows that solving the molecules ordering problem would then implicitly assumes to solve a graph modification problem, aimed at transforming a graph into a multiple-interval graph, with additional constraints about the number of occurrences of barcodes in a realization. Graph modification problems that aim to minimize the number of modifications are generally hard, even in the case of interval graph, [10], and so for multiple-interval graphs; note however that it was recently shown to be fixed-parameter tractable [27, 5, 10]. Such problems naturally translates into vertex ordering optimization problems (also known as graph layout problems) that can, in principle, be addressed with combinatorial optimization techniques such as Integer Linear Programming (ILP). However, ILP approaches to vertex ordering currently do not scale to the size of instances corresponding to sequencing experiments [9].

From the link we described above between barcode graphs and multiple-interval graphs, and the current state-of-the art in multiple-interval graphs algorithms, it does not appear that the problem of realizing a barcode graph can be addressed by existing algorithms, and we actually conjecture that this problem is difficult, whether the provided barcode graph is exact or not. Nevertheless, toward application to real sequencing data, additional assumptions about the sought realization, such as the expected length of intervals or the expected size of the barcodes, lead to specific open problems of interest in the field of multiple-interval graphs algorithms that deserves further research.

2.2 The Local Clique-Pairs Graph

In this section, we assume that we are given a barcode graph BG. The barcode graph needs not be perfect: it might contain additional (wrong) edges that do not correspond to true overlaps between molecules of two barcodes, or even have missing edges. We will describe the construction of another graph based on the BG: the local clique-pairs (lcp) graph. We will then use the lcp graph to identify a sequence of barcodes that reflects the true order of molecules.

The idea behind the lcp graph will be that, similarly to interval graphs, a realization of an exact barcode graph can be described as a succession of overlapping maximal cliques. Intuitively, these maximal cliques correspond to a set of barcodes that each contain at least one molecule coming from a common genomic region. The task is made more difficult by the fact that not all maximal cliques in a barcode graph satisfy this property. We observed that one can identify and skip such 'wrong' maximal cliques by instead considering a slightly more advanced object: pairs of co-localized maximal cliques, that we name local clique-pairs.

▶ **Definition 2.** *Let c be a vertex of a barcode graph BG. A neighbour of c is a vertex adjacent to c. The neighbourhood of c is the subgraph induced by the set of neighbours of c. A local clique-pair (lcp) is a triplet $(c; C_1, C_2)$ where C_1 and C_2 are maximal cliques in the neighbourhood of c. If there are k edges between vertices of C_1 and vertices of C_2 and d is the maximum number of vertices in either C_1 or C_2, the weight of the lcp (c, C_1, C_2) is defined by $w(c; C_1, C_2) = |d(d-1)/2 - k|$.* [2]

The definition of the weight of an lcp follows from the following observation: when molecules are all of the same length and are evenly spaced along the chromosome, if both cliques C_1 and C_2 are of size d and do indeed correspond to the d barcodes of the molecules preceding (resp. following) the molecule of barcode c, then one expects to observe $d(d-1)/2$ edges between them in the barcode graph. So the weight measures the divergence between the observed number of edges between C_1 and C_2 and the expected number of edges in the case of uniform sequencing (see Fig. 1).

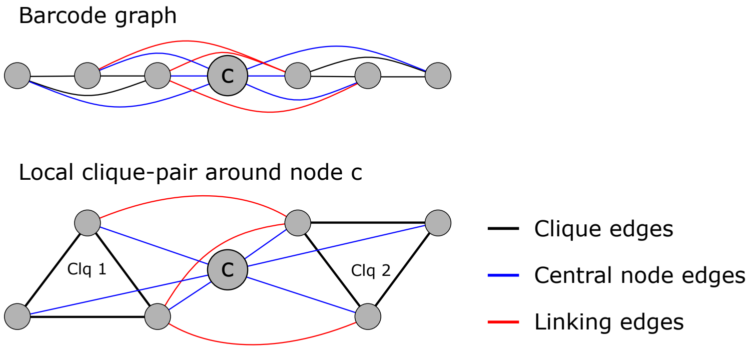


Figure 1 (Top) linear representation of a barcode graph obtained from 7 molecules of uniform length. (Bottom) The local clique-pair associated to c. In black, the edges from the side cliques of the unit 3-graph. In blue, the edges between the central nodes and the other nodes. In red, the edges between the cliques.

To motivate the introduction of lcps, we ran an experiment described in Appendix 5.1, showing that the rate of lcps that actually encode the barcodes of consecutive molecules is higher than the rate of maximal cliques having the same property (Table A1).

We now present our algorithm to compute lcps. Given a barcode c, there can be many maximal cliques among nodes in its neighbourhood, especially cliques that involve the two barcodes that respectively precede and follow c in the true barcode sequence. Given the

[2] The weight is presented for an ideal case where no node is shared between the cliques. If C_1 and C_2 share nodes, there are two modifications. For each node shared, 1 is added to the weight because the shared node is due to a barcode collision. Each shared edge between C_1 and C_2 counts for 2 additional points in the score instead of 1, because 2 edges can be merged inside.

set of all maximal cliques C in the neighbourhood of c, we thus need to extract a matching defining pairs of cliques C_1 and C_2 forming lcps. To do so, we consider the complete graph of size $|C|$ whose vertices are maximal cliques and edges are putative lcps. Edges are weighted by the previously-defined lcp weights. Let W be the maximum observed edge weight. We replace the weight w of each lcp by $W - w$ and apply a maximum-weight matching to clique pairs in order to obtain the set of lcps associated to c (Algorithm 1, illustrated in Fig. 2).

Algorithm 1 Determination of a set of lcps centered at a barcode c.

```
procedure COMPUTE_LCP($c, BG$)                        ▷ c: barcode, BG: barcode graph
    $LCP \leftarrow \emptyset$
    $ngbs \leftarrow BG.neighbours(c)$                              ▷ Neighbours of $c$
    $subgraph \leftarrow BG.induced\_subgraph(ngbs)$               ▷ Neighbourhood of $c$
    $cliques \leftarrow subgraph.max\_cliques()$              ▷ Enumerate maximal cliques
    $CG \leftarrow clique\_graph(cliques)$
    $m = CG.maximum\_weight\_matching()$
    for $(C_1, C_2) \in m$ do
        $LCP \leftarrow LCP \cup new\_lcp(c, C_1, C_2)$                 ▷ Add the new lcp
    return $LCP$
```

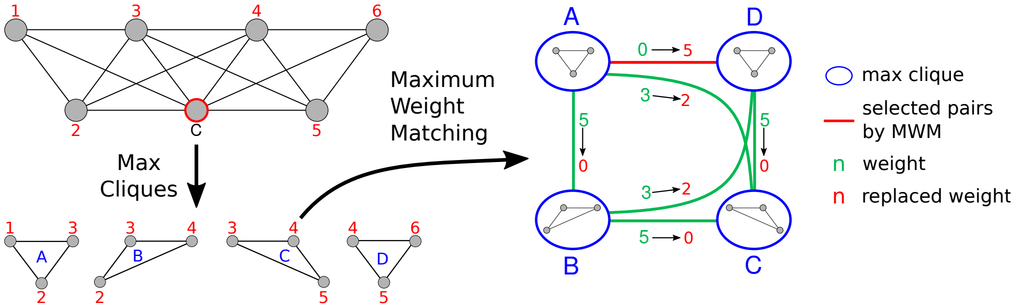


Figure 2 Top left: barcode graph; bottom left: max-cliques of the barcode graph; right: max-clique graph construction and maximum weight matching to construct lcps. The resulting maximum-weight matching is the edge A-D, yielding a single lcp with clique-pair (A, D).

The time complexity of enumerating all maximal cliques of a graph is exponential [26], while computing a maximum-weight matching is polynomial-time solvable [14]. We implemented Algorithm 1 in Python using the output-sensitive cliques enumeration and maximum-weight matching methods implemented in the Networkx library [17]. Its complexity is $O(\max(C^3, M(n)C))$ with n the number of graph nodes, C the number of maximal cliques in the graph, and $M(n)$ the cost of multiplying two $n \times n$ matrices.

Local search for linked cliques are akin to local graph community detection. Soft clustering is being performed with maximal clique detection, i.e. a node may belong to multiple communities. This property leads to a lcp detection algorithm that, intuitively, is resilient to the situation where a barcode corresponds to two or more molecules. Yet it is not perfect: some of the generated lcps may not reflect a collection of overlapping molecules (due to additional artifactual maximal cliques); and also, missing edges in the barcode graph may lead to missing lcps. In the ideal case, lcps can be totally ordered according to their overlaps. But because of artefactual and missing lcps, a total order is not always self-evident.

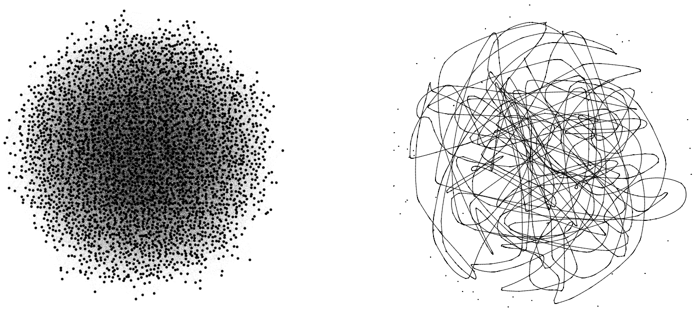


Figure 3 Left: barcode graph (3k nodes and 98k edges) of a simulated interval graph. Right: Resulting lcp graph (13k nodes and 23k edges). Graphs are drawn using Gephi, ForceAtlas 2 layout [3].

▶ **Definition 3.** *Let BG be a barcode graph and V a set of lcps obtained from BG. The lcp graph $lcp(BG)$ is the weighted graph with V for vertex set and where there is an edge between two lcps $(b; B_1, B_2)$ and $(c; C_1, C_2)$ such that some barcode belongs to both one of the C_i cliques and one of the B_i cliques. The weight of an edge is the size of the symmetric differences of the barcodes content of $(b; B_1, B_2)$ and $(c; C_1, C_2)$.*

The lcp graph is a framework for determining which lcps are consecutive, also enabling to identify lcps that are not overlapping with others. Figure 3 shows a simulated barcode graph where the corresponding lcp graph has a linear structure, similar to the original interval graph among molecules. This makes the task of finding a suitable path within the lcp graph, which reflects the ordering of molecules, easier than in the barcode graph. In the following, we will describe how we determine a barcode ordering based on finding a path in the lcp graph.

2.3 Finding a suitable path in the lcp graph

Recall that the molecule counting problem amounts to finding how many molecules were merged in each barcode. The molecule ordering problem asks for a sequence of barcodes that reflects the order of molecules. As these two problems are centered on barcodes and not lcps, we need a way to convert a lcp path into an ordered list of barcodes. We do this as follows: i) each lcp in the path is replaced by its central barcode, ii) an edge reduction step is applied to the lcp graph, and finally iii) a path is found using a branch-and-bound algorithm. Formally, the algorithmic problem we address heuristically in this section is to find a path in the lcp graph that maximizes the sum of the weight of the selected lcps and of the selected edges between lcps, under the constraint that the union of the selected lcps covering sets contains all edges of the initial barcode graphs.

lcp graph simplification

We simplify the lcp graph by performing transitive reduction over triplets. Given an edge (a, b) of weight w_{ab}, we remove this edge from the graph if there exist 2 edges (a, c) of weight w_{ac} and (b, c) of weight w_{bc} such that $w_{ab} \leqslant w_{ac} + w_{bc}$. This operation does not change the node set (lcps) but reduces the number of possible paths to explore. Intuitively, requiring to go through lcp c when going from lcp a to lcp b forces to select two higher-confidence lcp overlaps instead of one lower-confidence overlap between two lcps.

lcp path construction

Assuming the barcode graph has been obtained by merging nodes of an exact interval graph defined by the molecules intersections, every edge of the barcode graph corresponds to one (or potentially several) edges of the interval graph; we show in Section 3.3 that barcode graphs created from reads have nearly all correct edges corresponding to such molecules intersections. This observation motivates to require that a walk in the lcp graph that reflects the true order of molecules should be composed of lcps that contain most of the edges of the original barcode graph. Each lcp is an induced subgraph of the barcode graph, and we associate to it a *covering set* defined as the set of edges of the barcode graph it contains. We will seek a path such that the union of covering sets over all its constituent lcps is as close as possible to the set of all edges of the barcode graph.

Our lcp path construction strategy is a local branch & bound algorithm. Assuming we have already constructed a path of lcps $p = l_1, \dots, l_i$, we consider as candidates for l_{i+1} all the neighbours of l_i in the lcp graph such that $l_{i+1} \notin p$. Those neighbours are sorted by priority over three criteria: first if one or more lcp(s) cover at least one uncovered edge of the original barcode graph, we prioritize those lcps. For the second sorting criterion, we sort the candidate l_{i+1}'s by increasing lcp weight (Def. 3). Last, if multiple candidates have equal clique pair weights, we sort them by increasing $l_i \to l_{i+1}$ edge weight in the lcp graph. Selecting the first element in the sorted neighbours at each step defines a greedy heuristic for the path computation.

The above algorithm might result in a short path due to tips in the lcp graph, i.e. nodes of degree 1. In order to address this issue, we use a local branch and bound algorithm and backtrack a few nodes when a dead end is reached. This can result in several paths and we use the size of the union of covering sets in the path as a score to keep only the best solutions according to that score.

The last part of the algorithm is the selection of the first node l_1 of the path. We initially select a l_1 at random among all lcps, and compute a path using the above procedure which ends at some node l_e. We then discard this path and restart again our algorithm from $l'_1 = l_e$ to create a new path, where l_e has a higher chance to be an endpoint of the true lcp path than l_1.

We will show in the next section that despite this heuristic being very simple and likely leaving room for improvement, it does work very well on simulated data, suggesting the lcp graph does actually capture a robust signal toward recovering the correct barcode sequence.

3 Results

3.1 Overview

Simulated data

We will examine three types of barcode graphs ordered by increasing level of realism. They will be generated from either:

1. entirely synthetic sets of intervals (i.e. interval graphs) with randomly identified vertices,
2. intersections of molecules sampled from a genome,
3. directly from simulated linked-read sequencing data.

Analysis pipeline

Our complete analysis pipeline performs the following steps:

1. Generate all the lcps from the barcode graph (Algorithm 1)
2. Generate the lcp graph
3. Simplify the graph by transitive reduction of the triplets (Section 2.3)
4. Generate the lcp path using the hybrid greedy/branch and bound algorithm (Section 2.3)
5. Replace all the lcp by their central barcodes
6. Evaluate the accuracy of the resulting barcode sequence

In the remaining of the Results sections, all the graphs and paths are generated by the above pipeline, implemented using `Snakemake` [20] and available at `https://gitlab.pasteur.fr/ydufresne/linkedreadsmoleculeordering`.

Quality metrics

We design quality metrics that are applicable to both barcode graphs and lcp graphs. To do so, in lcp graphs we identify each lcp to its central barcode. We consider three metrics over the graphs: accuracy, sensitivity and longest correct path. The first two metrics are estimated by randomly sampling paths having $l \in \{2, 4, 10, 100\}$ edges from the graph. To measure accuracy, a path having barcodes $(b_0, b_1, \ldots, b_l)$ is considered to be correct if there exists $m_0, m_1, \ldots, m_l$ overlapping (but not necessarily consecutive) molecules such that $m_i \in m(b_i)$, $0 \leq i \leq l$. Accuracy is then defined as the number of correct paths over the total number of sampled paths. To measure sensitivity, we determined for all $(l+1)$-tuples $m_i, m_{i+1}, \ldots, m_{i+l}$ of consecutive molecules in the genome, whether there exist a path $b(m_i), b(m_{i+1}), \ldots, b(m_{i+l})$ in the graph. Sensitivity is then the ratio of such paths that are found in the graph. Finally, the Longest Correct path (LC) metric is defined as the longest path that can be found in the lcp graph that is correct, i.e. corresponding to a barcode sequence equal to the barcodes of a sequence of overlapping molecules. This measure is not informative on barcode graphs; it measures the conservation of molecule overlap information in a lcp graph.

Two additional quality metrics are defined on lcp paths found by our branch-and-bound algorithm: Undercounted/Overcounted (U/O) molecules and Longest Common Subsequence (LCS). The U/O metric is computed by recording two counters, U and O initialized at 0. Given each barcode b that appears within a lcp path, we compare the number of occurrences of b to M_b, the true number of molecules having barcode b. If b occurs in the lcp path strictly more (resp. less) than M_b times, U (resp. O) is incremented by the absolute difference. U and O should both be as close to zero as possible, and they indicate how well we solve the molecule counting problem. For the LCS metric, we compute the longest common subsequence between central nodes of the lcp path and the molecule path where each molecule is replaced by its barcode. The LCS reflects how well we solve the molecule ordering problem.

3.2 Simulated data from interval graphs

Dataset generation

At first we focus on purely synthetic interval graphs, where a genome is conceptually a real line and molecules are intervals on this line. We make the simplifying assumption that molecules all have the same size, and are evenly distributed along the genome. To simulate barcode graphs, we start from an intersection graph of molecules and perform so-called *merges* of molecules. A merge is defined as follows: given two nodes a and b that will be merged, create a new node c; for all neighbours v of either a or b, create edges (c, v), and finally delete a and b. Merging two nodes in the graph is equivalent to replacing two molecules by one barcode corresponding to those two molecules. A succession of merge operation creates an exact barcode graph as defined in Section 2.

We created 8 synthetic test datasets, using the following grid of parameters: $5,000$ or $10,000$ molecules, average number of merges (i.e. molecules per barcode) of 2 or 3, standard deviation in the number of merges of 0 or 1.

Quality of lcp graphs

Table 1 shows the accuracy and sensitivity of barcode graphs and their corresponding lcp graphs. Recall that accuracy measures whether a random path in the graph has a correct order of barcodes. As expected, paths in the barcode graph are mostly inaccurate, as one may jump from one genome location to another due to barcode merges. Conversely paths in the lcp graph are very accurate (100% for nearly all $l = 10$ paths), with a slight decrease at $l = 100$ ($95\% - 100\%$). The sensitivity metric measures how much of the true barcode ordering is present in short paths of the graph. It is (unsurprisingly) high for barcode graphs, as they indeed record all overlaps between molecules. Note that some merges collapse consecutive molecules by chance, hence the sensitivity of barcode graphs can sometimes be lower than 1. On lcp graphs, sensitivity is high for short paths ($> 93\%$ for $l = 10$) and drops for long ones ($54\% - 98\%$ for $l = 100$). Nevertheless, this shows that at least partial molecule order can be inferred through looking at central nodes of lcps in the lcp graph, and that the lcp graph shows a better balance between accuracy and sensitivity than the barcode graph. Note that central nodes are not the only way to infer molecule order, as one could also extract information from clique-pairs, yet we leave this direction for future work.

Overall, lcp graphs are clearly more informative than barcode graphs for reconstructing accurate barcode orderings. The hardest instances, in terms of accuracy and sensitivity on lcp graphs, are when the number of molecules is low and the number of merges is high.

Quality of lcp paths

Table 2 reports additional metrics on lcp graphs and lcp paths constructed using the branch-and-bound algorithm, over the same 8 datasets. All lcp graphs have a high longest correct path (LC), confirming the theoretical possibility of reconstructing over 99% of the true barcode order, through central nodes of a suitable path of lcps. The last two metrics of Table 2 are computed on lcp paths found by the algorithm described in Section 2.3. On 10,000 molecules graphs, the longest common subsequence (LCS) of the computed lcp path is 90% of the true barcode order, indicating that we nearly recovered the correct barcode order. The 5,000 molecules graphs appear to be more challenging to process as, smaller graphs are more sensitive to information loss by the merging process, yet LCS values remain above 79%. The U/O metric reports the ability to count the number of molecules that are present in each barcode, though counting the number of times each barcode occurs in the computed lcp path. Overall, lcp paths tend to undercount molecules (higher U metric than O), yet both U and O metrics are around or below 10% of the number of molecules, indicating that lcp path provides a reliable estimation of the number of molecules per barcode.

Table 1 Accuracy and sensitivity of randomly sampled paths of lengths 2, 4, 10 and 100 edges in lcp graphs generated from merged interval graphs, compared to sampled paths of the same lengths in barcode graphs (G_b) as a base-line.

Graph			**l=2**		**l=4**		**l=10**		**l=100**	
# mols	Merges	Type	Acc	Sens	Acc	Sens	Acc	Sens	Acc	Sens
5,000	2 ± 0	G_b	0.48	1.00	0.09	1.00	0.00	0.99	0.00	0.94
		lcp	1.00	1.00	1.00	0.99	1.00	0.98	1.00	0.88
5,000	2 ± 1	G_b	0.46	1.00	0.09	1.00	0.00	1.00	0.00	0.98
		lcp	1.00	1.00	1.00	0.99	1.00	0.98	0.99	0.84
5,000	3 ± 0	G_b	0.31	1.00	0.03	1.00	0.00	0.99	0.00	0.88
		lcp	1.00	0.99	1.00	0.98	0.99	0.95	0.99	0.60
5,000	3 ± 1	G_b	0.33	1.00	0.03	1.00	0.00	1.00	0.00	0.96
		lcp	1.00	0.99	1.00	0.97	0.99	0.93	0.95	0.54
10,000	2 ± 0	G_b	0.48	1.00	0.10	1.00	0.00	1.00	0.00	1.00
		lcp	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.98
10,000	2 ± 1	G_b	0.47	1.00	0.09	1.00	0.00	1.00	0.00	0.97
		lcp	1.00	1.00	1.00	1.00	1.00	0.99	0.99	0.93
10,000	3 ± 0	G_b	0.31	1.00	0.02	1.00	0.00	1.00	0.00	1.00
		lcp	1.00	1.00	1.00	0.99	1.00	0.99	1.00	0.87
10,000	3 ± 1	G_b	0.31	1.00	0.03	1.00	0.00	1.00	0.00	0.97
		lcp	1.00	0.99	1.00	0.99	1.00	0.97	0.99	0.78

Table 2 Experiments on synthetic barcode graphs. The dataset is described on the first part of the columns (Number of molecules in the molecule graph, number of merges, resulting number of barcodes in the barcode graph). The LC column is the length of the longest correct path in the lcp graph. The U/C column is the number of undercounted and overcounted molecules per barcode in our computed lcp path, and the LCS column is the length of the longest common subsequence between the lcp path and the correct barcode order.

# mols	Merges	# barcodes	LC	U/O Counts	LCS
5,000	2 ± 0	2500	4990	227/56	4748
5,000	2 ± 1	2428	4991	405/109	4512
5,000	3 ± 0	1667	4985	549/240	4282
5,000	3 ± 1	1682	4975	498/665	3972
10,000	2 ± 0	5000	9992	268/68	9667
10,000	2 ± 1	4889	9993	418/129	9531
10,000	3 ± 0	3334	9981	593/184	9309
10,000	3 ± 1	3341	9987	753/201	9140

Table 3 Accuracy and sensitivity of randomly sampled paths of lengths 2, 4, 10 and 100 edges in lcp graphs, compared to sampled paths of the same lengths in barcode graphs (G_b) as a base-line, with 15 kbp *E. coli* molecules, 50X coverage, minimal molecule overlap lengths of 7000.

	l=2		**l=4**		**l=10**		**l=100**	
Graph	Acc	Sens	Acc	Sens	Acc	Sens	Acc	Sens
$\boldsymbol{G_b, m=1}$	1	1	1	1	1	1	1	1
$\boldsymbol{lcp, m=1}$	1	1	1	0.99	1	0.99	1	0.84
$\boldsymbol{G_b, m=2}$	0.50	1	0.12	0.99	0.001	0.99	0	0.99
$\boldsymbol{lcp, m=2}$	0.99	0.99	0.99	0.99	0.99	0.99	0.94	0.84
$\boldsymbol{G_b, m=3}$	0.34	1	0.04	1	0	1	0	1
$\boldsymbol{lcp, m=3}$	0.99	0.99	0.99	0.99	0.98	0.99	0.88	0.88
$\boldsymbol{G_b, m=4}$	0.26	1	0.02	0.99	0	0.99	0	0.99
$\boldsymbol{lcp, m=4}$	0.99	0.99	0.99	0.99	0.98	0.99	0.83	0.87

3.3 Genome graphs

Quality of genome LCP graphs

We designed experiments to evaluate the quality of lcp graphs constructed from the barcode graphs that originate from real molecules. We created a synthetic *E.coli* molecule graph by simulating molecules of length 15 kbp using `wgsim`, corresponding to sequences of the *E. coli* genome, at 50x coverage of the genome and with no sequencing errors. Overlaps between all pairs of molecules were computed using `minimap2` using default parameters, and we selected overlaps of lengths greater than 7000 using `fpa` [22].

Table 3 shows the accuracy and sensitivity on our constructed lcp graphs versus the average number of merges, i.e. average number of molecules per barcode. As in Section 3.2, barcode graphs have poor accuracy, which is expected due to the glueing of molecules, and near-perfect sensitivity as all molecule overlaps are found. In contrast, lcp-graphs manage to keep both near-perfect accuracy and sensitivity (> 0.98) for short paths (< 10) and have a decrease in accuracy ($0.83 - 0.94$) and sensitivity ($0.84 - 0.87$) for paths of length 100.

Construction of genome barcode graphs from reads

In this section we describe a method that constructs an accurate barcode graph directly from linked-read data. This closes a gap between our theoretical results, that required to already have a barcode graph, and experimental data which only consist of sequencing reads.

We simulated reads from the *E. coli* genome at 50X coverage using LRSIM[21]. We assembled these reads using SPAdes[1] version 3.12.0 without using linked-read information (only using paired-end information), in order to generate contigs to which linked-reads can be mapped to using the EMA aligner[25]. We designed an algorithm[3] to infer molecule overlaps given the set of contigs and the EMA alignments. In brief, the algorithm proceeds as follows.

For each barcode, and within each contig, we collect and sort the mapping positions of all reads associated that barcode. We define a molecule interval to be the first and last mapping positions of a group of mapping positions that are all within a distance $< M_d$ than each other. A barcode can be associated to multiple molecule intervals even within the same contig. We construct the barcode graph by looking at overlapping molecule intervals from different barcodes. If two intervals share an overlap larger than a parameter M_o, we add an edge between the two associated barcodes.

[3] Available at `https://github.com/natir/mapping2barcode`

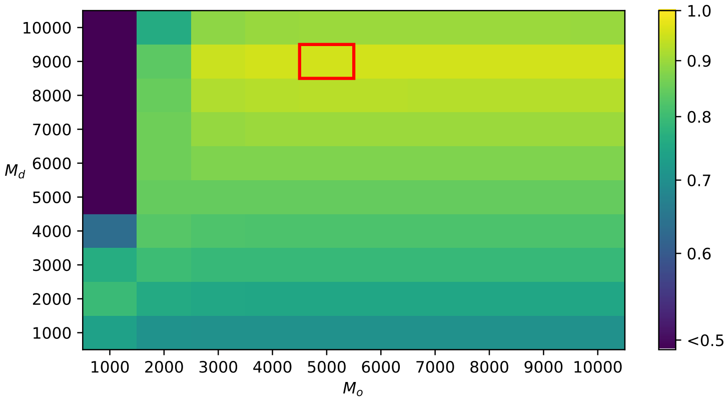


Figure 4 Quality of barcode graph construction from a set of reads and corresponding paired-end assembly. Each square represents the F1-score given parameters M_d (read mapping distance) and M_o (minimal molecule overlap length), from purple (low F1-score) to yellow (high F1-score).

The algorithm has two key parameters: M_d, the maximal distance between two reads in an inferred molecule interval, and M_o, the minimal overlap length between molecules. As we used simulated data, we were able to generate a ground-truth barcode graph given that molecule intervals in the underlying genome are known for each barcode. Figure 4 shows the performance of the algorithm in terms of F1-score (combining both sensitivity and precision, computed by comparing the edge set of the inferred barcode graph versus the edge set of the ground truth). We observe that the best F1-score (0.953) is reached for $(M_o, M_d) = (5000, 9000)$, with otherwise consistently high F1-scores (≥ 0.9) whenever $M_o > 2000$ and $M_d > 7000$.

4 Conclusion

In this paper, we introduced novel approaches to analyze linked-reads sequencing data. We introduced the problem of recovering a barcode sequence from the barcode graph, and described its link with natural algorithmic problems on multiple-interval graphs; we believe that the potential applications in sequencing data analysis motivate further research on these algorithmic questions. Moreover, motivated by classic algorithmic techniques in interval graph realization, we introduced the concept of local clique pairs (lcp) and lcp graph. Our experiments on simulated data suggests that the lcp graph exhibits a much more linear structure than the barcode graph and is likely a relevant intermediate structure between the barcode graph and the barcode sequence.

This work casts a spotlight on a couple open problems in graph theory, for which linked-reads bring an additional practical application: recognizing perfect barcode graphs, realizing multiple-interval graphs, and the complexity of recognizing unit f-interval graphs. We suspect also that more effective algorithms than the ones proposed here may exist for constructing lcp graphs and finding lcp paths.

While our treatment of synthetic barcode graphs demonstrates the feasibility of recovering barcode orders, we encountered difficulties going further in our analyses of realistic instances (e.g. real *E. coli* reads). First, a more advanced path(s) discovery procedure will be needed to deal with lcp graphs constructed from real molecule intersections (Section 3.3), which have inferior accuracy than those in Section 3.2. Second, refinements to Algorithm 1, potentially in the form of post-processing, will be needed to avoid outputting too many artefactual lcps in barcode graphs of high-coverage molecules, such as the one produced in Section 3.3.

Finally, the proposed barcode graph construction approach has potential to be applied to larger instances, but so far we only tested it on simulated data and is merely a proof of concept. The current method relies on having sufficient assembly contiguity (longer contigs than molecules). A potential direction that we leave for future work is to determine molecule intervals using the structure of an assembly graph instead of contigs.

References

1 Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A Gurevich, Mikhail Dvorkin, Alexander S Kulikov, Valery M Lesin, Sergey I Nikolenko, Son Pham, Andrey D Prjibelski, et al. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *J. Comput. Biol.*, 19(5):455–477, 2012. doi:10.1089/cmb.2012.0021.

2 Reuven Bar-Yehuda, Magnús M. Halldórsson, Joseph Naor, Hadas Shachnai, and Irina Shapira. Scheduling split intervals. *SIAM J. Comput.*, 36(1):1–15, 2006. doi:10.1137/S0097539703437843.

3 Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: An open source software for exploring and manipulating networks. In *Proceedings of the Third International Conference on Weblogs and Social Media, ICWSM 2009, San Jose, California, USA, May 17-20, 2009*. The AAAI Press, 2009. URL: http://aaai.org/ocs/index.php/ICWSM/09/paper/view/154.

4 Alex Bishara, Eli L Moss, Mikhail Kolmogorov, Alma E Parada, Ziming Weng, Arend Sidow, Anne E Dekas, Serafim Batzoglou, and Ami S Bhatt. High-quality genome sequences of uncultured microbes by assembly of read clouds. *Nat. Biotechnol.*, 36:1067–1075, 2018. doi:10.1038/nbt.4266.

5 Ivan Bliznets, Fedor V. Fomin, Marcin Pilipczuk, and Michal Pilipczuk. Subexponential parameterized algorithm for interval completion. *ACM Trans. Algorithms*, 14(3):35:1–35:62, 2018. doi:10.1145/3186896.

6 Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.

7 Ayelet Butman, Danny Hermelin, Moshe Lewenstein, and Dror Rawitz. Optimization problems in multiple-interval graphs. *ACM Trans. Algorithms*, 6:268–277, 2007. doi:10.1145/1721837.1721856.

8 Zhoutao Chen, Long Pham, Tsai-Chin Wu, Guoya Mo, Yu Xia, Peter Chang, Devin Porter, Tan Phan, Huu Che, Hao Tran, Vikas Bansal, Justin Shaffer, Pedro Belda-Ferre, Greg Humphrey, Rob Knight, Pavel Pevzner, Son Pham, Yong Wang, and Ming Lei. Ultra-low input single tube linked-read library method enables short-read NGS systems to generate highly accurate and economical long-range sequencing information for de novo genome assembly and haplotype phasing. *bioRxiv*, page 852947, 2019. doi:10.1101/852947.

9 David Coudert. A note on integer linear programming formulations for linear ordering problems on graphs. Research Report hal-01271838, INRIA, I3S, Université Nice Sophia, 2016. URL: https://hal.inria.fr/hal-01271838.

10 Christophe Crespelle, Paal Gronaas Drange, Fedor V. Fomin, and Petr A. Golovach. A survey of parameterized algorithms and the complexity of edge modification. *ArXiv*, abs/2001.06867, 2020. URL: https://arxiv.org/abs/2001.06867.

11 David C Danko, Dmitry Meleshko, Daniela Bezdan, Christopher Mason, and Iman Hajirasouliha. Minerva: an alignment and reference free approach to deconvolve linked-reads for metagenomics. *Genome Res.*, 29:116–124, 2019. doi:10.1101/gr.235499.118.

12 Michael R Fellows, Danny Hermelin, Frances A Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009.

13 Mathew C. Francis, Daniel Gonçalves, and Pascal Ochem. The maximum clique problem in multiple interval graphs. *Algorithmica*, 71(4):812–836, 2015. doi:10.1007/s00453-013-9828-6.

14 Zvi Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Comput. Surv.*, 18(1):23–38, 1986. doi:10.1145/6462.6502.

15 Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57).* North-Holland Publishing Co., 2004.

16 Stephanie U Greer, Lincoln D Nadauld, Billy T Lau, Jiamin Chen, Christina Wood-Bouwens, James M Ford, Calvin J Kuo, and Hanlee P Ji. Linked read sequencing resolves complex genomic rearrangements in gastric cancer metastases. *Genome Med.*, 9(1):57, 2017. doi:10.1186/s13073-017-0447-8.

17 Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11–15, Pasadena, CA, 2008.

18 Minghui Jiang. Recognizing d-interval graphs and d-track interval graphs. *Algorithmica*, 66(3):541–563, 2013. doi:10.1007/s00453-012-9651-5.

19 Johannes Köbler, Sebastian Kuhnert, and Osamu Watanabe. Interval graph representation with given interval and intersection lengths. *J. Discrete Algorithms*, 34:108–117, 2015. doi:10.1016/j.jda.2015.05.011.

20 Johannes Köster and Sven Rahmann. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, 2012.

21 Ruibang Luo, Fritz J Sedlazeck, Charlotte A Darby, Stephen M Kelly, and Michael C Schatz. LRSim: a linked-reads simulator generating insights for better genome partitioning. *Comput Struct Biotechnol J.*, 15:478–484, 2017. doi:10.1016/j.csbj.2017.10.002.

22 Pierre Marijon, Rayan Chikhi, and Jean-Stéphane Varré. yacrd and fpa: upstream tools for long-read genome assembly. *Bioinformatics*, advance access:btaa262, 2020. doi:10.1093/bioinformatics/btaa262.

23 Ross M. McConnell. Linear-time recognition of circular-arc graphs. *Algorithmica*, 37(2):93–147, 2003. doi:10.1007/s00453-003-1032-7.

24 Itsik Pe'er and Ron Shamir. Realizing interval graphs with size and distance constraints. *SIAM J. Discret. Math.*, 10(4):662–687, 1997. doi:10.1137/S0895480196306373.

25 Ariya Shajii, Ibrahim Numanagić, and Bonnie Berger. Latent variable model for aligning barcoded short-reads improves downstream analyses. In *Research in Computational Molecular Biology - 22nd Annual International Conference, RECOMB 2018*, volume 10812 of *Lecture Notes Comput. Sci.*, pages 280–282. Springer, 2018. doi:10.1007/978-3-319-89929-9.

26 Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theor. Comput. Sci.*, 363(1):28–42, 2006. doi:10.1016/j.tcs.2006.06.015.

27 Yngve Villanger, Pinar Heggernes, Christophe Paul, and Jan Arne Telle. Interval completion is fixed parameter tractable. *SIAM J. Comput.*, 38(5):2007–2020, 2009. doi:10.1137/070710913.

28 Ou Wang, Robert Chin, Xiaofang Cheng, Michelle Ka Yan Wu, Qing Mao, Jingbo Tang, Yuhui Sun, Ellis Anderson, Han K. Lam, Dan Chen, Yujun Zhou, Linying Wang, Fei Fan, Yan Zou, Yinlong Xie, Rebecca Yu Zhang, Snezana Drmanac, Darlene Nguyen, Chongjun Xu, Christian Villarosa, Scott Gablenz, Nina Barua, Staci Nguyen, Wenlan Tian, Jia Sophie Liu, Jingwan Wang, Xiao Liu, Xiaojuan Qi, Ao Chen, He Wang, Yuliang Dong, Wenwei Zhang, Andrei Alexeev, Huanming Yang, Jian Wang, Karsten Kristiansen, Xun Xu, Radoje Drmanac, and Brock A. Peters. Efficient and unique cobarcoding of second-generation sequencing reads from long DNA molecules enabling cost-effective and accurate sequencing, haplotyping, and de novo assembly. *Genome Res.*, 29(5):798–808, 2019. doi:10.1101/gr.245126.118.

29 Neil I Weisenfeld, Vijay Kumar, Preyas Shah, Deanna M Church, and David B Jaffe. Direct determination of diploid genome sequences. *Genome Res.*, 27, 2017. doi:10.1101/gr.214874.116.

30 Douglas B. West and David B. Shmoys. Recognizing graphs with fixed interval number is NP-complete. *Discret. Appl. Math.*, 8(3):295–305, 1984. doi:10.1016/0166-218X(84)90127-6.

31 Sarah Yeo, Lauren Coombe, René L Warren, Justin Chu, and Inanç Birol. ARCS: scaffolding genome drafts with linked reads. *Bioinformatics*, 34(5):725–731, 2017. doi:10.1093/bioinformatics/btx675.

32 Fan Zhang, Lena Christiansen, Jerushah Thomas, Dmitry Pokholok, Ros Jackson, Natalie Morrell, Yannan Zhao, Melissa Wiley, Emily Welch, Erich Jaeger, Ana Granat, Steven J. Norberg, Aaron Halpern, Maria C Rogert, Mostafa Ronaghi, Jay Shendure, Niall Gormley, Kevin L. Gunderson, and Frank J. Steemers. Haplotype phasing of whole human genomes using bead-based barcode partitioning in a single tube. *Nat. Biotechnol.*, 35(9):852–857, September 2017. doi:10.1038/nbt.3897.

5 Appendix

5.1 Experiment on perfect cliques versus perfect lcps

Given an interval graph with m vertices, and an integer parameter f, we repeated for each vertex x the following process f times: pick another unmerged node y at random and merge vertices x and y. This generates a simulated barcode graph where all barcodes correspond to exactly f molecules. Then on this graph we computed all maximal cliques and all lcps and call such a set of vertices *perfect* if it corresponds to a set of consecutive intervals in the original interval graph.

Table A1 Average number of perfect maximal clique vs perfect lcps, averaged over 10 runs for each setting defined by m and f.

m	f	cliques	perfect cliques	lcp	perfect lcp
5000	2	5318.5	54645.2 (9.73%)	4600.2	12763 (36.04%)
5000	3	6361.8	76569.4 (8.31%)	4054.3	29924.8 (13.55%)
10000	2	10344.8	104817.5 (9.87%)	9586.2	18958.4 (50,56%)
10000	3	12070.6	130092.4 (9.28%)	9031.2	44276 (20.40%)

Exact Transcript Quantification Over Splice Graphs

Cong Ma[1]
Computational Biology Department, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
congm1@andrew.cmu.edu

Hongyu Zheng[1]
Computational Biology Department, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
hongyuz1@andrew.cmu.edu

Carl Kingsford[2]
Computational Biology Department, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
carlk@cs.cmu.edu

Abstract

The probability of sequencing a set of RNA-seq reads can be directly modeled using the abundances of splice junctions in splice graphs instead of the abundances of a list of transcripts. We call this model graph quantification, which was first proposed by Bernard et al. (2014). The model can be viewed as a generalization of transcript expression quantification where every full path in the splice graph is a possible transcript. However, the previous graph quantification model assumes the length of single-end reads or paired-end fragments is fixed. We provide an improvement of this model to handle variable-length reads or fragments and incorporate bias correction. We prove that our model is equivalent to running a transcript quantifier with exactly the set of all compatible transcripts. The key to our method is constructing an extension of the splice graph based on Aho-Corasick automata. The proof of equivalence is based on a novel reparameterization of the read generation model of a state-of-art transcript quantification method. This new approach is useful for modeling scenarios where reference transcriptome is incomplete or not available and can be further used in transcriptome assembly or alternative splicing analysis.

2012 ACM Subject Classification Applied computing → Computational transcriptomics

Keywords and phrases RNA-seq, alternative splicing, transcript quantification, splice graph, network flow

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.12

Supplementary Material The source code is available at `https://github.com/Kingsford-Group/subgraphquant`.

Funding This work was partially supported in part by the Gordon and Betty Moore Foundation's Data-Driven Discovery Initiative through Grant GBMF4554, by the US National Science Foundation (DBI-1937540), by the US National Institutes of Health (R01GM122935), and by The Shurl and Kay Curci Foundation. This project is funded, in part, under a grant (#4100070287) with the Pennsylvania Department of Health. The Department specifically disclaims responsibility for any analyses, interpretations or conclusions.

Acknowledgements We would also like to thank Natalie Sauerwald, Dr. Guillaume Marçais, Xiangrui Zeng and Dr. Jose Lugo-Martinez for insightful comments on the manuscript. C.K. is a co-founder of Ocean Genomics, Inc.

[1] Equal Contribution
[1] Equal Contribution
[2] Corresponding Author

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti; Article No. 12; pp. 12:1–12:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Transcript quantification has been a key component of RNA-seq analysis pipelines, and the most popular approaches (such as RSEM [9], kallisto [4], and Salmon [12]) estimate the abundance of individual transcripts by inference over a generative model from transcripts to observed reads. To generate a read in the model, a transcript is first sampled proportional to its relative abundance multiplied by length, then a fragment is sampled as a subsequence of the transcript according to bias correction models. The quantification algorithm thus takes the reference transcriptome and the set of reads as input and outputs a most probable set of relative abundances under the model. We focus on a generalization of the problem, called graph quantification, that allows for better handling of uncertainty in the reference transcriptome.

The concept of graph quantification was first proposed by Bernard et al. [3], which introduced a method called FlipFlop. Instead of a set of linear transcripts, a splice graph is given and every transcript compatible with the splice graph (a path from transcript start to termination in the splice graph) is assumed to be able to express reads. The goal is to infer the abundance of edges of the splice graph (or its extensions) under flow balance constraints. Transcript abundances are obtained by flow decomposition under this setup. FlipFlop infers network flow on its extension of splice graphs, called fragment graphs, and uses the model to further assemble transcripts. However, the proposed fragment graph model only retains its theoretical guarantee when the lengths of single-end reads or paired-end fragments are fixed. In this work, we propose an alternative approach to graph quantification that correctly addresses the variable-length reads and corrects for sequencing biases. Our method is based on flow inference on a different extension of the splice graph.

Modeling RNA-seq reads directly by network flow on splice graphs (or variants) is advantageous when the set of transcript sequences is uncertain or incomplete. It is unlikely that the set of reference transcripts is correct and complete for all genes in all tissues, and therefore, many transcriptome assembly methods have been developed for reconstructing a set of expressed transcripts from RNA-seq data [18, 13, 10, 14], including FlipFlop [3]. Recent long read sequencing confirms the expression of unannotated transcripts [17], but they also show that the individual exons and splice junctions are relatively accurate. With incomplete reference transcripts but correct splice graphs, it is more appropriate to model RNA-seq reads directly by splice graph network flows compared to modeling using the abundances of an incomplete set of transcripts.

The network flow of graph quantification may be incorporated into other transcriptome assembly methods in addition to FlipFlop. StringTie [13] iteratively finds the heaviest path of a flow network constructed from splice graphs. A theoretical work by Shao et al. [15] studies the minimum path decomposition of splice graphs when the edge abundances satisfy flow balance constraints. Better network flow estimation on splice graphs inspires improvement of transcriptome assembly methods.

The splice graph flow itself is biologically meaningful as it indicates the relative usage of splice junctions. Estimates of these quantities can be used to study alternative splicing patterns under the incomplete reference assumption. PSG [8] pioneered this line of work but with a different abundance representation in splice graph. It models splice junction usage by fixed-order Markov transition probabilities from one exon (or fixed number of predecessor exons) to its successor exon in the splice graph. It develops a statistical model to detect the difference in transition probability between two groups of samples. However, fixed-order Markov chain is less expressive: a small order cannot capture long-range phasing relationships, and a large order requires inferring a number of transition probabilities that are likely to lack sufficient read support. Markov models set the abundance of a transcript to the product of transition probabilities of its splice junctions, which implicitly places a strong constraint on the resulting transcriptome. Many other previous studies of splice junction usage depend on a list of reference transcripts and compute the widely used metric Percentage Spliced In (PSI) [7, 16, 19]. Under an incomplete reference assumption, the estimated network flow is a potential candidate to compute PSI and study alternative splicing usage.

A key challenge of graph quantification, especially for paired-end reads, is to incorporate the co-existence relationship among exons in transcripts. When a read spans multiple exons, the exons must co-exist in the transcript that generates this read. Such a co-existence relationship is called phasing, and the corresponding read is said to contain phasing information. For these reads, the flows of the spanned splice edges may be different from each other, and in this case, the probability of the read cannot be uniquely inferred from the original splice graph flow. FlipFlop solves this problem by expanding the splice graph into a fragment graph, assuming all reads are fixed-length. In a fragment graph, every vertex represents a phasing path, two vertices are connected if the phasing paths represented by the vertices differ by one exon, and every transcript on the splice graph maps to a path on the fragment graph. The mapped path in the fragment graph contains every possible phasing path from a read in the transcript, in ascending order of genomic location. However, it is not possible to construct this expansion of splice graphs when the reads or fragments are of variable lengths. There is no longer a clear total order over all phasing paths possible from a given transcript, and it is unclear how to order the phasing paths in a fragment graph. We detail the FlipFlop model in Section A.3.

To incorporate the phasing information from variable-length reads or fragments, we develop a dynamic unrolling technique over the splice graph with an Aho-Corasick automaton. The resulting graph is called prefix graph. We prove that optimizing network flow on the prefix graph is equivalent to the state-of-the-art transcript expression quantification formulation when all full paths of splice graphs are provided as reference transcripts, assuming modeled biases of generating a fragment are determined by the fragment sequence itself regardless of which transcript it is from. In other words, quantification on prefix graphs generates exact quantification for the whole set of full splice graph paths. The proof is done by reparameterizing the sequencing read generation model from transcript abundances to edge abundances in the prefix graph. We also propose a specialized EM algorithm to efficiently infer a prefix graph flow that solves the graph quantification problem.

As a case study, we apply our method on paired-end RNA-seq data of bipolar disease sequencing samples and estimate flows for neurogenesis-related genes, which are known to have complex alternative splicing patterns and unannotated isoforms. We use this case study to demonstrate the applicability of our method to handle variable-length fragments. Additionally, the network flow leads to different PSI compared to the one computed with reference transcripts, suggesting reference completeness should be considered in alternative splicing analysis.

2 Methods

We now provide a brief technical overview of the method section.

In Section 2.1, we describe the detailed derivation and procedure to reparameterize the generative model in transcript quantification. A key component in this process is re-defining transcript effective length. The transcript effective length is introduced to offset sampling biases towards shorter transcripts, and an empirical formula by penalizing transcript length with average fragment length has been widely used. We show that this empirical formula has a more elegant explanation, purely from introspection of the generative model. Based on this observation, we naturally introduce the path abundances, the new set of variables that parameterize the generative model, and the path effective lengths, weights of the path abundances in the normalization constraints. To introduce bias correction, we introduce the concept of affinity that encodes bias corrected likelihood for generating a fragment at a particular location, and the rest follows naturally by redefining the effective lengths.

In Section 2.2, we describe the prefix graph, whose purpose is to map the abundances of compatible transcripts onto network flows that preserve path abundances. This is beneficial, as we avoid enumerating compatible transcripts and only need to infer the prefix graph flow. The key technical contribution in this section is connecting the process of matching phasing paths onto transcripts, to the general problem of multi-pattern matching. This leads to a rollout of the splice graph according to an Aho-Corasick automaton, and the correctness (that the flow preserves of path abundances) can be proved by running the Aho-Corasick algorithm on the compatible transcripts.

In Section 2.3, we describe the inference process for the prefix graph flows, as we need to expand our model to handle multi-mapped reads within a gene or across different genes. We employ a standard EM algorithm for multi-mapped reads, similar to existing approaches. Inference across genes is enabled by another reparameterization of the generative model, which relativizes edge abundances to its incident gene. We are able to decouple the inference for each gene during the M-step, which combined with a simple E-step, allows for efficient inference and completes the specification of our methods.

We formally define the following terms. A **splice graph** is a directed acyclic graph representing alternative splicing events in a gene. The graph has two special vertices: S represents the start of transcripts and T represents the termination of transcripts. Every other vertex represents an exon or a partial exon. Edges in the splice graph represent **splice junctions**, potential adjacency between the exons in transcripts, or connect two adjacent partial exons. A **path** is a list of vertices such that every adjacent pair is connected by an edge, and an $S-T$ path is a path that starts with S and ends with T. We refer to **phasing paths** as the paths of which the exons co-exist in some transcripts. Specifically, we use a generalized notion of phasing path that includes singleton path (path of a single vertex) and path consisting of a single edge, so all vertices and edges are considered phasing paths. Each transcript corresponds to a unique $S-T$ path in the splice graph, and as discussed in the introduction, we will assume every $S-T$ path is also a transcript. Graph quantification generalizes transcript quantification as we can set up a “fully rolled out” splice graph, containing only chains each corresponding to a linear transcript. We use the phrase **quantified transcript set** to denote a set of transcripts with corresponding abundances.

2.1 Reparameterization

Our goal in this section is to establish an alternative set of parameters for the graph quantification problem. In the transcript quantification model, every transcript corresponds to a variable denoting its relative abundance. We will identify a more compact set of parameters that would represent the same model, as described below.

We start with the core model of transcript quantification at the foundation of most modern methods [9, 6, 4, 12]. Assume the paired-end reads from an RNA-seq experiment are error-free and uniquely aligned to a reference genome with possible gaps as fragments (assumptions will be relaxed later). We denote the set of fragments (mapped from paired-end reads) as F, the set of transcripts as $\mathcal{T} = \{T_1, T_2, \dots, T_n\}$ with corresponding lengths $l_1, l_2, \dots, l_n$ and abundances (copies of molecules) $c_1, c_2, \dots, c_n$. This can be used to derive other quantities, for example, the transcripts per million (TPM) values are calculated by normalizing $\{c_i\}$ then multiplying the values by 10^6. Under the core model, the probability of observing F is:

$$P(F \mid \mathcal{T}, c) = \prod_{f \in F} \sum_{i \in \mathrm{idx}(f)} P(T_i) P(f \mid T_i).$$

Here, $P(T_i)$ denotes the probability of sampling a fragment from transcript T_i, and $P(f \mid T_i)$ denotes the probability of sampling the fragment f given it comes from T_i. $\mathrm{idx}(f)$ is the set of transcript indices onto which f can map. Let $D(l)$ be the distribution of generated fragment length. In the absence of bias correction, $P(f \mid T_i)$ is proportional to $D(f) = D(l(f))$, where $l(f)$ denotes the fragment length inferred from mapping f to T_i. Define the effective length for T_i as $\hat{l}_i = \sum_{j=1}^{l_i} \sum_{k=j}^{l_i} D(k-j+1)$ (which can be interpreted as the total "probability" for T_i to generate a fragment), and $P(f \mid T_i) = D(f)/\hat{l}_i$. The probability of generating a fragment from T_i is assumed to be proportional to its abundance times its effective length, meaning $P(T_i) \propto c_i \hat{l}_i$. Our definition of effective length is different from existing literature, where it is usually defined as $l_i - \mu(T_i)$, the actual length of transcript l_i minus the truncated mean of D, and the truncated mean is defined as $\mu(T_i) = (\sum_{j=1}^{l_i} jD(j))/(\sum_{k=1}^{l_i} D(k))$. However, these two definitions are actually essentially the same most of the time:

▶ **Lemma 1.** $\hat{l}_i = \sum_{j=1}^{l_i} \sum_{k=j}^{l_i} D(k-j+1) = (\sum_{t=1}^{l_i} D(t))(l_i + 1 - \mu(T_i))$.

Proof.

$$\begin{aligned} \hat{l}_i &= \sum_{t=1}^{l_i} D(t)(l_i + 1 - t) \\ &= (l_i + 1) \sum_{t=1}^{l_i} D(t) - \sum_{t=1}^{l_i} tD(t) \\ &= \left(\sum_{t=1}^{l_i} D(t)\right) \left(l_i + 1 - \frac{\sum_{t=1}^{l_i} tD(t)}{\sum_{t=1}^{l_i} D(t)}\right) \\ &= \left(\sum_{t=1}^{l_i} D(t)\right) (l_i + 1 - \mu(T_i)) \end{aligned}$$

This means ignoring the difference between l_i and $l_i + 1$, the two definitions differ by a multiplicative factor of $\sum_{t=1}^{l_i} D(t)$. ◀

The factor $\sum_{t=1}^{l_i} D(t)$ is the probability of sampling a fragment no longer than l_i. It is very close to 1 as long as the transcript is longer than most fragments, which is usually true in practice. We refer to previous papers [9, 11, 6, 4, 12] for more detailed explanation of the model. This leads to:

$$P(F \mid \mathcal{T}, c) = \prod_{f \in F} (\sum_{i \in \mathrm{idx}(f)} c_i) D(f) / (\sum_{T_i \in \mathcal{T}} c_i \hat{l}_i).$$

We now propose an alternative view of the probabilistic model with paths on splice graphs in order to derive a compact parameter set for the quantification problem. The splice graph is constructed so each transcript can be uniquely mapped to an $S-T$ path $p(T_i)$ on the graph, and we assume the read library satisfies that each fragment f can be uniquely mapped to a (non $S - T$) path $p(f)$ on the graph (this assumption will also be relaxed later). With this setup, $i \in \text{idx}(f)$ if and only if $p(f)$ is a subpath of $p(T_i)$, or $p(f) \subset p(T_i)$.

We now define $c_p = \sum_{i:T_i \in \mathcal{T}, p \subset p(T_i)} c_i$ to be the total abundance of transcripts including path p, called **path abundance**, and $\hat{l}_p = \sum_{j=1}^{l_i} \sum_{k=j}^{l_i} \mathbf{1}(p(T_i[j,k]) = p) D(k - j + 1)$ called **path effective length**, where $T_i[j,k]$ is the fragment generated from transcript i from base j to base k and $\mathbf{1}(\cdot)$ is the indicator function. Intuitively, the path effective length is the total probability of sampling a fragment that maps exactly to the given path. This definition is independent of the chosen transcript T_i and any T_i yields the same result as long as T_i includes p. Next, let $\mathcal{P}$ be the set of paths from the splice graph satisfying $\hat{l}_p > 0$.

▶ **Lemma 2.** *The normalization term can be reparameterized:* $\sum_{T_i \in \mathcal{T}} c_i \hat{l}_i = \sum_{p \in \mathcal{P}} c_p \hat{l}_p$.

Proof. The idea is to break down the expression of $\hat{l}_i$ into a sum over fragments, and regroup the fragments by the path to which they are mapped:

$$\begin{aligned}
\sum_{T_i \in \mathcal{T}} \hat{l}_i c_i &= \sum_{T_i \in \mathcal{T}} \sum_{j=1}^{l_i} \sum_{k=j}^{l_i} D(k - j + 1) c_i \\
&= \sum_{p \in \mathcal{P}} \sum_{i,j,k:p(T_i[j,k])=p} D(k - j + 1) c_i \\
&= \sum_{p \in \mathcal{P}} (\sum_{j,k:\exists i, p(T_i[j,k])=p} D(k - j + 1))(\sum_{i:p \subset p(T_i)} c_i) \\
&= \sum_{p \in \mathcal{P}} \hat{l}_p c_p
\end{aligned}$$

The third equation holds because the sum of $D(k - j + 1)$ across any transcripts containing path p is the same, as a shift in the reference does not change $D(k - j + 1)$ assuming there are no sequencing biases. ◀

The likelihood objective can now be rewritten as:

$$\begin{aligned}
P(F \mid \mathcal{T}, c) &= \prod_{f \in F} (\sum_{j:p(f) \subset p(T_j)} c_j) D(f) / (\sum_{p \in \mathcal{P}} c_p \hat{l}_p) \\
&\propto \prod_{f \in F} c_{p(f)} / (\sum_{p \in \mathcal{P}} c_p \hat{l}_p) \qquad (1)
\end{aligned}$$

This reparameterizes the model with $\{c_p\}$, the path abundance. In practice, we reduce the size of $\mathcal{P}$ by discarding long paths with small $\hat{l}_p$ and no mapped fragments, as they contribute little to the likelihood (see Section A.2). To incorporate bias correction into our model, we define the affinity $A_p(j,k)$ to be the unnormalized likelihood of generating a read pair mapped to path p from position j to k. This is the analog for $P(f \mid t_i)$ in the transcript quantification model. In the non-bias-corrected model, we simply have $A_p(j,k) = D(k - j + 1)$. Certain motif-based corrections and GC-content-based corrections, which are calculated from the genomic sequence in between the paired-end alignment, can then be integrated into our analysis naturally. To adapt the likelihood model to bias correction, we define transcript and path effective length as follows:

$$\hat{l}_i = \sum_{j=1}^{l_i} \sum_{k=j}^{l_i} A_{p(T_i[j,k])}(j,k)$$

$$\hat{l}_p = \sum_{j=1}^{l_i} \sum_{k=j}^{l_i} A_p(j,k) \mathbf{1}(p(T_i[j,k]) = p), \forall p \subset p(T_i)$$

$\hat{l}_p$ is still the same for any T_i that includes p, so it does not matter which transcript is used to compute it. $p(T_i[j,k])$ denotes the path that $T_i[j,k]$ (transcript T_i from location j to k) maps to, and we assume the coordinate when calculating A_p coincides with that of T_i. The definition of path abundance remains unchanged, and all of our proposed methods will work in the same way. Transcript-specific bias correction requires an approximation to the affinity term, and we discuss this topic in detail in Section A.1.

We have now completed the necessary steps to claim the following theorem, which formally establishes the correctness of the reparameterization procedure with bias correction:

▶ **Theorem 3.** *Assuming each read is uniquely mapped to one phasing path, the following two optimization instances are equivalent:*

- *optimizing* $\{c_p\}$, *which are the path abundances under the reparameterized objective* $\prod_{f\in F} c_{p(f)}/(\sum_{p\in\mathcal{P}} c_p\hat{l}_p)$, *conditioned on* $\{c_p\}$ *corresponding to a valid quantified set of transcripts;*
- *optimizing* $\{c_i\}$ *which are the transcript abundances under the original objective* $\prod_{f\in F}(\sum_{i\in idx(f)} c_i)/(\sum_{T_i\in\mathcal{T}} c_i\hat{l}_i)$.

Here $\hat{l}_i$ *and* $\hat{l}_p$ *are transcript effective length and path effective length defined with the same set of affinities* $A_p(j,k)$.

Proof. This naturally follows in two steps. First, we can prove Lemma 2 with bias correction using the identical technique of breaking $\hat{l}_i$ down to sum over fragments, then regroup by path mappings. This means the normalization term can be reparameterized. We finish by reparameterizing the whole likelihood in the same way as in the non-bias-corrected case (see equation (1)), again with identical technique. ◀

With reads multimapped to different phasing paths (within or across genes), let $M(f)$ denote the set of phasing paths f can map onto, and for $p \in M(f)$ let $A(f \mid p)$ denote the affinity of f mapping to p. In this case, we can use the same idea of grouping transcripts by the phasing path that f maps onto:

$$\begin{aligned}
P(f) &= \sum_{i\in \mathrm{idx}(f)} P(T_i)P(f \mid T_i)\\
&= \sum_{p\in M(f)} \sum_{i:p\subset T_i} c_i A(f \mid p)\\
&= \sum_{p\in M(f)} c_p A(f \mid p).
\end{aligned}$$

The reparameterization theorem holds by replacing $c_{p(f)}$ with $\sum_{p\in M(f)} c_p A(f \mid p)$ in the objective function.

2.2 Prefix Graphs

In Theorem 3, we showed that to perform graph quantification, it is sufficient to optimize the path abundances under a reparameterized objective, conditioned on that the path abundances correspond to a quantified set of transcripts. This means to apply the theorem for optimization of path abundance, we need a set of constraints that ensures this condition. One solution is to introduce a variable for every compatible transcript and then use the definition of c_p as the constraints. However, this will lead to an impractically large model, as the number of $S-T$ paths in the splice graph can be exponentially larger than the size of the prefix graph. In this section, we derive a set of linear constraints governing $\{c_p\}$ that achieves this purpose.

To motivate the next step, assume every inferred fragment either resides within an exon or contains one junction. In this case, the phasing paths are nodes or edges in the splice graph. If the quantified transcript set is mapped onto the splice graph, we obtain a network flow. The path abundance for a phasing path equals either the flow through a vertex or an edge. By the flow decomposition theorem, given a network flow on the splice graph, we can decompose it into $S-T$ paths with weights, which then naturally maps back to a quantified transcript set. As the two-way mapping (between quantified transcript sets and splice graph flows) preserves path abundances, we conclude optimization over a splice graph flow would achieve the goal of graph quantification. Specifically, it is easy to restructure the constraints to represent a splice graph flow, and optimizing the resulting model is equivalent to the transcript quantification model with all compatible transcripts included.

This solution no longer works when some phasing path p contains three or more exons. This is because one cannot determine the total flow that goes through two consecutive edges (corresponding to a phasing path with two junctions) just from the flow graph, and different decompositions of the flow lead to different answers. Informally, this can be solved by constructing higher-order splice graphs (as done by Legault et al. [8] for example), or fixed-order Markov models, but the size of the resulting graph grows exponentially fast and some phasing paths can be very long. Instead, we choose to "unroll" the graph just as needed, roughly corresponding to a variable-order Markov model, similar to FlipFlop [3] but applicable to paired-end reads.

To motivate our proposed unrolling method, consider the properties it needs to satisfy. Roughly speaking, the unrolled graph needs to exactly identify every path in $\mathcal{P}$ to accurately calculate the path abundances. That is, for every path p in $\mathcal{P}$, there is a set of vertices or edges in the unrolled graph, such that a transcript includes p if and only if its corresponding $S-T$ path intersects with this set. We can view this "identify phasing paths" problem as an instance of multiple pattern matching. That is, given $\mathcal{P}$, for a given transcript T_i, we want to determine the set of paths in $\mathcal{P}$ that are subpaths of T_i, reading one exon of T_i at a time. Similar to our previous example, if $\mathcal{P}$ contains only single exons, we only need to recognize $[x]$ (the singleton path including only x) when we read exon x, and we will recognize a general phasing path p when the transcript we have seen admits p as a suffix. To speed up the process, we can memorize a suffix of the transcript we have seen that is a prefix of some path in p, so we do not need to check all preceding exons again when trying to recognize p. This is not a new idea and in fact is the Aho-Corasick algorithm [1], a classical algorithm for multiple pattern matching where the nodes in the splice graph (set of exons) is the alphabet, $\mathcal{P}$ is the set of patterns and T_i is the text, and the idea is formalized as a finite state automaton (FSA) that maintains the longest suffix of current text that could extend and match a pattern in the future. This can be regarded as an unrolling of the splice graph, which has the power of exactly matching arbitrarily phasing paths, and a flow on the automaton is the analog of a splice graph flow that also is unrolled enough to recover path abundances, as we will prove in this section.

We formalize the idea. Consider the Aho-Corasick FSA constructed from $\mathcal{P}$, where we further modify the finite state automaton as follows. Transitions between states of the FSA, called dictionary suffix links, indicate the next state of the FSA given the current state and the upcoming character. We do not need the links for all characters (exons), as we know $T_i \in \mathcal{T}$ is an $S-T$ path on the splice graph. If x is the last seen character, the next character y must be a successor of x in the splice graph, and we only generate the state transitions for this set of movements. With an FSA, we now construct a directed graph from its states and transitions as described above:

▶ **Definition 4** (Prefix Graph). *Given splice graph G_S and set of splice graph paths $\mathcal{P}$ (assuming every single-vertex path is in $\mathcal{P}$), we construct the corresponding prefix graph G as follows:*

The vertices V of G are the splice graph paths p such that p is a prefix of some path in $\mathcal{P}$. For $p \in V$, let x be the last exon in p. For every y that is a successor of x in the splice graph, let p' be the longest path in V that is a suffix of py (py is the path generated by appending y to p). We then add an edge from p to p'.

The source and sink of G are the vertices corresponding to splice graph paths $[S]$ and $[T]$, where $[x]$ denotes a single-vertex path. The set $AS(p)$ is the set of vertices p' such that p is a suffix of p'.

Intuitively, the states of the automaton are the vertices of the graph and are labeled with the suffix in consideration at that state. The edges of the graph are the dictionary suffix links of the FSA, now connecting vertices. For $p \in \mathcal{P}$, $AS(p)$ denotes the set of states in FSA that recognizes p. All transcripts start with S, end with T and there is no path in $\mathcal{P}$ containing either of them as they are not real exons, so there exist two vertices labeled $[S]$ and $[T]$. We call them the source and sink of the prefix graph respectively, and we will see they indeed serve a similar purpose.

▶ **Lemma 5.** *There is a one-to-one correspondence between $S-T$ paths in the splice graph and $[S]-[T]$ paths in the prefix graph.*

Proof. Every transcript can be mapped to an $[S]-[T]$ path on the prefix graph by feeding the transcript to the finite state automaton and recording the set of visited states, excluding the initial state where no string is matched. The first state after the initial state is always $[S]$ as the first vertex in an $S-T$ path is S, and the last state is always $[T]$ because there are no other vertexes in the prefix graph that would contain T. Conversely, a $[S]-[T]$ path on the prefix graph can also be mapped back to a transcript, as it has to follow dictionary suffix links (transitions between FSA states), which by our construction can be mapped back to edges in the splice graph. ◀

This implies that the prefix graph is also a DAG: If there is a cycle in the prefix graph, it implies an exon appears twice in a transcript, which violates our assumption that the splice graph is a DAG.

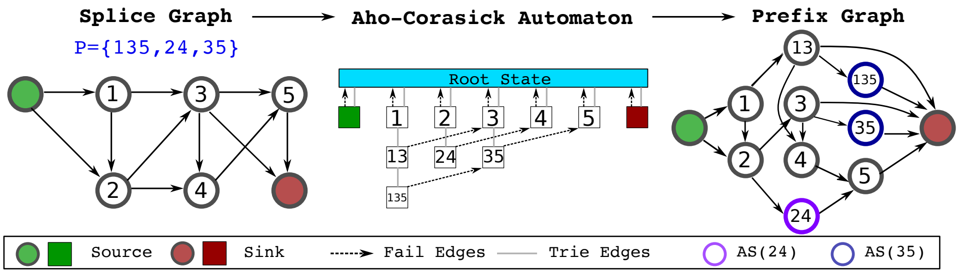


Figure 1 An example construction of the Prefix Graph. The source and sink of the prefix graph are $[S]$ and $[T]$, respectively. The set of phasing paths $\mathcal{P}$ is shown in blue in the left panel, and we does not include the singleton paths for simplicity. We draw the trie and the fail edges for the A-C automaton as it reduces cluttering (dictionary suffix link can be derived from both edge sets). The colored nodes in prefix graph are the vertices (states) in $AS(35)$ and $AS(24)$.

The resulting prefix graph flow serves as a bridge between the path abundance $\{c_p\}$ and the quantified transcript set $\{c_i\}$:

▶ **Theorem 6.** *Every quantified transcript set can be mapped to and from a prefix graph flow. The path abundance is preserved during the mapping and can be calculated exactly from prefix graph flow: $c_p = \sum_{s \in AS(p)} f_s$, where f_s is the flow through vertex s.*

Proof. Using the path mapping between splice graph and prefix graph, we can map a quantified transcript set onto the prefix graph as a prefix graph flow and reconstruct a quantified transcript set by decomposing the flow and map each $[S] - [T]$ path back to the splice graph as a transcript.

To prove the second part, let $\{c_p\}$ be the path abundance calculated from the definition given a quantified transcript set, and $\{c'_p\}$ be the path abundance calculated from the prefix graph flow. We will show $\{c_p\} = \{c'_p\}$ for any finite decomposition of the prefix graph flow.

For any transcript T_i and any path $p \in \mathcal{P}$, since no exon appears twice for a transcript, if T_i contains p, it will be recognized by the FSA exactly once. This means the $[S] - [T]$ path to which T_i maps intersects with $AS(p)$ by exactly one vertex in this scenario, and it contributes the same abundance to c'_p and c_p. If T_i does not contain p, by similar reasoning, it contributes to neither c'_p nor c_p. This holds for any transcript and any path, so the two definitions of path abundance coincide and are preserved in mapping from quantified transcript set to prefix graph flow. Since the prefix graph flow is preserved in flow decomposition, the path abundance is preserved as a function of prefix graph flow. ◀

This connection allows us to directly optimize over $\{c_p\}$ by using the prefix graph flow as variables (the path abundances c_p is now represented as seen in Theorem 6), and use flow balance and non-negativity as constraints, as we describe in the next section. The corresponding quantified transcript set is guaranteed to exist by a flow decomposition followed by the mapping process.

We next describe an improvement to the prefix graph, which we call compact prefix graph. The idea is to recognize phasing paths at the edges of the resulting graph, instead of at the vertices. We will still start with the Aho-Corasick FSA, but we will be building the graph in a way that states of the FSA correspond to edges of the resulting graph, as described below:

▶ **Definition 7** (Compact Prefix Graph)**.** *Given splice graph G_S and set of splice graph paths $\mathcal{P}$, we construct the corresponding compact prefix graph G' as follows. The vertex set of the compact prefix graph is the union of*

- *all single-vertex paths on the splice graph;*
- *any splice graph path p that is the prefix of some path p' in $\mathcal{P}$, while strictly shorter than p'.*

For p in the compact prefix graph, let x be its last exon and y be a successor of x in the splice graph. We create an edge which has label py (again, appending y to p), originates from p, and leads to the node that is the longest suffix of py in the compact prefix graph.

The source and sink of G are the vertices corresponding to splice graph paths $[S]$ and $[T]$. The set $AS(p)$ is the set of **edges** *p' such that the edge label on p' is a suffix of p.*

The set $AS(p)$ bears the same meaning as in the original prefix graph, as the states of the Aho-Corasick FSA are now (roughly) the edges of the compact prefix graph. With this intuition, we can prove the same property as stated in Lemma 5 and Theorem 6 for compact prefix graph. The compact prefix graph by the virtue of its construction is a smaller graph (compared to the original prefix graph) with the same power and is preferred in practice.

2.3 Inference

While the restructuring process described in the previous section reduces the size of the optimization problem, we still need to solve it efficiently. We start with the base case, that is, a single gene and every read pair maps to exactly one path. Recall that $\mathcal{P}$ is the set of phasing paths we consider, $p(f)$ is the path fragment f maps to. For a phasing path p, c_p is the path abundance, $\hat{l}_p$ is the path effective length. For the prefix graph, we let f_e denote the flow through an edge, f_v denote the flow through a vertex, and $AS(p)$ is the set of vertices (as FSA states) that recognize p. We also use $\mathrm{In}(v)$ to denote the incoming edges of vertex v, and $\mathrm{Out}(v)$ similarly for the outgoing edges. The full instance in this case, with the prefix graph proposed the previous section, is:

$$\begin{aligned}
\max \quad & \sum_{f \in F} \log c_{p(f)} & \\
\text{s.t.} \quad & \sum_{p \in \mathcal{P}} c_p \hat{l}_p = 1 & \\
& c_p = \sum_{v \in AS(p)} f_v & \forall p \in \mathcal{P} \\
& f_v = \sum_{e \in \mathrm{In}(v)} f_e = \sum_{e \in \mathrm{Out}(v)} f_e & \forall v \in V - \{[S], [T]\} \\
& f_e \geq 0 & \forall e \in E
\end{aligned}$$

This is slightly different from what we described in Section 2.1. First, we maximize the logarithm of the likelihood objective. Second, we explicitly fix the normalization constant to be 1, instead of placing it on the divisor of the fragment likelihood. This does not change the objective, and the only difference is that in the original form $\{c_p\}$ can be arbitrarily scaled, while here the scaling is fixed. The variables and the constraints come from the prefix graph flow, and c_p is represented as in Theorem 6. For a compact prefix graph as described in Definition 7, we simply replace the equation of c_p to sum over f_e with $e \in AS(p)$. This is a convex problem, as the target function is convex with respect to $\{c_p\}$, and the constraints are all linear. We can solve the problem with general purpose convex solvers.

With the presence of multimapped reads (to multiple genes and/or multiple paths within one gene), we can employ a standard EM approach. Recall $M(f)$ is the set of phasing paths onto which f can map, and for $p \in M(f)$ let $A(f \mid p)$ denote the affinity of f mapping to p, as described in Section 2.1. We also let z denote the hidden allocation vector, where $z_{f,p}$ denotes the probability that fragment f is mapped onto splice graph path p. We can alternatively optimize for $\{z_{f,p}\}$ and $\{c_p\}$ until convergence as follows:

$$\begin{aligned}
z_{f,p}^{(t)} &= c_p^{(t)} A(f \mid p) / (\sum_{p' \in M(f)} c_{p'}^{(t)} A(f \mid p')) \\
c^{(t+1)} &= \arg\max_c \sum_{p \in \mathcal{P}} (\sum_{f \in F} z_{f,p}^{(t)}) \log c_p, \text{s.t.} \sum_{p \in \mathcal{P}} c_p \hat{l}_p = 1
\end{aligned}$$

$z^{(t)}$ and $c^{(t)}$ denote the variables at iteration t. We hide the constraint from prefix graphs for clarity. The optimization for $z_{f,p}^{(t)}$ can be run in parallel, so we focus on the M-step that optimizes $c^{(t+1)} = \{c_p^{(t+1)}\}$, hiding the superscript whenever it is clear from context. When we optimize over the whole genome, the instance becomes impractically huge. This is because we need to infer the flow for every prefix graph (one for each gene) across the whole genome, and we need to satisfy flow balance for each graph and normalization for all graphs together.

We let $\mathcal{G}$ denote the set of genes. Denote the gene abundance $c_g = \sum_{p\in\mathcal{P}_g} c_p\hat{l}_p$, where $\mathcal{P}_g$ is the set of phasing paths in gene g. We then define relative abundance $c^*_p = c_p/c_g$ for every phasing path p. Plugging $c_p = c_g c^*_p$ into the expression for M-step, we have the following transformed objective:

$$\begin{aligned}
\max \quad & \sum_{p\in\mathcal{P}}(\sum_{f\in F} z^{(t)}_{f,p})(\log c^*_p + \log c_g)\\
&= \sum_{g\in\mathcal{G}}\sum_{p\in\mathcal{P}_g}(\sum_{f\in F} z^{(t)}_{f,p})\log c^*_p + \sum_{g\in\mathcal{G}}(\sum_{f\in F}\sum_{p\in\mathcal{P}_g} z^{(t)}_{f,p})\log c_g\\
&= \sum_{g\in\mathcal{G}}\sum_{p\in\mathcal{P}_g}(\sum_{f\in F} z^{(t)}_{f,p})\log c^*_p + \sum_{g\in\mathcal{G}} s_g\log c_g\\
\text{s.t.} \quad & \sum_{p\in\mathcal{P}} c_g c^*_p\hat{l}_p = \sum_{g\in\mathcal{G}} c_g = 1\\
& \sum_{p\in\mathcal{P}_g} c^*_p\hat{l}_p = 1, \forall g\in\mathcal{G}
\end{aligned}$$

Here, $s_g = \sum_{f\in F}\sum_{p\in\mathcal{P}_g} z^{(t)}_{f,p}$ can be interpreted as the estimated read count of gene g. Again for clarity we hide the prefix graph constraints for c^*_p, which retain their original form because all prefix graph constraints are affine. Now, we can decouple optimization of c^*_p and c_g, as the objective function is split into two parts, and each constraint only involves one of them. The optimization for c^*_p can be done for each gene independently, and it is exactly the single-gene optimization as we described above except we weight $\log c_{p(f)}$ with $z^{(t)}_{f,p}$ in the objective. The optimization for c_g has the form $\max\sum_{g\in\mathcal{G}} s_g\log c_g$ constrained by $\sum_{g\in\mathcal{G}} c_g = 1$, from which we derive that $c_g \propto s_g$. Since $\sum_{g\in\mathcal{G}} s_g = \sum_{p\in\mathcal{P}}\sum_{f\in F} z^{(t)}_{f,p} = \sum_{f\in F} 1 = |F|$, we have the following localized EM algorithm:

$$\begin{aligned}
\text{Global E-step:} \quad & z^{(t)}_{f,p} = c^{(t)}_p A(f\mid p)/(\sum_{p'\in M(f)} c^{(t)}_{p'}A(f\mid p'))\\
\text{Gene-Level M-step:} \quad & c^{(t+1)} = \arg\max_c \sum_{p\in\mathcal{P}_g}(\sum_{f\in F} z^{(t)}_{f,p})\log c_p\\
& \text{s.t.} \sum_{p\in\mathcal{P}_g} c_p\hat{l}_p = \sum_{p\in\mathcal{P}_g}\sum_{f\in F} z^{(t)}_{f,p}/|F|, \forall g\in\mathcal{G}
\end{aligned}$$

The M-step is run independently for each gene and can be parallelized. Again we omit listing the prefix graph constraint over c_p for clarity, and c_g is implicitly derived as the right-hand side of the normalization constraint.

3 Experiments

Based on the expression quantification method Salmon [12] and its effective lengths, we implement our method and call it Graph Salmon. We apply Graph Salmon on three bipolar disease (BD) RNA-seq samples and three control samples to estimate the expression network flow on neurogenesis-related genes (GO:0022008), which are known to have complex alternative splicing patterns and novel isoforms. We use this as a case study to show that Graph Salmon is applicable with variable fragment lengths and that the relative usage of splice junctions under the incomplete reference assumption are different from those under complete reference assumption.

3.1 Implementation

The splice graphs are constructed using the reference exons and splice junctions of Gencode [5] version 26. Since Salmon's effective lengths are needed for path effective lengths, we first run Salmon on the samples. We also use Salmon read mappings (obtained with the `-writeMappings` argument) and convert their coordinates onto splice graph nodes and edges. Prefix graphs are constructed with the converted read mappings. Each edge in the prefix graph corresponds to a path in the original splice graph, and we compute the path effective length by taking the average of the effective lengths of the corresponding region in reference transcripts that include the corresponding path (for details see Supplementary Material Section A.1). With the converted read mappings and path effective lengths, the probabilistic model of graph quantification can be specified.

Since only neurogenesis-related genes are of interest and the rest of the genes are assumed to have complete reference transcripts, we assume that Salmon correctly estimates the probability of each paired-end read generated from each gene when the read is mapped to multiple genes. We use Salmon's gene-level weight assignment as read count and only solve the flow optimization problem within each gene, which corresponds to one round of the gene-level M step for each gene.

3.2 Graph Salmon reveals unique between-sample differences of PSI for neurogenesis genes

The RNA-seq data can be accessed from Gene Expression Omnibus (GEO) database with accession numbers GSM1288369, GSM1288370, GSM1288371 for bipolar disease samples, and GSM1288374, GSM1288375, GSM1288376 for control samples [2].

The mean fragment lengths of the six sample range from 349.17 bp to 375.28 bp. The standard deviations of fragment lengths are between 53.00 bp and 82.15 bp. Meanwhile, 30% of the exons (or subexons) across the splice graphs are less than 56 bp long, and the 40% quantile of subexon lengths is 79 bp. Graph Salmon is needed in this dataset because of the large standard deviation of fragment lengths compared to subexon lengths.

We computed Percentage Spliced In (PSI) of 2441 skipped exon events using the Graph Salmon network flow and compare them with PSIs calculated using Salmon's expression quantification based on the reference transcripts. Given three exons, the PSI is defined as the total abundance of transcripts that include all three of them, divided by total abundance of transcripts that include the first and the last (but not necessarily the middle one). The correlations of Graph Salmon PSI and Salmon PSI of the same sample are around 0.51 to 0.57 (for both Spearman and Pearson), while the correlations of PSI between different samples computed by the same quantification method are over 0.75 (for both Spearman and Pearson and both methods). The large correlation between different samples can be explained by the fact that they are from the same tissue and should follow the tissue-specific expression and alternative splicing patterns. The smaller correlation between different quantification methods indicates the incomplete reference and complete reference assumptions lead to very different splice junction abundance estimates.

An example of different PSI computed by Graph Salmon and Salmon is shown in Figure 2 and Supplementary Figure A2 on *LPAR1* gene. *LPAR1* gene encodes a lysophosphatidic acid (LPA) receptor that functions in the LPA signaling pathway, which is related to cognitive behavioral deficits such as schizophrenia and depression when dysregulated [20]. We focus on the event that describes the percentage of expression of the inclusions of exon 6 (position 110973480-110973558 in GRCh38) between exon 3 (position 111037840-111038043 in GRCh38)

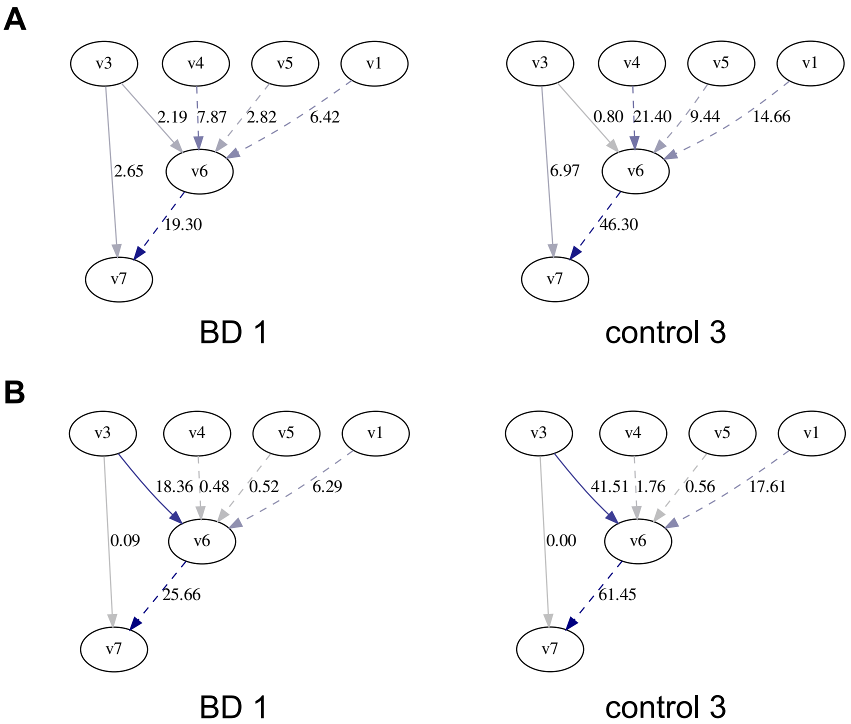


Figure 2 (A) Network flow of BD 1 and control 3 samples estimated by Graph Salmon. The subgraph includes exons 1, 3 to 7, and exons are represented by nodes and node label indicates the index of exon. PSI of inclusion of exon 6 between exon 3 and 7 is computed. Edges of which the flows are involved in PSI calculation are solid; the rest edges are dashed. (B) Network flow of the same samples computed by Salmon with reference transcripts.

and exon 7 (position 110972072-110972220 in GRCh38). Graph Salmon computes the PSIs to be 0.45 to 0.64 for BD samples and 0.07 to 0.33 for control samples, whereas PSIs computed by Salmon are larger than 0.95 for all six samples.

Even though this difference is not evaluated by rigorous statistical testing, it indicates that when reference is incomplete, previous reference-based alternative splicing analysis may lead to different results. Considering the incomplete reference assumption in alternative splicing analysis enlarges the pool of candidate alternative splicing events.

4 Discussion

We improve the graph quantification model of FlipFlop to incorporate phasing information from variable length reads or fragments. The key algorithmic contributions are a provably correct reparameterization process and the introduction of the prefix graph inspired by Aho-Corasick automata for inference.

To demonstrate the feasibility of our method to handle variable length fragments, we apply our method to neurogenesis-related genes of bipolar disease RNA-seq samples and control RNA-seq samples. The RNA-seq samples contain paired-end reads with mean fragment lengths around 350 bp and standard deviation around 53 – 82 bp. We show that our method successfully estimates network flows on prefix graphs and the estimated flow (under the incomplete reference assumption) only has around 0.5 correlation (both Pearson and Spearman) with the flow estimated by Salmon under the complete reference assumption.

The size of the prefix graph depends on the length of the phasing paths exponentially. Unfortunately, for long read sequencing, especially with transcript-long reads, the prefix graph may be as large as the set of all $S - T$ paths (equivalently the set of all possible transcripts) and its efficiency compared to the naïve implementation of graph quantification (where we enumerate every compatible transcript) may diminish. It is still open what algorithmic tools are required to avoid this ineffiency.

An intrinsic issue with graph quantification is non-identifiability: Many configurations of transcript abundances lead to the same read generation model, and thus it is impossible to distinguish which configuration is closer to the ground truth if our goal is to recover an underlying transcriptome. While our prefix graph representation is compact, for many downstream analyses, we are invariably forced to perform a flow decomposition to transform prefix graph flow into quantified transcript sets. The non-identifiability problem manifests in this step, as different decompositions can lead to the same prefix graph flow, which as we proved implies the same model of read generation. Therefore, it is possible to assess the severity of non-identifiability problem by inspecting different ways of decomposing a fixed prefix graph flow.

This work focuses on theoretical improvements of the graph quantification model, while its practical utility is still largely unexplored. For example, our proposed approach may be a promising method for transcript assembly similar to FlipFlop, where we use quantification for assembly. The method also has potential use cases in alternative splicing analyses and other related tasks in RNA-seq. However, careful benchmarking is needed to determine the cases when graph quantification is superior to standard quantification with a given set of transcripts.

References

1 Alfred V Aho and Margaret J Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

2 N Akula, J Barb, X Jiang, JR Wendland, KH Choi, SK Sen, L Hou, DTW Chen, G Laje, K Johnson, et al. RNA-sequencing of the brain transcriptome implicates dysregulation of neuroplasticity, circadian rhythms and GTPase binding in bipolar disorder. *Molecular Psychiatry*, 19(11):1179–1185, 2014.

3 Elsa Bernard, Laurent Jacob, Julien Mairal, and Jean-Philippe Vert. Efficient RNA isoform identification and quantification from RNA-Seq data with network flows. *Bioinformatics*, 30(17):2447–2455, 2014.

4 Nicolas L Bray, Harold Pimentel, Páll Melsted, and Lior Pachter. Near-optimal probabilistic RNA-seq quantification. *Nature Biotechnology*, 34(5):525–527, 2016.

5 Adam Frankish, Mark Diekhans, Anne-Maud Ferreira, Rory Johnson, Irwin Jungreis, Jane Loveland, Jonathan M Mudge, Cristina Sisu, James Wright, Joel Armstrong, et al. GENCODE reference annotation for the human and mouse genomes. *Nucleic Acids Research*, 47(D1):D766–D773, 2018.

6 James Hensman, Panagiotis Papastamoulis, Peter Glaus, Antti Honkela, and Magnus Rattray. Fast and accurate approximate inference of transcript expression from RNA-seq data. *Bioinformatics*, 31(24):3881–3889, 2015.

7 Yarden Katz, Eric T Wang, Edoardo M Airoldi, and Christopher B Burge. Analysis and design of RNA sequencing experiments for identifying isoform regulation. *Nature Methods*, 7(12):1009, 2010.

8 Laura H LeGault and Colin N Dewey. Inference of alternative splicing from RNA-Seq data with probabilistic splice graphs. *Bioinformatics*, 29(18):2300–2310, 2013.

9 Bo Li and Colin N Dewey. RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome. *BMC Bioinformatics*, 12(1):323, 2011.

10 Juntao Liu, Ting Yu, Tao Jiang, and Guojun Li. TransComb: genome-guided transcriptome assembly via combing junctions in splicing graphs. *Genome Biology*, 17(1):213, 2016.

11 Lior Pachter. Models for transcript quantification from RNA-Seq. *arXiv preprint arXiv:1104.3889*, 2011.

12 Rob Patro, Geet Duggal, Michael I Love, Rafael A Irizarry, and Carl Kingsford. Salmon provides fast and bias-aware quantification of transcript expression. *Nature Methods*, 14(4):417–419, 2017.

13 Mihaela Pertea, Geo M Pertea, Corina M Antonescu, Tsung-Cheng Chang, Joshua T Mendell, and Steven L Salzberg. StringTie enables improved reconstruction of a transcriptome from RNA-seq reads. *Nature Biotechnology*, 33(3):290–295, 2015.

14 Mingfu Shao and Carl Kingsford. Accurate assembly of transcripts through phase-preserving graph decomposition. *Nature Biotechnology*, 35(12):1167–1169, 2017.

15 Mingfu Shao and Carl Kingsford. Theory and a heuristic for the minimum path flow decomposition problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16(2):658–670, 2019.

16 Shihao Shen, Juw Won Park, Zhi-xiang Lu, Lan Lin, Michael D Henry, Ying Nian Wu, Qing Zhou, and Yi Xing. rMATS: robust and flexible detection of differential alternative splicing from replicate RNA-Seq data. *Proceedings of the National Academy of Sciences*, 111(51):E5593–E5601, 2014.

17 Manuel Tardaguila, Lorena De La Fuente, Cristina Marti, Cécile Pereira, Francisco Jose Pardo-Palacios, Hector Del Risco, Marc Ferrell, Maravillas Mellado, Marissa Macchietto, Kenneth Verheggen, et al. SQANTI: extensive characterization of long-read transcript sequences for quality control in full-length transcriptome identification and quantification. *Genome Research*, 28(3):396–411, 2018.

18 Cole Trapnell, Brian A Williams, Geo Pertea, Ali Mortazavi, Gordon Kwan, Marijke J Van Baren, Steven L Salzberg, Barbara J Wold, and Lior Pachter. Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature Biotechnology*, 28(5):511–515, 2010.

19 Juan L Trincado, Juan C Entizne, Gerald Hysenaj, Babita Singh, Miha Skalic, David J Elliott, and Eduardo Eyras. SUPPA2: fast, accurate, and uncertainty-aware differential splicing analysis across multiple conditions. *Genome Biology*, 19(1):40, 2018.

20 Yun C Yung, Nicole C Stoddard, Hope Mirendil, and Jerold Chun. Lysophosphatidic acid signaling in the nervous system. *Neuron*, 85(4):669–682, 2015.

A Additional Methods

In Section A.1 and Section A.2, we describe two modifications to our proposed model in practice. These modifications greatly improve practicality of our proposed model. On the other hand, inclusion of these modifications means the conditions for Theorem 3 no longer hold and the inference is no longer over the exact set of all possible transcripts with exact bias correction. In Section A.3, we describe the fragment graph constructed in FlipFlop in more detail, and describe our reasoning why it loses the theoretical guarantee in presence of variable-length reads or fragments.

A.1 Bias Correction, Continued

Admittedly, there is no way to know the exact location of the read pair within the transcript after reparameterization with path abundance, so these specific biases cannot be integrated into our proposed models directly. Nonetheless, since the splice graph is known in full, approximate bias correction is possible. Let $B_i(j,k)$ denote the affinity value calculated from a full bias correction model for the fragment generated from base j to k on transcript T_i, and $\hat{r}_i$ be the reference abundance of transcript T_i. We let $A_p(j,k) = (\sum_{i:p\subset p(T_i)} \hat{r}_i B_i(j',k'))/(\sum_{i:p\subset p(T_i)} \hat{r}_i)$, where j' and k' are the coordinates of the path sequence in the reference coordinates of T_i. This means we average the affinity value calculated from known transcripts on this locus, weighted by their reference abundance. For simplicity, we use Salmon outputs as reference abundance, but other approaches, including an iterative process of estimating $A_p(j,k)$ and c_p alternatively, are possible.

One limiting factor for bias correction in our proposed framework is that calculating $\hat{l}_p$ can be expensive as we need to calculate the affinity value for every possible fragment. While we did this for our experiments, there are also alternative approaches that speeds up the process by approximating $\hat{l}_p$. We can use a simplified form for $A_p(j,k)$ so $\hat{l}_p$ has a closed-form solution (for example, do not allow bias correction when calculating effective length, similar to existing approaches to calculate effective length of transcripts), sample from possible $A_p(j,k)$ when the number of fragments from a particular path is large, or precompute the values for fixed genome and bias correction model. These approaches may result in slightly inaccurate path effective length, and it is still an open question how it affects downstream procedures.

A.2 Trimming Set of Phasing Paths

Recall the likelihood function under our reparameterized model:
$P(F \mid \mathcal{T}, c) \propto \prod_{f\in F} c_{p(f)}/(\sum_{p\in\mathcal{P}} c_p\hat{l}_p)$. Paths $p \in \mathcal{P}$ with no mapped fragments do not contribute to $\prod_{f\in F} c_{p(f)}$, as there are no $f \in F$ such that $p(f) = p$. These paths do play a role in calculating the normalization constant $\sum_{p\in\mathcal{P}} c_p\hat{l}_p$. However, since we only remove paths with very low $\hat{l}_p$, the contribution of $c_p\hat{l}_p$ from this set is small. This removal thus causes small underestimation of the normalization constant, and in turn small overestimation of transcript abundances (and path abundances).

If there is a removed path with large c_p when optimized under this model, it means in the inferred quantified transcript set there is are many fragments mapped to path p, even though exactly zero fragments are mapped to p in the sequencing library. This mostly happens if there is a dominant transcript with high abundance, and p is part of the transcript. For such things to happen, the transcript must have many fragments mappable, and the fact that no single fragment mapped to path p indicates $\hat{l}_p$ is small, or the modeling might be faulty. This trimming is necessary in practice, as the fragment length distribution $D(l)$ usually has a long tail when inferred from experiments, due to smoothing and potential mapping errors, leading to many extremely long paths that are near impossible to sample a read from.

A.3 FlipFlop and the Fragment Graph

The fragment graph constructed by FlipFlop is defined as follows. Given splice graph G and a set of phasing paths $\mathcal{P}$ (again we consider a general notion of phasing paths, meaning single exon paths also count as phasing), the fragment graph G_F is constructed such that

- Each vertex in G_F is either S, T, or a phasing path.
- There is an edge connecting X to Y only if Y is a single exon extension or shrinking of X (unless one of them is either S or T).
- Every $S - T$ path in the splice graph can be mapped uniquely to a $S - T$ path in the fragment graph, and the set of vertices included in the $S - T$ path is exactly the set of phasing paths that are a subpath of the transcript.

We only list a necessary condition in the second item, and we will not discuss how the graph is constructed in practice here. The third item is essential for the FlipFlop algorithm, as it solves the inference problem with convex cost flows which requires that every phasing path is represented by a single vertex in the graph.

As discussed in the introduction, the FlipFlop algorithm is correct when input library is single-end reads with fixed read length. This implies that if X is a phasing path, there are no phasing paths that are extension of X on both ends (for example, if $X = [3, 4]$, then $[2, 3, 4, 5]$ cannot be a phasing path), otherwise it would violate the condition that all reads have equal length. We now show that there exists no correct fragment graph when the condition is violated.

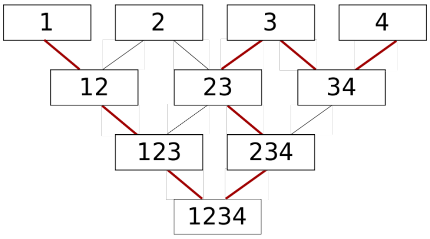


Figure A1 The fragment graph with 4 exons and 10 phasing paths, not including S and T. Blocks denote vertices of the fragment graph, and lines denote possible edges between vertices (phasing paths). A path visiting 9 vertices (excluding the singleton phasing path [2]) is marked in dark red, and there is no Hamiltonian path in the graph.

Consider a splice graph with a chain of four exons denoted $1, 2, 3$ and 4, and where every subpath of $[1, 2, 3, 4]$ is a phasing path. The fragment graph, if exists, will contain 10 vertices (excluding S and T) and a Hamiltonian path corresponding to the transcript $[1, 2, 3, 4]$. However, as seen in the above figure, the graph will not contain a Hamiltonian path no matter how the graph is constructed.

B Additional Figure

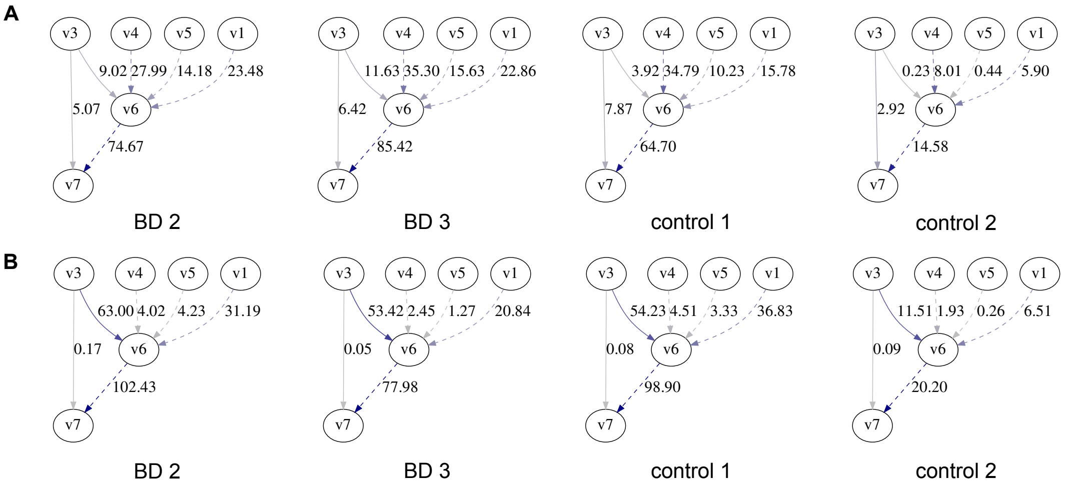


Figure A2 (A) Network flow of BD 2, BD 3, control 1, and control 2 samples estimated by Graph Salmon. The subgraph includes exons 1, 3 to 7, and exons are represented by nodes and node label indicates the index of exon. PSI of inclusion of exon 6 between exon 3 and 7 is computed. Edges of which the flows are involved in PSI calculation are solid; the rest edges are dashed. (B) Network flow of the same samples computed by Salmon with reference transcripts.

Shape Decomposition Algorithms for Laser Capture Microdissection

Leonie Selbach
Department of Computer Science and Center for Protein Diagnostics, Ruhr University Bochum, Germany
leonie.selbach@rub.de

Tobias Kowalski
Department of Biophysics and Center for Protein Diagnostics, Ruhr University Bochum, Germany
tobias.kowalski@rub.de

Klaus Gerwert
Department of Biophysics and Center for Protein Diagnostics, Ruhr University Bochum, Germany
klaus.gerwert@rub.de

Maike Buchin
Department of Computer Science, Ruhr University Bochum, Germany
maike.buchin@rub.de

Axel Mosig
Department of Biophysics and Center for Protein Diagnostics, Ruhr University Bochum, Germany
axel.mosig@rub.de

Abstract

In the context of biomarker discovery and molecular characterization of diseases, laser capture microdissection is a highly effective approach to extract disease-specific regions from complex, heterogeneous tissue samples. These regions have to be decomposed into feasible fragments as they have to satisfy certain constraints in size and morphology for the extraction to be successful. We model this problem of constrained shape decomposition as the computation of optimal feasible decompositions of simple polygons. We use a skeleton-based approach and present an algorithmic framework that allows the implementation of various feasibility criteria as well as optimization goals. Motivated by our application, we consider different constraints and examine the resulting fragmentations. Furthermore, we apply our method to lung tissue samples and show its advantages in comparison to a heuristic decomposition approach.

2012 ACM Subject Classification Applied computing → Bioinformatics; Theory of computation → Computational geometry; Theory of computation → Dynamic programming

Keywords and phrases Laser capture microdissection, shape decomposition, skeletonization

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.13

1 Introduction

Laser capture microdissection (LCM) [14] is a highly effective approach to extract specific cell populations from complex, heterogeneous tissue samples and has been used extensively in the context of biomarker discovery [15] as well as the molecular characterization of diseases [17]. Since LCM separates homogeneous and disease-specific regions from their heterogeneous and unspecific surrounding tissue regions, the characterizations obtained from genomic, transcriptomic or proteomic characterizations of samples processed with LCM provide more accurate molecular markers of diseases [9, 20]. With LCM being used more and more commonly in clinical studies, there is a need to automate all procedures involved in sample processing.

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti; Article No. 13; pp. 13:1–13:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Problem Statement

In our present contribution, we address one central problem of processing samples with LCM. Namely that the regions of interest (ROIs) consist of complex shapes of varying size which in general cannot be extracted in one piece from the tissue sample. Rather, the ROI needs to be fragmented into small subregions that satisfy certain constraints in size and morphology: Fragments must not exceed certain limits of minimal or maximal size and should be of approximately round shape. If a fragment does not meet these constraints, it cannot be properly extracted from the surrounding tissue. This negatively affects the sample quality and thus compromises the advantages of LCM-based sample preparation.

By interpreting each connected component of the ROI as a simple polygon, we can model this problem of constrained shape decomposition as the computation of optimal feasible decompositions of polygons. The constraints can be modeled as certain feasibility criteria and optimization goals. Our decomposition method utilizes a skeleton of the shape and follows a dynamic approach. Specifically, we restrict our cuts to certain line segments based on the skeleton. This not only results in simple cuts but also in a flexible framework that allows to integrate various criteria. With respect to our application, we consider different criteria regarding for example the area, convexity or fatness of a polygon or the length of inserted cuts. However, other constraints as well as combinations of multiple criteria are possible.

Application

Our contribution is motivated by an application introduced in [15] in which the ROI to be dissected from the tissue sample is identified using label-free hyperspectral infrared microscopy. In this approach, an infrared microscopic image of the sample yields infrared pixel spectra at a spatial resolution of about 5 μm. A previously trained random forest classifier assigns each pixel spectrum to one tissue component such as *healthy* or *diseased*, with the *diseased* class being further subdivided into *inflamed tissue* as well as several subtypes of thoracal tumors. The general sample preparation task in the context of LCM is to dissect all tumor regions (or all regions identified as one specific tumor subtype) from a sample. While our current contribution deals with the specific context of label-free infrared microscopy, our shape decomposition approach equally applies more broadly to LCM in the context of other microscopic modalities, most notably H&E stained images [9] for which recent digital pathology approaches facilitate reliable computational identification of disease specific regions [26, 28].

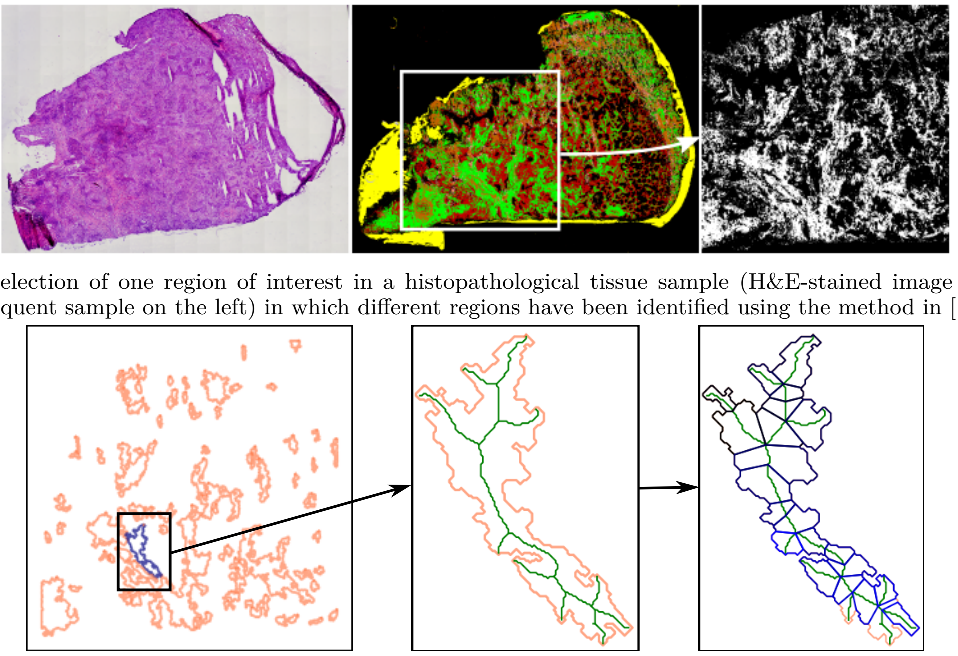
2 Related Work

Polygon decomposition is an important tool in computational geometry, as many algorithms work more efficiently on certain polygon classes, for example convex polygons [16]. Moreover, polygon decomposition is frequently used in applications such as pattern recognition or image processing [16]. Object recognition, biomedical image analysis and shape decomposition are typical areas of application that utilize skeletons [23]. Skeletons are oftentimes used to analyze the morphology of a given shape and work especially well on elongated structures, such as vessels [11], pollen tubes [27] or neuron images [19, 24]. There are several shape decomposition methods based on the skeleton or some other medial representation of a shape. However, most of these methods are designed for object recognition and thus focus on decomposing a shape into “natural” or “meaningful” parts [22, 18, 21]. In some approaches even decompositions with overlapping parts are allowed [10, 25]. None of the established decomposition methods facilitate a straightforward introduction of adjustable size and shape constraints as needed for our application.

We utilize the skeleton for two main reasons: it is well-established to represent shape morphology and has proved useful for shape decomposition previously. As cancerous tissue regions often present themselves as highly complex and ramified shapes, we apply the skeleton to obtain a morphological representation, based on which we compute a decomposition that includes the morphological features.

3 Methods

As an input, we receive a binary mask of a microscopic slide with the region of interest (ROI) as the foreground (Fig. 1a). After a preprocessing step, the foreground is reduced to connected components without holes. We can interpret each of these components as a polygon by taking each boundary pixel as a polygon vertex. The goal is to produce a decomposition of each component in such a way that the fragments fulfill certain constraints in size and morphology. We compute this fragmentation using a skeleton-based approach for polygon decomposition (Fig. 1b).



(a) Selection of one region of interest in a histopathological tissue sample (H&E-stained image of a subsequent sample on the left) in which different regions have been identified using the method in [15].

(b) After a preprocessing step, each connected component is a simple polygon without holes and is decomposed using the method presented in this paper.

Figure 1 Decomposition of ROI polygons in a histopathological tissue sample.

3.1 Skeleton of a shape

Our approach is based on the medial axis or *skeleton* of the polygon. The medial axis of a shape is the set of points that have more than one closest point on the shape's boundary. The medial axis was introduced for the description of biological shapes [3, 4] but is now widely used in other applications such as object recognition, medical image analysis and shape decomposition [23]. An important property is that the medial axis represents the object and its geometrical and topological characteristics while having a lower dimension [8, 29].

Formally, the medial axis of a shape D is defined as the set of centers of maximal disks in D. A closed disk $B \subset D$ is maximal in D if every other disk that contains B is not contained in D. A point s is called *skeleton point* if it is the center of a maximal disk $B(s)$ (see Fig. 2). For a skeleton point s, we call the points where $B(s)$ touches the boundary the *contact points* – every skeleton point has at least two contact points. A skeleton S is given as a graph consisting of connected arcs S_k, which are called *skeleton branches* and meet at *branching points*. Given a simple polygon without holes the skeleton is an acyclic graph.

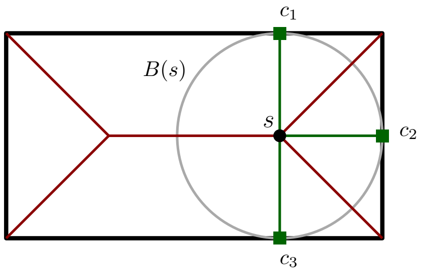


Figure 2 Medial axis of a simple shape. The skeleton point s is a branching point with three contact points c_1, c_2, c_3.

There are various methods for the computation of the medial axis in practice [23]. In general, the medial axis is very sensitive to noise in an object's boundary. This is a problem that often occurs in digital images and leads to spurious skeleton branches. Therefore, many approaches apply some kind of pruning method to remove those branches. In our application, we have a discrete input and a discrete output is expected. Because of that, we use a skeletonization algorithm that computes a discrete and pruned skeleton, which consists of a finite number of skeleton pixels as our skeleton points [2]. Furthermore, the computed skeleton has the property that every branching point has a degree of exactly three.

3.2 Skeleton-based polygon decomposition

We consider the following problem: Given a simple polygon P, compute an optimal feasible decomposition of P. A decomposition is *feasible* if every subpolygon is feasible, in the sense that it fulfills certain conditions on for instance its size and shape. We present an algorithmic framework that allows the integration of various criteria for both feasibility and optimization, which are discussed later. As for now, we only consider criteria that are locally evaluable.

In our skeleton-based approach, we allow only cuts that are line segments between a skeleton point and its corresponding contact points. Thus, the complexity of our algorithm mainly depends on the number of skeleton points rather than the number of boundary points of the polygon. Every subpolygon in our decomposition is generated by two or more skeleton points. We present two decomposition algorithms: One in which we restrict the subpolygons to be generated by exactly two skeleton points and a general method. In the first case, each subpolygon belonging to a skeleton branch can be decomposed on its own and in the second case the whole polygon is decomposed at once.

Decomposition based on linear skeletons

First, let us consider the restriction that the subpolygons are generated by exactly two skeleton points. In this case, the corresponding skeleton points have to be on one skeleton branch S_k. In our work, a branching point belongs to exactly three branches and has exactly three contact points. When a branching point generates a subpolygon with another skeleton point on an adjacent branch, we choose those two line segments that correspond to this branch. Due to the Domain Decomposition Lemma (see Fig. 3, proof in [8]) and the following corollary, we can decompose each skeleton branch on its own.

▶ **Theorem 1** (Domain Decomposition Lemma). *Given a domain D with skeleton $S(D)$, let $p \in S(D)$ be some skeleton point and let $B(p)$ be the corresponding maximal disk. Suppose $A_1, A_2, \ldots, A_k$ are the connected components of $D \setminus B(p)$. Define $D_i = A_i \cup B(p)$ for all i. Then:*

$$S(D) = \bigcup_{i=1}^{k} S(D_i).$$

Moreover, we have

$$S(D_i) \cap S(D_j) = p \; \forall \; i \neq j.$$

▶ **Corollary 2.** *Let $p \in S(D)$ and $A_1, A_2, \ldots, A_k$ be as above. For each skeleton point $q \neq p$ exists an i such that all contact points of q are contained in A_i.*

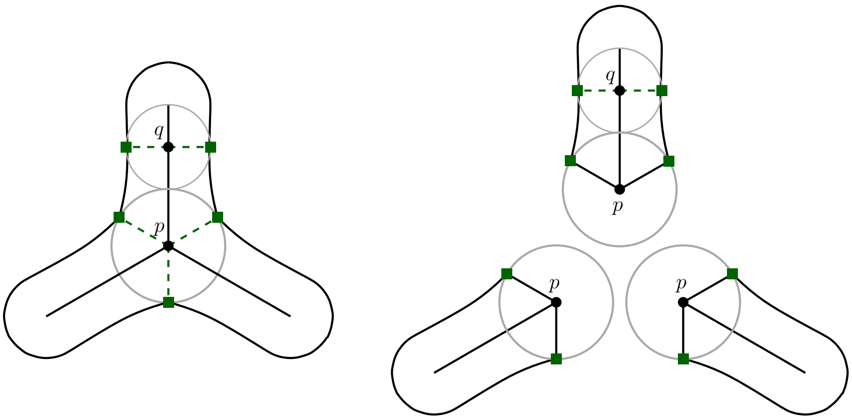


Figure 3 Domain Decomposition Lemma.

Let S_k be a skeleton branch with a linear skeleton of size n_k and let P_k be the polygon belonging to this branch. By $P_k(i, j)$, we denote a subpolygon that is generated by two skeleton points i and j on S_k (see Fig. 4). Thus, we have $P_k(1, n_k) = P_k$. First, we consider the decision problem. Note that there exists a feasible decomposition of a polygon $P_k(i, n_k)$ if either

- $P_k(i, n_k)$ is feasible or
- there exists $j > i$ such that $P_k(i, j)$ is feasible and $P_k(j, n_k)$ has a feasible decomposition.

Thus, we can solve the problem by using dynamic programming and use backtracking to compute the corresponding decomposition. We can include different optimization criteria by choosing an optimal point j.

▶ **Lemma 3.** *Given a subpolygon P_k with a linear skeleton S_k consisting of n_k points, one can compute a feasible decomposition of P_k based on S_k in time $\mathcal{O}({n_k}^2 F)$, with F being a factor depending on the feasibility criteria.*

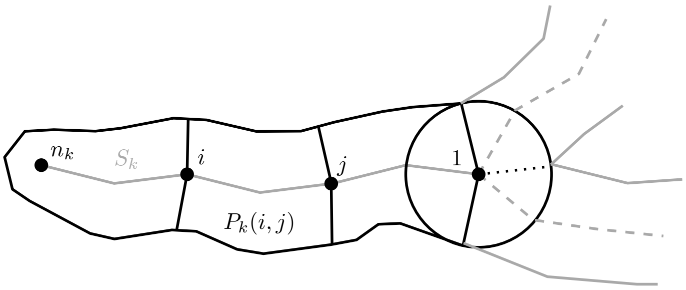


Figure 4 Polygon P_k belonging to a skeleton branch S_k.

Proof. For every skeleton point i, for $i = n_k - 1$ down to 1, we compute $X(i)$ such that $X(i)$ equals `True` if there exists a feasible decomposition of $P_k(i, n_k)$. To compute $X(i)$, we consider $\mathcal{O}(n_k)$ other values $X(i')$ for $i < j \leq n_k$ and check in time $\mathcal{O}(F)$ if the polygon $P_k(i,j)$ is feasible. The correctness follows inductively. ◀

The factor F is determined by the runtime it takes to decide whether a subpolygon is feasible. This factor might depend on for instance the number of points in the skeleton or in the boundary of the polygon. We discuss examples in the following section. After computing decompositions for each subpolygon corresponding to a skeleton branch, we can combine those to obtain a decomposition of the entire polygon. This leads to the following result.

▶ **Theorem 4.** *Given a simple polygon P with skeleton S consisting of n points, one can compute a feasible decomposition of P based on the skeleton branches of S in time $\mathcal{O}(n^2F)$, with F being a factor depending on the feasibility criteria.*

Note that there might not exist a feasible decomposition of the entire polygon or for certain subpolygons. By using this method, we are able to obtain partial decompositions. Thus, this approach can be favorable in practice.

General decomposition

In the general setting, subpolygons are allowed to be generated by more than two skeleton points. In this paper, we will briefly explain the idea of our method (see [5] for a more detailed description and the corresponding formulas). Recall that our skeleton is an acyclic graph consisting of a finite number of vertices, i.e. skeleton points. The skeleton computed for our application (method of Bai et al. [2]) has the property that the maximal degree of a skeleton point is three. We select one branching point as a root and consider a rooted skeleton tree. Since branching points belong to three different branches, these nodes are duplicated in the skeleton tree such that each node corresponds to the cut edges on the respective branch (see Fig. 5). Our method and its runtime are based on two main observations.

▶ **Observation 5.** The maximal number of skeleton points that can generate a subpolygon is equal to the number of endpoints in the skeleton, i.e. the number of leaves in the skeleton tree.

▶ **Observation 6.** Every subpolygon can be represented as the union of subpolygons generated by just two skeleton points.

Let i be a node in the skeleton tree and T_i the subtree rooted in i. By $P(i)$, we denote the subpolygon ending in the skeleton point i. This polygon corresponds to the subtree T_i in the given tree representation (see Fig. 6). For each node i (bottom-up), we compute if there exists a feasible decomposition of the polygon $P(i)$. Such a decomposition exists if either

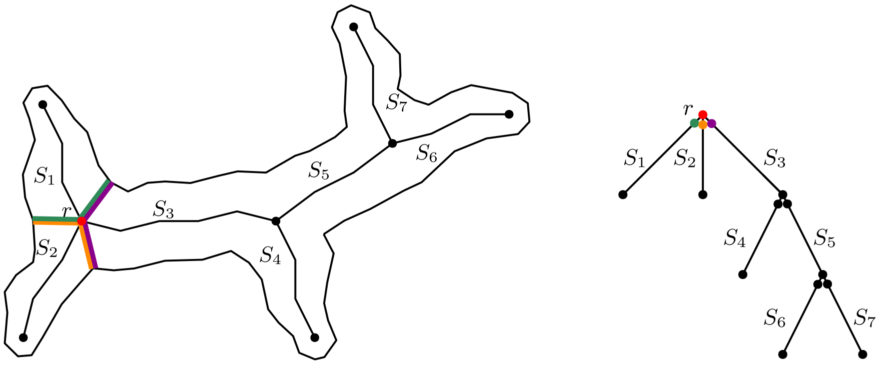


Figure 5 Representing the skeleton graph as a tree rooted at the point r.

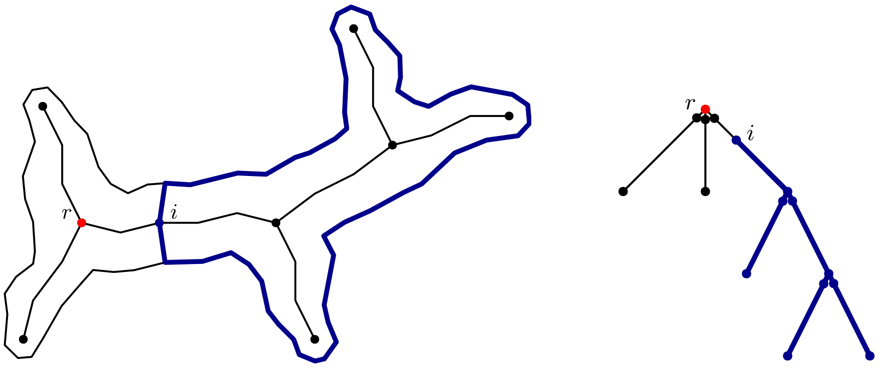


Figure 6 Subpolygon $P(i)$ ending in the skeleton point i and the corresponding subtree T_i.

- $P(i)$ is feasible or
- there exists a feasible polygon P' ending in i and feasible decompositions of the connected components of $P(i) \setminus P'$.

Thus, we have to consider all different combinations of skeleton points that together with i can form such a polygon P'. In a top-down manner, we consider the different combinations of nodes $[i_1, i_2, \dots, i_l]$ such that $i_j \in T_i$ and $T_{i_j} \cap T_{i_{j'}} = \emptyset$ for all $j \neq j'$. The polygon P' corresponds to the subtree rooted in i with $i_1, i_2, \dots, i_l$ as the leaves, depicted in blue in Fig. 7. Note that we can compute P' as a union of subpolygons iteratively. We check if P' is feasible and we have feasible decompositions for each $P(i_j)$, meaning every subtree T_{i_j} (gray in Fig. 7). Because of Observation 5, we know that $l \leq k$, for k being the number of leaves in the skeleton tree. We have a feasible decomposition of the whole polygon if there exists one of the polygon $P(r)$. This computation dominates the runtime with the maximum number of combinations to consider being in $\mathcal{O}(n^k)$. Note that this approach does not depend on the initial choice of the root node.

▶ **Theorem 7.** *Given a simple polygon P with skeleton S consisting of n points with degree at most three, one can compute a feasible decomposition of P based on S in $\mathcal{O}(n^k F)$ time, with k being the number of leaves in the skeleton tree and F as above.*

3.3 Feasibility constraints and optimization

Our decomposition method is highly versatile framework that can be adjusted for different feasibility constraints and optimization goals. In the following, we present a few examples of criteria we considered with regard to our application. As stated before, the ROI in our

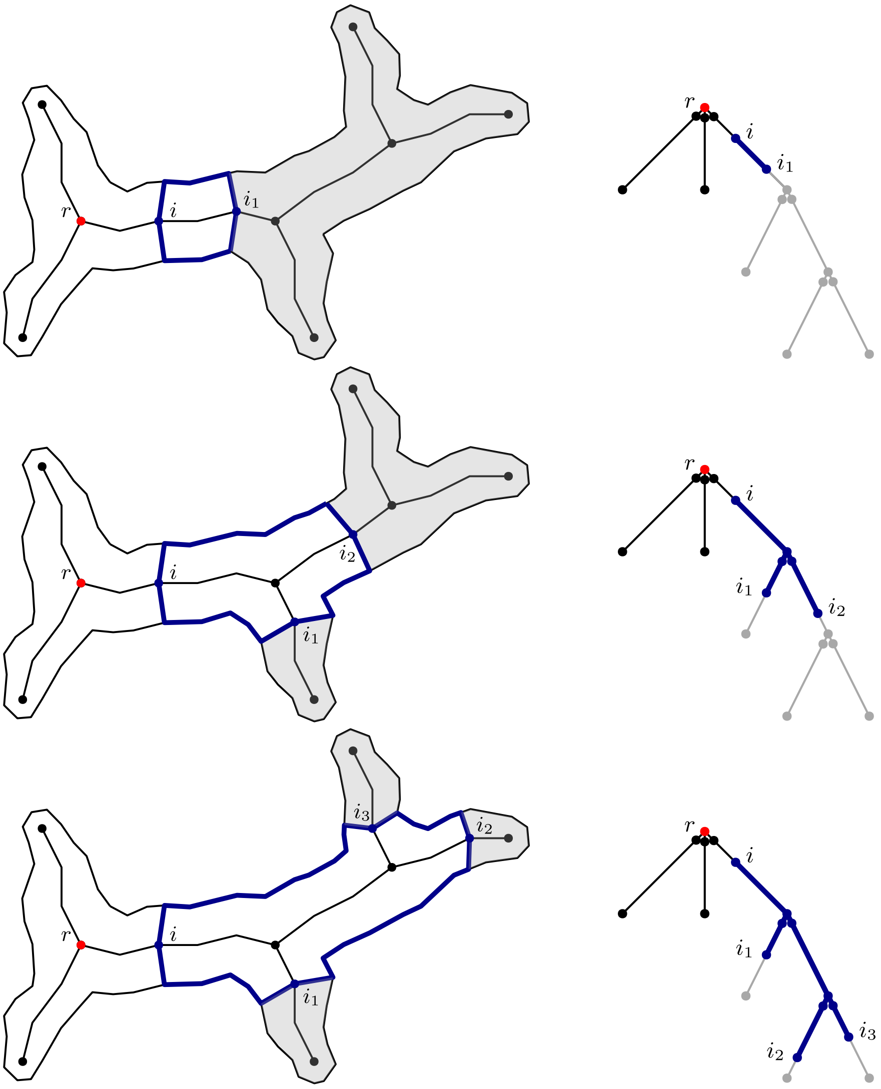


Figure 7 Different possibilities of skeletons points to consider when computing the decomposition of $P(i)$.

tissue sample needs to satisfy constraints in size and morphology for the LCM. In LCM, a laser separates a tissue fragment from its surrounding sample and a subsequent laser pulse catapults the fragment into a collecting device. On the one hand, the fragment has to have a certain size to ensure that enough material is supplied to be analyzed. On the other hand, the size cannot be too large otherwise the transferring process will fail. The transferring process also fails if the fragment has an irregular shape, since the laser pulse is concentrated on only one boundary point of the fragment. Specifically, elongated shapes or objects with narrow regions (bottlenecks) are problematic, as the tissue can tear and is only transferred partly or not at all. For simplicity, we consider only polygons with a linear skeleton in the following, but all mentioned constraints can be applied to the general decomposition method as well.

Feasibility constraints

First, we consider the size constraint by constraining the *area* of the subpolygons. Given two bounds l and u, a polygon P is feasible if $l \leq A(P) \leq u$, for $A(P)$ being the area of the polygon. Instead of the area one could also constrain the number of boundary points. Regarding a shape constraint for the polygons, we considered *approximate convexity* and *fatness*. For the convexity, a polygon P would be feasible if every inner angle lies between two given bounds. However, this does not prevent elongated shapes. By constraining the fatness of a polygon, we can achieve less elongated shapes. Fatness is defined by the aspect ratio $AR(P)$ of a polygon, which is the ratio of its width to its diameter. For a simple polygon, the diameter is defined as the diameter of the smallest enclosing circle and the width as the diameter of the largest inscribed circle. The aspect ratio lies in $[0, 1]$ and the higher its value the more circular and less elongated the shape is. We have the following constraint: A polygon is feasible if it is α-fat, meaning that $AR(P) \geq \alpha$ given some parameter α.

For all these feasibility criteria, we can compute all feasible subpolygons beforehand in time $\mathcal{O}(m)$ for m being the number of polygon vertices. Thus, we are able to verify if a subpolygon is feasible in constant time, which leads to an overall runtime of $\mathcal{O}(n^2 + m)$ to compute a feasible decomposition of the polygon.

Optimization goals

To solve the decision problem, we assign a value $X(i)$ to every skeleton point i. This value equals `True` if there exists a feasible decomposition of the polygon $P(i, n)$ and `False` otherwise. We initialize $X(1) = $ `True` and compute $X(i)$ as described before, that is $X(i) = $ `True` if there exists $j \geq i$ such that $P(i, j)$ is feasible and $X(j) = $ `True`. We can define $X(i)$ in different ways to achieve various optimization goals. For a point i, let I be the set of points j such that $P(i, j)$ are feasible. One possible optimization goal would be to find the *minimal decomposition* (MinNum). We define $X(i)$ as the number of subpolygons in an optimal feasible decomposition of $P(i, n)$, set $X(1) = 0$ and compute $X(i) = \min_{j \in I} X(j) + 1$. If one prefers the cuts to be at bottlenecks of the polygon, one could also consider *minimizing the length of the cut edges* (MinCut). Since every skeleton point i is the center of a maximal disk, we can obtain the cut length by the corresponding radius $r(i)$. We can define $X(i)$ either as the length of the longest cut or as the sum of cut lengths in the optimal decomposition of $P(i, n)$ and compute $X(i) = \max\{\min_{j \in I} X(j), r(i)\}$ resp. $X(i) = \min_{j \in I} X(j) + r(i)$. Another optimization goal we considered is *maximizing the fatness* (MaxFat). We define $X(i)$ as the smallest aspect ratio of a subpolygon in the decomposition and compute $X(i) = \max_{j \in I}\{\min\{X(j), AR(P(i, j))\}\}$. The runtime of the algorithm stays the same for different optimization goals. Obviously, one can combine different feasibility criteria and optimization goals.

The versatility of our algorithm allows the implementation of many different criteria. Note that for certain combinations other (faster) methods might exist. One example is finding the minimal (MinNum) decomposition in which the area of the subpolygons is bounded. For polygons with linear skeletons this can be modeled as finding the minimal segmentation of a weighted trajectory (in $\mathcal{O}(n \log n)$ time [1]). For general polygons, this problem can be modeled as computing the minimal (l, u)-partition of a weighted cactus graph (in $\mathcal{O}(n^6)$ time [6, 7]).

4 Results

We evaluated our algorithm by computing decompositions of tumor regions in infrared microscopic images of lung tissue samples that were identified using the method in [15]. With the tumor regions as our ROI, we decomposed each connected component by applying our algorithm on each branch of the corresponding skeletons separately. This approach follows the practical consideration that a polygon as a whole may not possess a feasible decomposition, while some individual branches do. We assessed the decomposition outcome in two respects. First, we present the differences in the decompositions when using different feasibility criteria and optimization goals. Then, we compare our decomposition approach based on the maximal fatness (MaxFat) to a heuristic recursive bisection method (BiSect) that was used to decompose tissue samples for LCM in previous work [15].

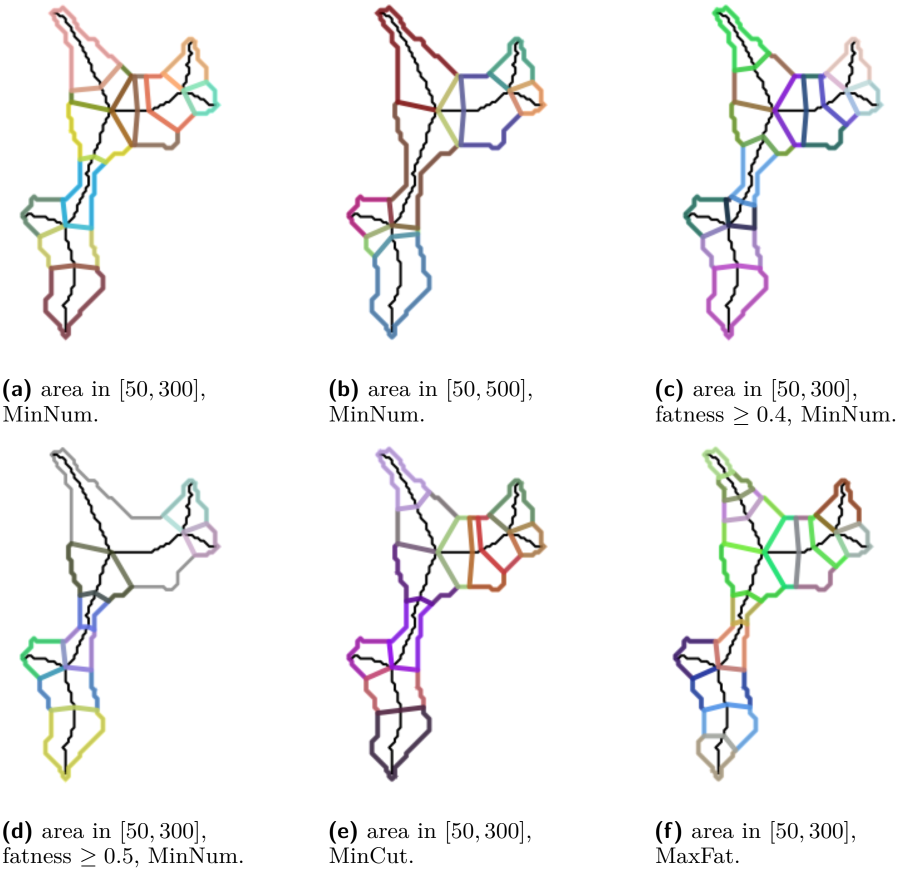
4.1 Comparison of feasibility constraints and optimization goals

Our algorithm facilitates the use of a wide range of feasibility criteria and optimization goals. The number of subpolygons as well as the positions of cuts depend on these constraints. Having a larger upper bound on, for example, the maximal area of a subpolygon results in fewer subpolygons, as illustrated in Fig. 8a and 8b. These decompositions both minimized the number of subpolygons, but the solutions are not necessarily unique and one optimal decomposition is chosen arbitrarily. This can be seen at the bottom-most skeleton branch in both polygons as the decomposition in Fig. 8a would be feasible with the constraints in 8b as well. In Fig. 8c and 8d, a fatness criterion has been added through a lower bound on the aspect ratio of subpolygons. This criterion avoids the tendency towards elongated subpolygons that can be observed in Fig. 8a. If the bounds in some constraints are too tight, a feasible decomposition might not exist. We illustrate this case in Fig. 8d, where the algorithm did not decompose the polygon parts depicted in gray. This is not favorable for our application, as it reduces the amount of extracted tissue material.

Regarding size as the feasibility criterion, we applied the different optimization goals described in Section 3.3, which we denoted by MinNum, MinCut and MaxFat. While choosing a different optimization criterion will not affect the area of the polygon that is successfully decomposed, the amount and positions of cut edges may change significantly. These changes may have an influence on the amount of successfully extracted fragments with LCM in practice later on. If we look at the decompositions of the top left skeleton branch, we can see that in Fig. 8a and 8e we have the same number of fragments, but with MinCut a cut with a lower length is chosen. However, maximizing the fatness of each subpolygon usually results in a higher number of subpolygons, but, as can be seen in Fig. 8f, each subpolygon is less elongated and somewhat rounder in shape. We expect these to be the desired shapes for our application. Thus, we used this optimization goal in the comparison of decomposition methods.

4.2 Comparison to BiSect

In this section, we evaluate our method with respect to its benefits for LCM. As mentioned before, the success of tissue extraction with LCM highly depends on the size and shape of the given tissue region. For the comparison, we applied the algorithm that maximizes the fatness (the aspect ratios) of the computed subpolygons while using a size constraint (lower and upper bound on the subpolygon's area). We denote this approach by *MaxFat* and compare it to a heuristic method we call *BiSect*. This method decomposes a polygon by



(a) area in $[50, 300]$, MinNum.

(b) area in $[50, 500]$, MinNum.

(c) area in $[50, 300]$, fatness ≥ 0.4, MinNum.

(d) area in $[50, 300]$, fatness ≥ 0.5, MinNum.

(e) area in $[50, 300]$, MinCut.

(f) area in $[50, 300]$, MaxFat.

Figure 8 Decomposition with different feasibility criteria and optimization goals.

recursively bisecting it if its area exceeds the upper size bound. If the area of a (sub)polygon is below the given lower bound, it is discarded. For technical reasons with LCM, this method is designed to leave a strip of tissue behind with every bisection (see Fig. 9).

We computed the decompositions of 10 different lung tissue samples with both methods and the same area bounds, namely, a minimal and maximal area of 100 px and 2800 px respectively. These 10 tissue samples contained 460 connected components of tumor regions as our regions of interest (ROI). Each ROI covered in average an area of 4100 px. Hence, for many ROIs the decomposition size with BiSect is fairly low. In fact, the average number of subpolygons per ROI was 2.3 for BiSect, but 9.3 for MaxFat.

Assessing the quality of results is of key importance for comparing results between the two methods. While an ideal assessment would compare the actual physical yield of tissue material, such experimental validation was not available in our present study, so that we rely on purely computational measures. Specifically, we employed two main measures: On the one hand, the amount of tissue loss in both decompositions, and on the other hand, the fatness of the resulting subpolygons, which as a single parameter captures reasonably how favourable a specific shape is for LCM. While MaxFat in fact optimizes towards obtaining fat fragments, it is still important to assess how far this translates into practice and how it compares to the fatness obtained by BiSect.

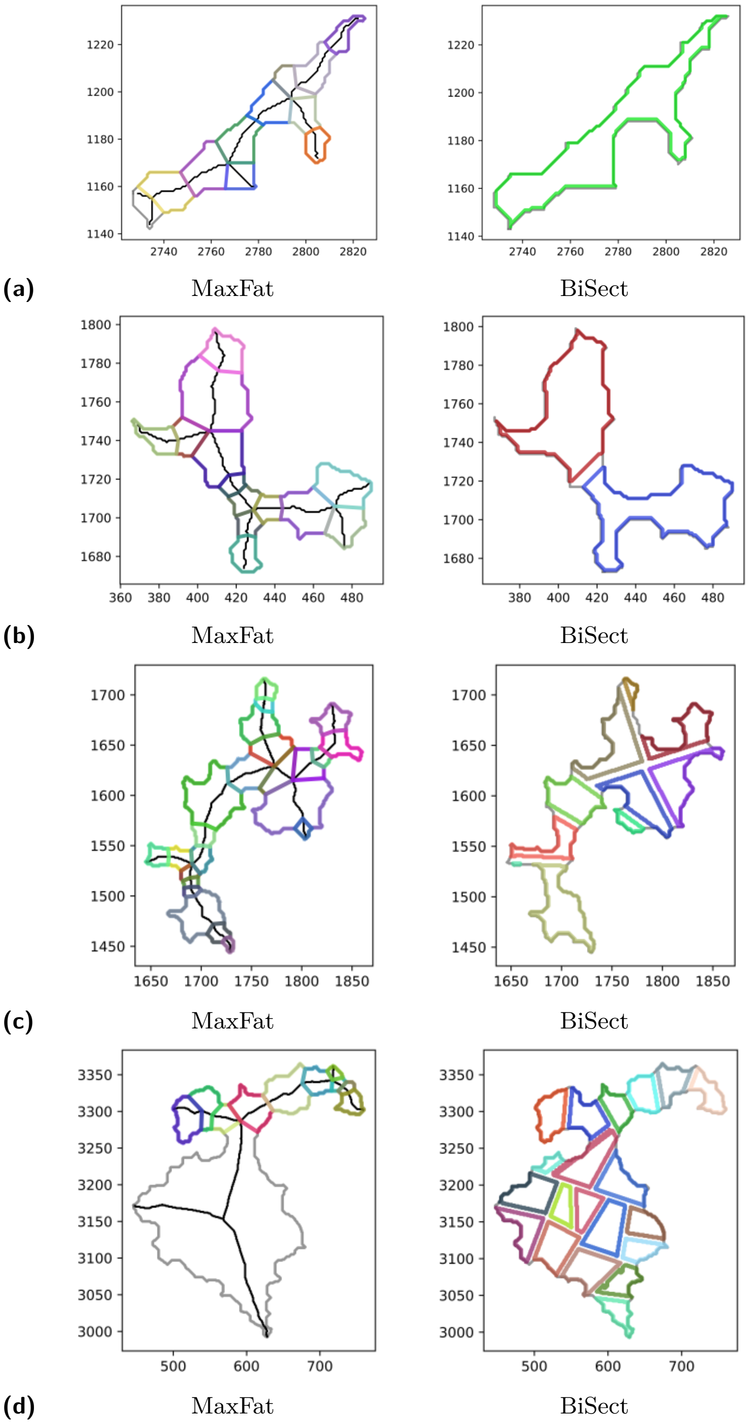


Figure 9 Four example shapes decomposed by MaxFat (left) and BiSect (right).

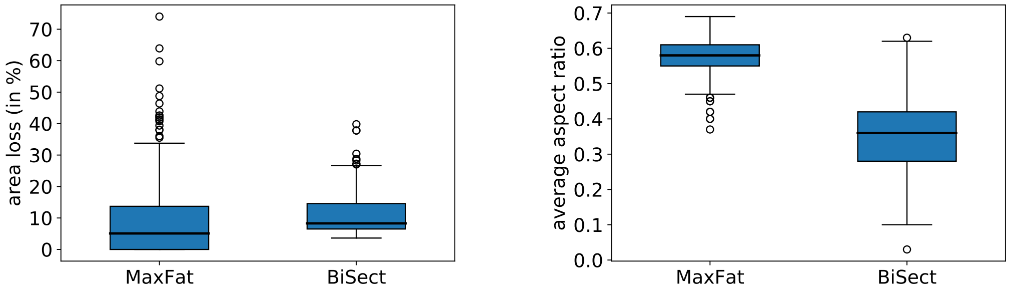
Area loss

First, we compare the amount of tissue loss (in percent) for each individual polygon/ROI. Both methods inherently involve tissue loss. In BiSect, each bisection results in some tissue loss, therefore, the amount of lost tissue rises with the number of subpolygons. In MaxFat, however, we lose tissue only if no feasible decomposition exists for a given skeleton branch. This mainly occurs when the corresponding (sub)polygon is either too slim or too wide. The first case is depicted in Fig. 9a: The polygons belonging to the bottom two skeleton branches (depicted in gray) are too small and thus no feasible decomposition was computed, resulting in this area being lost. This can be attributed to shortcomings of the underlying skeleton pruning method [2]. Improving the pruning of the skeleton may avoid such short branches. The second case of too wide (or fat) shapes is exemplified in Fig. 9d: In our approach, cut edges are introduced as line segments between a skeleton point and its closest boundary points. In the case of a wide shape being decomposed using a small upper bound for the size constraint, this leads to either thin-slicing or no feasible solution at all. This illustrates that our approach is tailored towards complex, ramified shapes rather than fat objects whose interior can be decomposed effectively through a simple grid pattern. It is also noteworthy that Polygon in Fig. 9d covers an area of around 43000 px and therefore presents an huge outlier in our sample.

The mean area loss with MaxFat is lower than the mean area loss with BiSect (see Fig. 10), yet with a greater variability in values and some high-loss outliers, which can be assigned to large and fat objects. While such objects do not occur frequently in our samples, the resulting area loss is obviously very high. This contributes to the higher standard deviation that we observe when considering the individual ROIs. Since the resulting fragments for each component in one tissue sample are collectively gathered, it is reasonable to validate on the level of samples and determine the entire yield for all ROIs in each sample. As can be seen in Table 1, the decomposition with MaxFat yields overall more tissue (with respect to area) to be collected for most tissue samples.

Fatness

It is important to note that the success of tissue collection using LCM depends not only on the size but also the shape of the fragments. BiSect applies merely a size constraint, whereas MaxFat considers both factors. While BiSect only decomposes polygons that exceed the



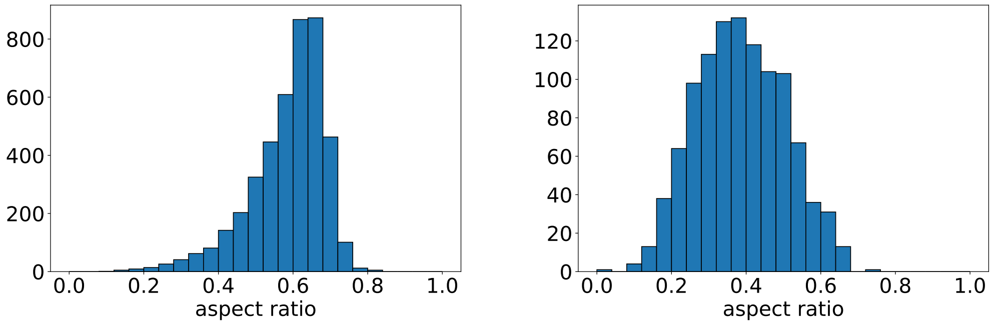
(a) MaxFat (M=8.84%, STD=11.18), BiSect (M=10.8%, STD=5.94).

(b) MaxFat (M=0.58, STD=0.05), BiSect (M=0.35, STD=0.11).

Figure 10 Comparison of the distribution of the percentage of area loss and the average fatness in the decompositions with MaxFat and BiSect for each ROI (n=460).

Table 1 Comparison of the yielded area (in %) of the different decomposition methods.

Sample	1	2	3	4	5	6	7	8	9	10
MaxFat	91.42	78.57	94.78	76.15	96.58	88.29	96.60	86.44	89.55	94.78
BiSect	89.38	81.96	86.56	76.59	86.53	80.98	87.00	85.41	81.52	86.56



(a) MaxFat (n=4285, M=0.58, STD=0.1) **(b)** BiSect (n=1066, M=0.38, STD=0.12)

Figure 11 Distributions of the aspect ratios for each subpolygon in the decompositions with MaxFat and BiSect for all samples combined.

given upper size bound, MaxFat searches for an decomposition in which the subpolygons have the highest possible aspect ratios. This obviously results in a higher number of subpolygons over all samples: MaxFat resulted in 4285 and BiSect in 1066 fragments. In some cases, the size of a ROI did not make it necessary to place any cut when decomposing with BiSect, but the shape of the ROI oftentimes is too complex or irregular for LCM to be successful. Additionally, the cut placement with BiSect is far from being optimal.

As can be seen in the examples of Fig. 9b and 9c, the shapes resulting from BiSect are highly irregular – except for most "internal" subpolygons in large, round ROI. In most cases, the resulting subpolygons are far from being convex (in an approximate sense), but elongated and show narrow bottlenecks. All these shape properties pose a risk for the tissue to tear in the extraction process. With the MaxFat approach, however, we overall receive less elongated and rounder shapes. When comparing the average fatness – meaning the average of aspect ratios of the subpolygons – in the decompositions for each ROI (Fig. 10), we clearly achieved higher values with MaxFat with a smaller variability. In fact, for over 75 % of ROI our method achieved an average fatness higher than 0.5, whereas with BiSect nearly 75 % of ROI have an average fatness lower than 0.4. When comparing the distribution of aspect rations for the individual subpolygons, it is revealed that BiSect shows the pattern of a normal distribution, whereas the distribution for MaxFat is clearly left-skewed (see Fig. 11). This shows that without applying additional shape constraints the decomposition does not result in fragments of the desired shape whereas our method consistently obtains such fragments.

5 Conclusion

In this paper, we presented a skeleton-based decomposition method for simple polygons as a novel approach to decompose disease-specific regions of tissue samples while aiming to optimize the amount of tissue obtained by laser capture microdissection (LCM). Compared to naive heuristic approaches that are currently used, our approach provably optimizes target functions under side constraints that are tailored towards relevance for tissue yield in

LCM, as this requires fragments of suitable size and morphology to minimize tissue loss. As we demonstrated, these theoretical properties translate into practice when comparing our approach to a recursive bisection approach that considers fragment size as the only criterion for decomposition.

Our approach is designed towards complex morphological structures that are commonly found in cancerous tissue and are usually the most challenging to extract using LCM without major loss of tissue mass. Not surprisingly, it does not perform well when fragmenting relatively fat shapes into small fragments. In this case, it is expected to yield thin slices not ideal for LCM or find no feasible decomposition at all. Yet, such fat shapes does not occur frequently and can easily be decomposed using simple approaches, e.g. bisection-based approaches, without major tissue loss. Thus, we expect to achieve the best outcome for the practical application if we combine both approaches. Our method can be used for the majority of the complex tissue regions and for simple fat morphologies, which can be easily distinguished and separated, other approaches can be applied.

The implementation of our approach relies on a skeletonization of the underlying polygons. Specifically, we utilizes the approach by Bai et al. [2], which implements a heuristic pruning approach. It is likely that recent improvements for skeletonization and pruning [12, 13] will further improve results, as in particular the pruning step of these recent methods promises to avoid short and other spurious branches which negatively affect the amount of yielded tissue in our approach.

Finally, our validation is merely based on quantitative morphological indicators about the resulting fragments. For future work, it will be important to validate the improvements experimentally, e.g. by comparing the actual yield of protein or DNA from different approaches. Overall, our work contributes to further optimization and automation of LCM and thus promises to contribute to the further maturing of the technology and enhancing its suitability for systematic use in larger scale clinical studies.

References

1 Sander P. A. Alewijnse, Kevin Buchin, Maike Buchin, Andrea Kölzsch, Helmut Kruckenberg, and Michel A. Westenberg. A framework for trajectory segmentation by stable criteria. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '14, page 351–360, New York, NY, USA, 2014. Association for Computing Machinery.

2 Xiang Bai, Longin Jan Latecki, and Wen-Yu Liu. Skeleton pruning by contour partitioning with discrete curve evolution. *IEEE transactions on pattern analysis and machine intelligence*, 29(3), 2007.

3 Harry Blum. A transformation for extracting new descriptors of shape. *Models for Perception of Speech and Visual Forms, 1967*, pages 362–380, 1967.

4 Harry Blum. Biological shape and visual science (part i). *Journal of theoretical Biology*, 38(2):205–287, 1973.

5 Maike Buchin, Axel Mosig, and Leonie Selbach. Skeleton-based decomposition of simple polygons. In *Abstracts of 35th European Workshop on Computational Geometry*, 2019. URL: `http://www.eurocg2019.uu.nl/papers/3.pdf`.

6 Maike Buchin and Leonie Selbach. Decomposition and partition algorithms for tissue dissection. In *Computational Geometry: Young Researchers Forum*, pages 24–27, 2020. URL: `http://www.computational-geometry.org/YRF/cgyrf2020.pdf`.

7 Maike Buchin and Leonie Selbach. A polynomial-time partitioning algorithm for weighted cactus graphs, 2020. `arXiv:2001.00204`.

8 Hyeong In Choi, Sung Woo Choi, and Hwan Pyo Moon. Mathematical theory of medial axis transform. *pacific journal of mathematics*, 181(1):57–88, 1997.

9 Soma Datta, Lavina Malhotra, Ryan Dickerson, Scott Chaffee, Chandan K Sen, and Sashwati Roy. Laser capture microdissection: Big data from small samples. *Histology and histopathology*, 30(11):1255, 2015.

10 Gabriella Sanniti di Baja and Edouard Thiel. (3, 4)-weighted skeleton decomposition for pattern representation and description. *Pattern Recognition*, 27(8):1039–1049, 1994.

11 Klaus Drechsler and Cristina Oyarzun Laura. Hierarchical decomposition of vessel skeletons for graph creation and feature extraction. In *Bioinformatics and Biomedicine (BIBM), 2010 IEEE International Conference on*, pages 456–461. IEEE, 2010.

12 Bastien Durix, Sylvie Chambon, Kathryn Leonard, Jean-Luc Mari, and Géraldine Morin. The propagated skeleton: A robust detail-preserving approach. In *International Conference on Discrete Geometry for Computer Imagery*, pages 343–354. Springer, 2019.

13 Bastien Durix, Géraldine Morin, Sylvie Chambon, Jean-Luc Mari, and Kathryn Leonard. One-step compact skeletonization. In *Eurographics (Short Papers)*, pages 21–24, 2019.

14 Michael R Emmert-Buck, Robert F Bonner, Paul D Smith, Rodrigo F Chuaqui, Zhengping Zhuang, Seth R Goldstein, Rhonda A Weiss, and Lance A Liotta. Laser capture microdissection. *Science*, 274(5289):998–1001, 1996.

15 Frederik Großerueschkamp, Thilo Bracht, Hanna C Diehl, Claus Kuepper, Maike Ahrens, Angela Kallenbach-Thieltges, Axel Mosig, Martin Eisenacher, Katrin Marcus, Thomas Behrens, et al. Spatial and molecular resolution of diffuse malignant mesothelioma heterogeneity by integrating label-free ftir imaging, laser capture microdissection and proteomics. *Scientific reports*, 7(1):1–12, 2017.

16 J Mark Keil. Polygon decomposition. *Handbook of Computational Geometry*, 2:491–518, 2000.

17 Yutaka Kondo, Yae Kanai, Michiie Sakamoto, Masashi Mizokami, Ryuzo Ueda, and Setsuo Hirohashi. Genetic instability and aberrant dna methylation in chronic hepatitis and cirrhosis—a comprehensive study of loss of heterozygosity and microsatellite instability at 39 loci and dna hypermethylation on 8 cpg islands in microdissected specimens from patients with hepatocellular carcinoma. *Hepatology*, 32(5):970–979, 2000.

18 Kathryn Leonard, Geraldine Morin, Stefanie Hahmann, and Axel Carlier. A 2d shape structure for decomposition and part similarity. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 3216–3221. IEEE, 2016.

19 Martha L. Narro, Fan Yang, Robert Kraft, Carola Wenk, Alon Efrat, and Linda L. Restifo. Neuronmetrics: Software for semi-automated processing of cultured neuron images. *Brain Research*, 1138:57–75, 2007.

20 Chin-Ann J Ong, Qiu Xuan Tan, Hui Jun Lim, Nicholas B Shannon, Weng Khong Lim, Josephine Hendrikson, Wai Har Ng, Joey WS Tan, Kelvin KN Koh, Seettha D Wasudevan, et al. An optimised protocol harnessing laser capture microdissection for transcriptomic analysis on matched primary and metastatic colorectal tumours. *Scientific Reports*, 10(1):1–12, 2020.

21 Nikos Papanelopoulos, Yannis Avrithis, and Stefanos Kollias. Revisiting the medial axis for planar shape decomposition. *Computer Vision and Image Understanding*, 179:66–78, 2019.

22 Dennie Reniers and Alexandru Telea. Skeleton-based hierarchical shape segmentation. In *Shape Modeling and Applications, 2007. SMI'07. IEEE International Conference on*, pages 179–188. IEEE, 2007.

23 Punam K Saha, Gunilla Borgefors, and Gabriella Sanniti di Baja. A survey on skeletonization algorithms and their applications. *Pattern Recognition Letters*, 76:3–12, 2016.

24 Martin R Schmuck, Thomas Temme, Katharina Dach, Denise de Boer, Marta Barenys, Farina Bendt, Axel Mosig, and Ellen Fritsche. Omnisphero: a high-content image analysis (hca) approach for phenotypic developmental neurotoxicity (dnt) screenings of organoid neurosphere cultures in vitro. *Archives of toxicology*, 91(4), 2017.

25 Mirela Tănase and Remco C Veltkamp. Polygon decomposition based on the straight line skeleton. In *Geometry, Morphology, and Computational Imaging*, pages 247–268. Springer, 2003.

26 Quoc Dang Vu, Simon Graham, Tahsin Kurc, Minh Nguyen Nhat To, Muhammad Shaban, Talha Qaiser, Navid Alemi Koohbanani, Syed Ali Khurram, Jayashree Kalpathy-Cramer, Tianhao Zhao, et al. Methods for segmentation and classification of digital microscopy tissue images. *Frontiers in bioengineering and biotechnology*, 7, 2019.

27 Chaofeng Wang, Cai-Ping Gui, Hai-Kuan Liu, Dong Zhang, and Axel Mosig. An image skeletonization-based tool for pollen tube morphology analysis and phenotyping f. *Journal of integrative plant biology*, 55(2):131–141, 2013.

28 Shidan Wang, Ruichen Rong, Donghan M Yang, Junya Fujimoto, Shirley Yan, Ling Cai, Lin Yang, Danni Luo, Carmen Behrens, Edwin R Parra, et al. Computational staining of pathology images to study the tumor microenvironment in lung cancer. *Cancer Research*, 2020.

29 Franz-Erich Wolter. Cut locus and medial axis in global shape interrogation and representation, 1993.

The Bourque Distances for Mutation Trees of Cancers

Katharina Jahn
Department of Biosystems Science and Engineering, ETH Zurich, Basel, Switzerland
SIB Swiss Institute of Bioinformatics, Basel, Switzerland

Niko Beerenwinkel
Department of Biosystems Science and Engineering, ETH Zurich, Basel, Switzerland
SIB Swiss Institute of Bioinformatics, Basel, Switzerland

Louxin Zhang
Department of Mathematics and Computational Biology Program, National University of Singapore, Singapore
matzlx@nus.edu.sg

Abstract

Mutation trees are rooted trees of arbitrary node degree in which each node is labeled with a mutation set. These trees, also referred to as clonal trees, are used in computational oncology to represent the mutational history of tumours. Classical tree metrics such as the popular Robinson–Foulds distance are of limited use for the comparison of mutation trees. One reason is that mutation trees inferred with different methods or for different patients often contain different sets of mutation labels. Here, we generalize the Robinson–Foulds distance into a set of distance metrics called Bourque distances for comparing mutation trees. A connection between the Robinson–Foulds distance and the nearest neighbor interchange distance is also presented.

2012 ACM Subject Classification Applied computing → Bioinformatics

Keywords and phrases mutation trees, clonal trees, tree distance, phylogenetic trees, tree metric, Robinson–Foulds distance, Bourque distance

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.14

Funding LX Zhang was supported by Singapore Ministry of Education Academic Research Fund Tier-1 [grant R-146-000-238-114].

1 Introduction

Trees have been used in biology to model the evolution of species, genes and cancer cells [15, 32, 39]; to represent the secondary structures of RNA molecules and to classify cell types, to name just a few uses [23, 37]. A fundamental issue arising from these applications of trees is how to quantitatively compare tree models that are inferred by different methods or from different data. A number of tree metrics have been proposed for comparisons, including the Robinson–Foulds (RF) [3, 35, 36], nearest-neighbor interchange (NNI) [31, 35] and triple(t) distances [7] for phylogenetic trees; gene duplication, gene loss and reconciliation costs [17, 27] for gene and species trees; and the tree-edit distances [40, 37, 43] for tree models of secondary RNA structures, etc. [2, 21, 26, 33, 41].

With advances in next-generation sequencing and single-cell sequencing technologies, a large amount of genomic data is now available for identifying tumour subclones and inferring their evolutionary relationships. The most common representation of these relationships are mutation trees, also known as clonal trees, which encode the (partial) temporal order in which mutations were acquired. Formally, a mutation tree on a finite set of mutations Γ is a rooted tree T with k nodes and a partition of Γ into k disjoint non-empty parts P_i so that each P_i is assigned as the label of a node of T [16, 32]. A large number of computational approaches

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti; Article No. 14; pp. 14:1–14:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

for reconstructing mutation trees from bulk sequencing data [9, 11, 14, 29, 34], single-cell sequencing data [6, 13, 19, 42], or a combination of both [30, 28] have been developed over the last years. Unlike phylogenetic trees, mutation trees inferred with these methods will not only differ in their topology but may also be defined on different sets of mutations. The latter happens in the comparison of methods using different data (e. g. single-cell vs. bulk) or divergent criteria for mutation calling. For that reason, classical tree distance measures are not immediately applicable to mutation trees. Instead novel measures have recently been developed [1, 4, 5, 10, 18, 20], but no standard approach for mutation tree comparison has yet emerged. Instead, shortcomings of some of these measures such as the inability to resolve major differences between trees have recently been demonstrated [5]. Additionally, computing the distances between two mutation trees takes at east quadratic time for each of these measures.

Here, we generalize the Robinson-Foulds metric, a classic distance measure for unrooted trees, for the comparison of mutation trees. This metric is based on the so-called (edge) contraction and decontraction operations introduced by Bourque for leaf-labeled unrooted trees in a study of Steiner trees [3]. A contraction on an edge $(u.v)$ of a tree T is an operation that transforms T into a new tree by shrinking (u, v) into a single node. The decontraction operation is the reverse of contraction. Robinson and Foulds independently adopted the contraction and decontraction to define a metric of unrooted labeled trees, where there is a finite set S and a partition of S into disjoint parts (some of which may be empty) so that nodes with a degree of at most 2 are each labeled with a unique non-empty part, and nodes with a degree of at least 3 are labeled with either a unique non-empty part or an empty part. They defined a metric, now called the Robinson-Foulds (RF) distance, in which the distance between two unrooted labeled trees is the minimum number of contraction or decontraction operations that are necessary to transform one into another [36]. The RF distance is equal to the number of edge-induced partitions that are not shared between the two trees and thus is computable in linear time [8].

Although the RF distance is popular in phylogenetic analysis, it is not robust when applied to the comparison of mutation trees with different sets of mutations, as it is simply equal to the total number of edges in the trees and thus fails to capture any topological similarity between the trees.

In this paper, by generalizing the RF distance, we propose a collection of distance measures to measure the topological dissimilarity between unrooted (resp. rooted) labeled trees with different label sets. We also apply these measures to simulated and real tumour mutation trees. To set our distances apart from another recently introduced generalised RF distance that is based on a node flip operation [4], we refer to our generalisations as *Bourque distances*, as they are closely related to the edge contraction and decontraction operations introduced by Bourque for leaf-labeled unrooted trees [3]. They are also shown to be related to the NNI distance [35]. Unlike previous measures proposed for the comparison of mutation trees, the basic version of the Bourque distance can be computed in linear time.

The rest of this paper is divided into seven sections. Section 2 introduces basic concepts and the notation that will be used. In Section 3, we present a connection between the NNI distance and the RF distance for both phylogenetic and arbitrary trees that are unrooted and labeled. In Section 4, we generalize the RF distance into the Bourque distances for unrooted labeled trees. In Section 5, we define the Bourque distances for mutation trees. In Section 6, we examine the relationships among the distance measures proposed in [10, 20, 5] and the Bourque distances on rooted 7-node trees and on random rooted trees with 30 nodes. In Section 7, we computed the Bourque distances on two sets of mutation trees. Section 8 concludes the study with a few remarks.

2 Concepts and Notation

A (unrooted) *tree* is an acyclic graph. A *rooted tree* is a directed tree with a designated root node ρ in which the edges are oriented away from ρ. There is a unique directed path from ρ to every other node.

For a tree or rooted tree T, the nodes, leaves and edges are denoted $V(T)$, $\text{Leaf}(T)$ and $E(T)$, respectively. Let $u \in V(T)$. The *degree* of u is the number of edges incident to it, where edge orientation is ignored if T is rooted. In a rooted tree, non-root nodes with a degree of one are called the *leaves*; non-leaf nodes are called *internal* nodes. One or more edges may leave an internal node, but exactly one edge enters every node that is not the root. An *internal edge* is an edge between two internal nodes.

Let T be a rooted tree and $u, v \in V(T)$. The node v is called a *child* of u and u is called the *parent* of v if $(u, v) \in E(T)$. The node v is a *descendant* of u and u is an *ancestor* of v if the unique path from the tree root to v contains u. We use $C_T(u)$, $A_T(u)$ and $D_T(u)$ to denote the set of all children, ancestors and descendants of u in T, respectively. Note that u is in neither $A_T(u)$ nor $D_T(u)$.

A *star tree* is a tree that contains only one non-leaf node, which is called the *center* of the tree. A *line tree* is a tree in which every internal node is of degree 2. A rooted line tree is a line tree whose root is of degree 1.

A tree is *binary* if every internal node is of degree 3. A rooted tree is *binary* if the root is of degree 2 and every other internal node is of degree 3. A *caterpillar tree* is a binary tree in which each internal node is adjacent to one or two leaves.

Let X be a finite set. A (rooted) *phylogenetic tree* on X is a binary (rooted) tree where the leaves are uniquely labeled with the elements of X, the taxon set. A (rooted) phylogenetic tree T on X is *labeled* if there is a set I that is disjoint from X and a labeling function $\ell : V(T) \setminus \text{Leaf}(T) \to I$ such that each u of $V(T) \setminus \text{Leaf}(T)$ is labeled with $\ell(u)$. If ℓ is a one-to-one function, T is said to be uniquely labeled. In a labeled phylogenetic tree, the label set for the internal nodes and the taxon set for the leaves are distinct and thus are not interchangeable.

A tree or rooted tree T with n nodes is *labeled* if there is a finite set M and a labeling function $\ell : V(T) \to 2^M$ satisfying $\cup_{v \in V(T)} \ell(v) = M$ and $\ell(v) \neq \emptyset$ for $v \in V(T)$ so that $f(v)$ is assigned as the label of v, where 2^M denotes the collection of subsets of M. Furthermore, if $\ell(v)$ contains exactly one element for each node v, we say T is *1-labeled* with L. Here, M is called the *label set* of T.

A *mutation tree* on a set M of mutations is a rooted labeled tree that has M as the label set, where the labels of different nodes are disjoint.

3 Metrics for labeled trees

For convenience, we will introduce new metrics on the space of 1-labeled trees and then generalize them to the space of mutation trees later.

3.1 Nearest neighbor interchanges on labeled phylogenetic trees

The NNI operation (Fig. 1A) and NNI distance were originally introduced for binary phylogenetic trees [35]. It is known that any binary phylogenetic tree can be transformed into another in $n \log n + 2n - 4$ NNIs at most [24]. The NNI operation for rooted phylogenetic trees is given in Fig. 1B. Since the NNI operation does never interchange the labels of internal nodes and of leaves, Proposition 1 is simple, but as far as we know, it has never appeared in literature.

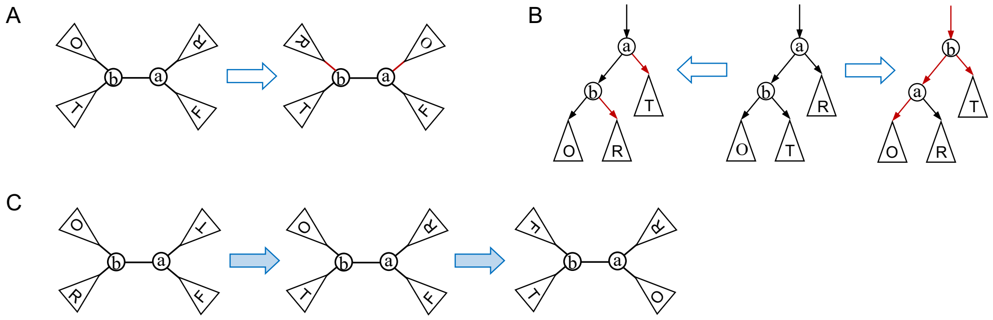


Figure 1 Illustration of the NNI operation on phylogenetic trees. (A) In a phylogenetic tree, an NNI operation on an internal edge (a, b) first selects two edges (a, x) and (b, y) that are, respectively, incident to a and b such that $(a, x) \neq (a, b) \neq (y, b)$; it then rewires them to the opposite end so that (a, y) and (b, x) are the two edges in the resulting tree (red). Since a and b are labeled differently, a unrooted tree can be transformed into one of four possible trees in one NNI. (B) In a rooted phylogenetic tree T, an NNI operation on an internal edge (a, b) (where b is a child of a) transforms T by either (i) selecting two edges (a, x) and (b, y) that leave from a and b, respectively, and replacing them with (a, y) and (b, x) (left), where $x \neq b$, or (ii) selecting an edge (b, y) leaving from b and replacing the unique edge (z, a) that enters a, (a, b) and (b, y) with (z, b), (b, a) and (a, y) (right), respectively. A rooted tree can be transformed into four different trees in one NNI. (C) Illustration of the interchange of two labels of the ends of an internal edge in two NNIs in an 1-labeled phylogenetic tree.

▶ **Proposition 1.** *In the space of binary (resp. rooted) phylogenetic trees where the internal nodes are 1-labeled, any tree can be transformed into another.*

Proof. This follows from the fact that two NNIs on an internal edge (a, b) are enough to exchange the labels of a and b (Fig. 1C). A similar fact is also true for binary rooted phylogenetic trees. ◀

3.2 Generalized NNI on 1-labeled trees

An arbitrary tree with n nodes can have 1 to $n-2$ internal nodes of degree ≥ 2. To transform a tree into any other of the same size with the same label set, we define the generalized NNI (gNNI) operation as follows.

▶ **Definition 2.** *Let T be a 1-labeled tree and $e = (a, b) \in E(T)$. A gNNI on e is an operation that transforms T into a new tree S by (i) selecting a subset C_a and a subset C_b of the edges that are, respectively, incident to a and b such that $e \notin C_a \cup C_b$ and then (ii) replacing each edge (a, x) of C_a with (b, x) and each edge (b, y) of C_b with (a, y).*

The gNNI operation is illustrated in Fig. 2. Note that if we apply a gNNI operation on an edge $e = (a, b)$ to reconnect all the children of a to b while keeping the children of b unmoved, a will become a leaf adjacent to b in the resulting tree. Another difference between gNNI and NNI is that gNNI can be applied to any edge, whereas NNI can only be applied on an internal edge.

Let L be a set of n elements. The gNNI graph $G_{\text{gnni}}(L)$ is defined as a graph in which the nodes are all 1-labeled trees with nodes labeled with L and two trees are connected by an edge if the two trees are one gNNI apart. The diameter of $G_{\text{gnni}}(L)$ is written as $D(G_{\text{gnni}}(L))$. The distance between two trees T' and T'' in the graph is called the *gNNI distance* between them, written as $d_{\text{gnni}}(T', T'')$.

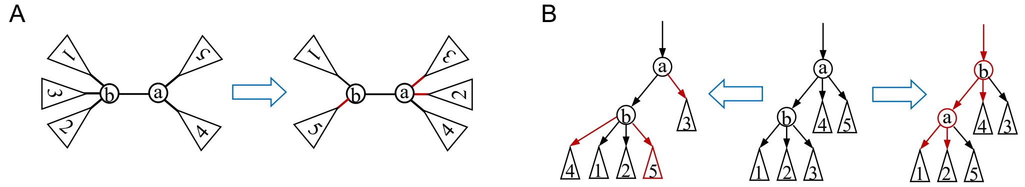


Figure 2 An illustration of the gNNI operation on a labeled tree (A) or a rooted labeled tree (B). A. A gNNI operation on an edge (a, b) interchanges one or more children of a with an arbitrary number of children of b. B. A gNNI operation on an edge (a, b) (where b is the child of a) not only rewires the selected edges leaving a and b (left), but also rewires the unique edge entering a and b simultaneously if necessary (right).

▶ **Proposition 3.** *Let L be a set of n elements. The graph $G_{\text{gnni}}(L)$ has the following properties:*

- $|V(G_{\text{gnni}}(L))| = n^{n-2}$;
- $G_{\text{gnni}}(L)$ *is connected;*
- $n - 2 \leq D(G_{\text{gnni}}(L)) \leq 2n - 4$

Proof. The first property is the Cayley formula on the count of 1-labeled trees with n nodes. The second property is a consequence of the third. We prove the third property as follows.

Let $T_1, T_2 \in V(G_{\text{gnni}}(L))$. Let r_1 and r_2 be the two nodes of T_1 and T_2, respectively, that have the same label. Each n-node tree has at least two leaves and therefore $n - 2$ internal nodes at most. By applying a gNNI operation on an edge (r_1, u), we can reconnect all the subtrees that each contain exactly one neighbor of u to r_1, producing a tree in which u becomes a leaf adjacent to r_1. By continuing to apply the gNNI operation on the edges between r_1 and its non-leaf neighbors, we can transform T_1 into the star tree centered at r_1 in $n - 2$ gNNIs at most. In reverse, we can transform the star tree centered at r_2 into T_2 in $n - 2$ gNNIs at most. By combining these two transformations, we transform T_1 into T_2 by using $2n - 4$ gNNIs at most. This proves the upper bound of the third property.

Let S be a line tree where the leaves are labeled with a and b and let T be a 1-labeled star tree centered at the node of the label a. The distances between a and b are $(n - 1)$ and 1 in S and T, respectively. It takes at least $(n - 2)$ gNNIs to transform S to T, as each gNNI can only decrease the distance between a and b by 1. This proves the lower bound of the third property. ◀

Let T be a tree in $G_{\text{gnni}}(L)$. We use $d(u, v)$ to denote the number of edges in the unique path between u and v in T. Any edge $(u, v) \in E(T)$ induces a two-part partition $P(e) = \{P_u, P_v\}$ of L, where $P_u = \{\ell(x) \mid d(x, u) < d(x, v)\}$, which contains u, and $P_v = \{\ell(y) \mid d(y, v) < d(y, u)\}$, which contains v. Let us define $\mathcal{P}(T) = \{P(e) \mid e \in E(T)\}$.

▶ **Proposition 4.** *For any two 1-labeled trees S, T of $G_{\text{gnni}}(L)$,*

$$\frac{1}{2}|\mathcal{P}(S)\Delta\mathcal{P}(T)| \leq d_{\text{gnni}}(S, T) < |\mathcal{P}(S)\Delta\mathcal{P}(T)|,$$

where Δ is the set symmetric difference operator.

Proof. Let S and T have n nodes in the tree space. The first inequality is derived from the following two facts:

- $\mathcal{P}(S) \setminus \mathcal{P}(T)$ contains exactly one partition $P(e)$ if T is obtained from S by applying a gNNI on any $e \in E(S)$;
- $A\Delta B \subseteq (A\Delta C) \cup (C\Delta B)$ for any three sets.

To prove the upper bound, we let $m = |\mathcal{P}(S) \cap \mathcal{P}(T)|$ and let

$$\mathcal{P}(S) \cap \mathcal{P}(T) = \{P(e'_1), P(e'_2), \cdots P(e'_m)\} = \{P(e''_1), P(e''_2), \cdots P(e''_m)\},$$

where $e'_i \in E(S), e''_i \in E(T)$ such that $P(e'_i) = P(e''_i)$ for each i. $S - \{e'_i | 1 \leq i \leq m\}$ is the disjoint union of $m+1$ subtrees S_j $(0 \leq j \leq m)$; similarly, $T - \{e''_i | 1 \leq i \leq m\}$ is the disjoint union of $m+1$ subtrees T_i $(0 \leq i \leq m)$. Additionally, for each $0 \leq j \leq m$, a unique index $k(j)$ exists such that S_j and $T_{k(j)}$ contain the same number (say o_i) of nodes. Note that

$$|\mathcal{P}(S) \Delta \mathcal{P}(T)| + 2m = |E(S)| + |E(T)| = 2n - 2. \tag{1}$$

There are three possible cases for each pair of subtrees S_j and $T_{k(j)}$. First, if $o_j = 1$, we do not need to do any local adjustments of S_j to transform S to T.

If both S_j and $T_{k(j)}$ contain two nodes u and v, (u, v) is then the only edge of S_j and $T_{k(j)}$. This implies that the two nodes are the ends of different edges of $\mathcal{P}(S) \cap \mathcal{P}(T)$ in S and T, and thus we need one gNNI to switch these two nodes in S so that they are incident to the same edges as in T after the operation.

If both S_j and $T_{k(j)}$ contain o_j (≥ 3) nodes, we select an internal node s of S_j and a node t of $T_{k(j)}$ such that s and t have the same label. By continuing to apply, at most, $o_j - 3$ gNNIs on the edges incident to s, we can transform S_j into a star tree C centered on s, as s is an internal node. Similarly, by applying $o_j - 2$ gNNIs at most, we can transform C into $T_{k(j)}$. Taken together, the two transformations give a transformation from S_j into $T_{k(j)}$ consisting of at most $2o_j - 5$ gNNIs at most.

Let m_i be the number of subtrees S_j such that $|S_j| = i$ for $i = 1, 2$ and let m_3 be the number of subtrees S_j such that $|S_j| \geq 3$. We have that $m_1 + m_2 + m_3 = m + 1$ and there are $n - m_1 - 2m_2$ nodes in the union of all subtrees S_j in Case 3. By combining all the transformations from S_j to $T_{k(j)}$, we can transform S to T in c gNNIs at most, where:

$$\begin{aligned} c &= 0 + m_2 + [2(n - m_1 - 2m_2) - 5m_3] \\ &= 2n - 2m_1 - 3m_2 - 3m_3 - 2m_3 \\ &= 2n - 2m_1 - 3m_2 - 3m_3 - 2(m + 1 - m_1 - m_2) \\ &= 2n - 2m - 2 - m_2 - 3m_3 \end{aligned}$$

Since $m_2 \geq 0$ and $m_3 \geq 0$, by Eqn. (1), $c \leq 2n - 2m - 2 = |\mathcal{P}(S) \Delta \mathcal{P}(T)|$. ◀

3.3 The RF distance

Let S and T be two 1-labeled trees. $|\mathcal{P}(S) \Delta \mathcal{P}(T)|$ is called the *RF distance* between S and T, denoted $\mathrm{RF}(S, T)$. For example, in the left tree given in Fig. 3A, the edge $(2, 4)$ (bold) induces the two-part partition $\{\{1, 2, 3\}, \{4, 5, 6, 7, 8\}\}$; the edge $(7, 8)$ (bold) induces $\{\{7\}, \{1, 2, 3, 4, 5, 6, 8\}\}$. These two partitions are not equal to any edge-induced partition in the right tree. Similarly, we have that the two-part partitions induced by the edges $(2, 4)$ and $(7, 8)$ in the right tree are not found in the left tree. One can also verify that the other five edge-induced partitions in both trees are identical. Hence, the RF distance between the left and right trees is 4.

Like the phylogenetic tree case, it is easy to see that the RF satisfies the non-negativity, symmetry and triangle inequality conditions.

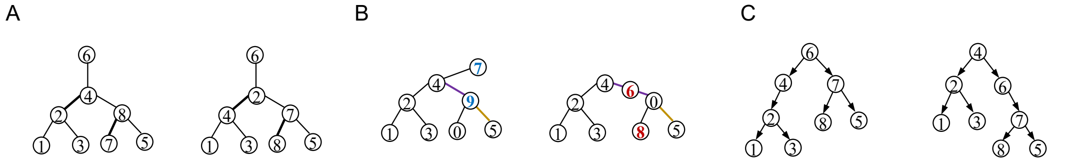


Figure 3 An illustration of the RF distance and the Bourque distance. **A.** Two unrooted 1-labeled trees. The RF distance between them is 4, as in the left tree, the edges $(2,4)$ and $(7,8)$ induces two partitions that are not found in the right tree and vice versa. **B.** The labels 0–5 are the labels appearing in the two trees. The Bourque distance between them is 9 (see the main text for details). **C.** The two labeled trees are rooted at different nodes. The RF distance between the left tree and the right tree is 2, as the partitions induced by $(6,4)$ of Tree A and $(4,6)$ of Tree B are different.

4 Generalizations of the RF distance for labeled trees

Let us consider labeled trees of different sizes or whose label sets are not the same (see the mutation trees studied in Section 7). The RF distance between any pair of such trees is simply equal to the total number of edges in the trees and thus fails to capture their dis-similarity. Here, we propose generalizations of the RF distance for measuring the dis-similarity of such trees better.

4.1 Bourque distances

For a labeled tree S, we use $\mathcal{L}(S)$ to denote the label set of S. Since each node of $V(S)$ is labeled with a non-empty subset of $\mathcal{L}(S)$, each edge $e = (u,v)$ induces the two-part partition $P(e) = \{L(u), L(v)\}$, where $L(u) = \cup_{x \in V(S): d(x,u) < d(x,v)} \ell(x)$ and $L(v) = \cup_{y \in V(S): d(y,v) < d(y,u)} \ell(y)$.

Let T be another labeled tree such that $C \triangleq \mathcal{L}(S) \cap \mathcal{L}(T) \neq \emptyset$. For $e' \in E(S)$ and $e'' \in E(T)$, we assume that the two-part partitions induced by e' and e'' are $P(e') = \{X, \mathcal{L}(S) \setminus X\}$ and $P(e'') = \{Y, \mathcal{L}(T) \setminus Y\}$, respectively, where $X \subset \mathcal{L}(S)$ and $Y \subset \mathcal{L}(T)$. $P(e')$ and $P(e'')$ are said to be *similar* if the following conditions are satisfied:

- $P(e') \neq P(e'')$;
- $X \cap C \neq \emptyset$ and $(\mathcal{L}(S) \setminus X) \cap C \neq \emptyset$;
- $\{X \cap C, (\mathcal{L}(S) \setminus X) \cap C\} = \{Y \cap C, (\mathcal{L}(T) \setminus Y) \cap C\}$.

We use $\sim$ to denote the similarity relationship of the edge-induced partitions of two trees. Note that the similarity relation is a many-to-many relation in the product space of edge-induced partitions $\mathcal{P}(\mathcal{S}) \times \mathcal{P}(T)$.

▶ **Definition 5.** *Let S and T be two labeled trees and let $\mathcal{P}$ be the set of two-part partitions of $\mathcal{L}(S) \cap \mathcal{L}(T)$. The Bourque metric $B(S,T)$ between S and T is defined as:*

$$B(S,T) \triangleq |\mathcal{P}(T) \Delta \mathcal{P}(S)| - \sum_{P \in \mathcal{P}} \min \left(|\{Q' \in \mathcal{P}(S) : Q' \sim P\}|, |\{Q'' \in \mathcal{P}(T) : Q'' \sim P\}| \right). \qquad (2)$$

The intuition behind this definition is that we “correct” the RF distance by those partitions, that would be shared between both trees when labels unique to either of the two trees were ignored. For example, in Fig. 3B, the labels $\{7,9\}$ that appear in the left tree are not found in the right tree, whereas the labels $\{6,8\}$ that appear in the right tree are not found in the left tree. Therefore, none of the seven edge-induced partitions in either tree is found in the other. This implies that the RF distance between the two trees is 14. Since the labels appearing in both trees are $\{1,2,3,4,5\}$, the edge $(4,9)$ (purple) of the left tree induces the same partition, $\{\{1,2,3,4\},\{0,5\}\}$ of $\{0,1,2,3,4,5\}$ as the edges $(4,6)$ and $(6,0)$ (purple) of the right tree. Furthermore, the edge $(1,2)$ (resp. $(2,3)$ and $(2,4)$) induces the same partition of $\{0,1,2,3,4,5\}$ in both trees; and the edge $(9,5)$ of the left tree induces the same partition of $\{1,2,3,4,5\}$ as the edge $(0,5)$ of the right tree. Therefore, the Bourque distance between both trees is $14-5=9$.

▶ **Proposition 6.** *Let S and T be two labeled trees with s and t nodes, respectively.*

(i) *If $\mathcal{L}(S)=\mathcal{L}(T)$, $2\times|s-t|\le B(S,T)=\mathrm{RF}(S,T)$.*

(ii) *If $\mathcal{L}(S)\neq\mathcal{L}(T)$, $\max(s,t)-1\le B(S,T)\le \mathrm{RF}(S,T)=s+t-2$.*

(iii) *If $\mathcal{L}(S)\cap\mathcal{L}(T)=\emptyset$, $B(S,T)=\mathrm{RF}(S,T)=s+t-2$.*

(iv) *The Bourque metric is a distance metric; in other words, it satisfies the non-negativity, symmetry and triangle inequality conditions.*

Proof. The full proof appears in the Appendix. ◀

▶ **Proposition 7.** *The Bourque distance between two labeled trees S and T can be computed in linear time $O(|\mathcal{L}(S)|+\mathcal{L}(T)|$.*

Proof. We assume node labels are integers (otherwise, we apply hashing to convert the labels into integers). By indexing labels with integers and fill a hash table, we can determine the set C of node labels that are in both trees. We then remove all labels that are not in C from the two trees, as well as nodes that lost all labels in the process, so that the resulting trees have only nodes with labels from C. Lastly, we apply the algorithm developed by Day [8] for the RF distance to compute the negative term of Eqn. (2) in linear time.

The first term of Eqn. (2) is the RF distance and can thus be found in linear time. ◀

4.2 High-order Bourque distances

Like the RF distance, the Bourque distance has the tendency to overpenalize certain labeling differences and can saturate quickly. Thus it is not refined enough for some applications. In this subsection, we will use the Bourque distances between local subtrees and a matching algorithm ([2, 26, 33]) to define new distance metrics. The new metrics will take more values than the basic version.

Let T be a labeled tree and $u\in V(T)$. For an integer $k>0$, the k-star subtree $N_k(u)$ centered at u is defined as the subtree induced by the vertex set $\{v\in V(T)\mid d(u,v)\le k\}$ in T. For any pair of labeled trees S and T of n and n' nodes, respectively, such that $n\ge n'$, define $\mathrm{BG}_k(S,T)$ as the weighted complete bipartite graph with two node parts $\{N_k(x)\ :\ x\in V(S)\}$ and $\{\emptyset_1,\cdots\emptyset_{n-n'}\}\cup\{N_k(y)\ :\ y\in V(T)\}$, where each $\emptyset_i$ is just the empty graph; the Bourque distance $B(N_k(x),N_k(y))$ is assigned to the edge $(N_k(x),N_k(y))$ as a weight for every $x\in V(S)$ and $y\in V(T)$ and $|N_k(x)|-1$ is assigned to the edge $(N_k(x),\emptyset_i)$ as a weight for any $\emptyset_i$. Although $N_k(x)$ can be identical for different nodes x, $\mathrm{BG}_k(S,T)$ always has $2n$ nodes.

▶ **Definition 8.** *Let S and T be two labeled trees. The k-Bourque distance $B_k(S,T)$ is defined as the minimum weight of a perfect matching in $\mathrm{BG}_k(S,T)$, $k\ge 1$.*

▶ **Proposition 9.** *The k-Bourque distances have the following properties:*

(1) *For any uniquely labeled trees S and T such that $|V(S)|=|V(T)|=n$, $B_k(S,T)=n\cdot B(S,T)$ for any $k\ge\max(\mathrm{diam}(S),\mathrm{diam}(T))$, where $\mathrm{diam}(X)$ is the diameter of X for $X=S,T$.*

(2) *$B_k(S,T)$ satisfies the non-negativity, symmetry and triangle inequality conditions for each $k\ge 1$.*

Proof. The full proof appears in the Appendix. ◀

▶ Remark. The run time of computing the k-Bourque distance for two labeled trees S and T with m and n nodes, respectively, is $O(\max(m, n)^3)$, as computing the Bourque distances between the k-star trees centered at tree nodes takes $O(\max(m, n)^2)$ in the worst case and computing the minimum weight perfect matching in $\mathrm{BG}_k(S, T)$ takes $O(\max(m, n)^3)$ time.

5 The Bourque distances for mutation trees

In this section, we will describe how to generalize the gNNI and Bourque distances to rooted labeled trees.

5.1 The gNNI

To transform a binary rooted phylogenetic tree into another in which the root is labeled differently, we add the so-called rotation operation that allows two nodes u and v that are connected by an edge to interchange not only one of their children but also their positions (right, Fig. 1B)[25]. A gNNI on a directed edge (a, b) of a rooted tree rewires some outgoing edges from a to b and vice versa and/or rewires the incoming edges to both a and b simultaneously (right, Figure 2B). More precisely, the gNNI is defined on rooted labeled trees as follows:

▶ **Definition 10.** *Let T be a rooted labeled tree and $e = (a, b) \in E(T)$ (where b is a child of a). An NNI operation on e transforms T by selecting a subset of edges $C_a = \{(a, x)\}$ that leave a, where $(a, b) \notin C_a$, and a subset of edges $C_b = \{(b, y)\}$ that leave b and then either (i) replacing each edge (a, x) of C_a with (b, x) and each edge (b, y) of C_b with (a, y) (left, Figure 2B) or (ii) rewiring the edges in C_a and C_b as in (i) as well as replacing the unique edge (z, a) that enters a and (a, b) with (z, b) and (b, a), respectively (right, Figure 2B).*

It is easy to see that for any pair of arbitrary labeled trees S and T, S can be transformed into T through a series of gNNIs as long as the labels appearing in the two trees are the same.

5.2 The RF and Bourque distances

In a rooted labeled tree, each directed edge also induces a 2-part partition on the label set. Therefore, the RF distance is well defined even for rooted trees that may not be uniquely labeled.

Let T be a rooted labeled tree. Recall that $\mathcal{L}(T)$ denotes the set of labels appearing in T. For a non-root node $u \in V(T)$, we use $L_T(u)$ to denote the set of the labels of u and its descendants. The unique edge entering u induces then an "ordered" two-part partition $(L_T(u), \mathcal{L}(T) \setminus L_T(u))$, which is an ordered pair of the two complementary subsets of $\mathcal{L}(T)$. Since the root of a rooted tree is a distinct node of the tree, we assume that the root is contained in the second part of an edge-induced partition. Hence, two edge-induced ordered partitions P' and P'' are *equal* if and only if the first part of P' is equal to the first component of P'' and the second part of P' is equal to the second component of P''. This is particularly useful when comparing two rooted trees with different roots. Let us define $\mathcal{OP}(T)$ to be the set of all edge-induced ordered partitions of T.

▶ **Definition 11.** *For two rooted labeled trees S and T, the RF distance $\mathrm{RF}(S, T)$ between S and T is defined as $|\mathcal{OP}(T) \,\Delta\, \mathcal{OP}(T)|$.*

For example, the two trees given in Figure 3C are obtained from rooting a unrooted labeled tree in different nodes. Only the partition induced by the edge $(6,4)$ of the left tree is not found in the right tree. Conversely, the partition induced by the edge $(4,6)$ in the right tree is not found in the left tree. Hence, the distance between these two trees is 2.

▶ **Proposition 12.** *Let S and T be two rooted labeled trees of equal size that have the same labels.*

(1) *Let $t \in V(T)$ such that it has the same label as the root r_S of S and let r_T be the root of T. We have that $RF(S,T) \geq 2d$, where d is the distance between r_T and t.*

(2) $\frac{1}{2}RF(S,T) \leq d_{\text{gnni}}(S,T) \leq RF(S,T)$.

Proof.

(1) Let the path between r_T and t be $r_T = t_0, t_1, t_2, \cdots, t_d = t$. All label sets $L_T(t_i)$ contain the label $\ell(r_S)$. However, only $L_T(t_0)$ is an element of $\{L_S(u) \mid u \in V(S)\}$. Furthermore, since both trees have the same number of nodes and edges, at least d subsets of $\{L_S(u) \mid u \in V(S)\}$ are not found in $\{L_T(v) \mid v \in V(T)\}$. Hence, $RF(S,T) \geq 2d$.

(2) The proof is similar to that of Proposition 4. ◀

Similarly, we can generate the similarity relationship of edge-induced partitions. For two non-root nodes $u \in V(S)$ and $v \in V(T)$, the ordered partitioned induced by the edges entering u and v are *similar* if and only if $(L_S(u), \mathcal{L}(S) \setminus L_S(u)) \neq (L_T(v), \mathcal{L}(T) \setminus L_T(v))$ but they are equal when restricted on $\mathcal{L}(S) \cap \mathcal{L}(T)$, denoted $(L_S(u), \mathcal{L}(S) \setminus L_S(u)) \sim (L_T(v), \mathcal{L}(T) \setminus L_T(v))$.

▶ **Definition 13.** *The Bourque distance $B(S,T)$ between two rooted labeled trees S and T is defined to be:*

$$|\mathcal{OP}(S)\Delta\mathcal{OP}(T)| - \sum_{P \in \mathcal{P}} \min(|\{(P' \in \mathcal{OP}(S) \ : \ P' \sim P\}|, |\{(P'' \in \mathcal{OP}(T) \ : \ P'' \sim P\}|). \tag{3}$$

▶ **Proposition 14.** *The Bourque distance between two mutation trees S and T can be computed in linear time $O(|\mathcal{L}(S)| + \mathcal{L}(T)|$.*

The proof is analogous to Proposition 7, but we only count partitions that have no root label(s) in their first part.

5.3 High-order Bourque distances

Let S and T be two rooted labeled trees and $k \geq 1$. Set $L_T^{(k)}(u) = \{\ell(v) \in L_T(u) \mid d_T(u,v) \leq k\}$. By Proposition 12.1, we naturally define the k-Bourque distance $B_k(S,T)$ to be the minimum weight of a perfect matching in the complete bipartite graph $G_k(S,T)$. Here, if we assume $|V(S)| \leq |V(T)|$, $G_k(S,T)$ has the vertex set $\{\emptyset_i, L^{(k)}(s) \mid 1 \leq i \leq |V(T)| - |V(S)|;\ s \in S\} \cup \{L^{(k)}(t) \mid t \in T\}$ and the edge set $\{\emptyset_i, L^{(k)}(s) \mid s \in S\} \times \{L^{(k)}(t) \mid t \in T\}$, together with the weight function $B(x,y)$, where each $\emptyset_i$ is a copy of the empty graph.

6 Comparison of eight distance measures on rooted labeled trees

In this section, we compare the *Bourque distance* (BD) against the *1-Bourque distance* (1-BD), the *2-Bourque distance* (2-BD) and five previously published distance measures: *Common Ancestor Set* (CASet) [18], *Distinctly Inherited Set Comparison* (DISC) [18], an *Ancestor Difference measure* (AD) [18], a Triplet-based Distance (TD) [5] and the *Multi-Labeled Tree Dissimilarity* (MLTD) measure [20]. A detailed description of these measures is given in the Appendix. The gNNI distance is not included in this comparison, as there is no known method for its efficient computation.

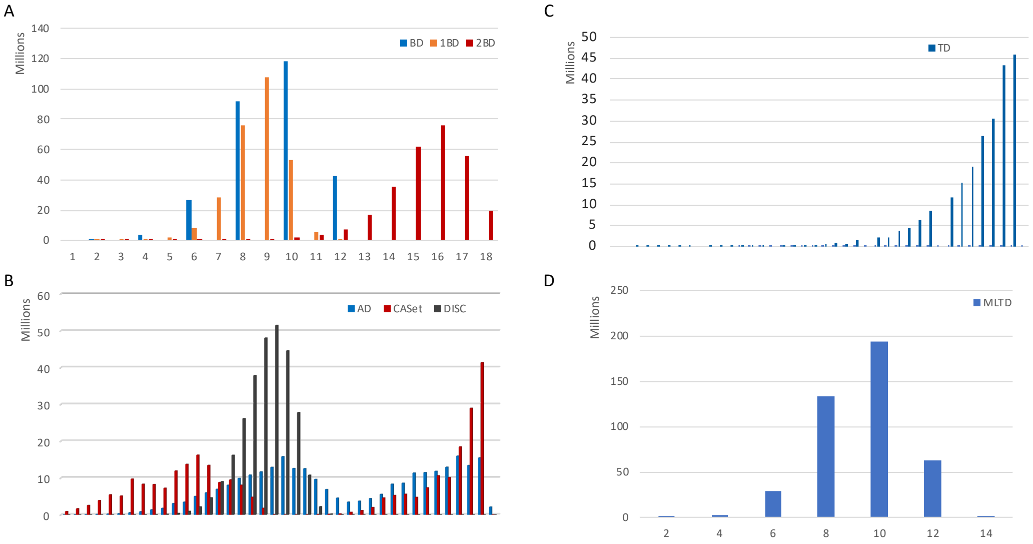


Figure 4 The frequency distribution of pairwise distances for the BD, 1-BD and 2-BD metrics (A), the AD, CASet and DISC measures (B), the TD measure (C) and the MLTD measure (D) in the space of rooted 1-labeled trees with 7 nodes. BD: Bourque distance; AD: Ancestor distance; CASet: Common Ancestor Set distance; DISC: Distinctly Inherited Set; TD: Triplet-based distance. MLTD: Multi-label tree distance.

6.1 Frequency distribution of the pair-wise distances in different metrics

There are $16,807$ unrooted and $7 \times 16,807$ rooted 1-labeled trees with seven nodes. Let R denote the set of such trees and let R_i denote the set of those rooted at Node i, where $1 \leq i \leq 7$. Let d be a distance function of rooted labeled trees. Clearly, for any i, $\{d(x, y) : x \in R_i, y \in R_i\} = \{d(x, y) : x \in R_1, y \in R_1\}$; for different nodes i and j, $\{d(x, y) : x \in R_i, y \in R_j\} = \{d(x, y) : x \in R_1, y \in R_2\}$. Therefore, we computed the pairwise Bourque distance (DB), 1-BD and 2-BD metrics between any $x \in R_1$ and any $y \in R_1 \cup R_2$ such that $x \neq y$. The frequency distributions of the three metrics are given in Fig. 4A, showing a Poisson distribution as the RF in the unrooted case [38].

The pairwise distances of AD, CASet, DISC and TD range from 0 to 1. We computed all the pair-wise distances for all possible pairs of distinct $x \in R_1$ and $y \in R_1 \cup R_2$. Because of the huge number of pair-wise distances, we binned them into 40 intervals $\left(\frac{i}{40}, \frac{i+1}{40}\right)$, $0 \leq i \leq 39$. The histograms for the frequency distributions of the pairwise distance values for the three measures are given in Fig. 4B. The AD and CASet measures have a similar distribution (blue and red in Fig. 4B), each having two peaks. The pairwise distances between trees rooted at the same node form the first peak, whereas the pairwise distances between trees rooted at different nodes form the second peak. These facts show that AD and CASet are sensitive to the root node. The frequency distribution (black) of the DISC measure appears to be again a kind of Poisson distribution. Whether the pairwise distances of the DISC, 1-BD and 2-BD between all 1-labeled trees with a given number of nodes follow a Poisson distribution or not needs further mathematical investigation. The bottom line is that the DISC measure and the Bourque metrics have different distributions of pairwise distances from the AD and DISC measures.

The frequency distribution of the TD is clearly different from the AD, CASet and DISC (Fig. 4C). More than 60% of the pairwise distances are greater than 0.9. For the discrete MLTD measure, we observe a Poisson-like distribution similar to the BD metric.

Lastly, for each of the AD, CASet, DISC, TD and MLTD measures, there are many pairs of trees with the same distance value, that have distinct distances in the BD metric. Figure S1 give an example for each.

6.2 Pairwise distances between random trees

We compared the BD, 1-BD, 2-BD, AD, CASet, DISC, TD and MLTD measures on rooted 1-labeled, 30-node trees that were randomly generated as follows. The tree generator first generated a random unrooted 1-labeled 30-node tree T_0 and then generated 20,000 random unrooted 1-labeled, 30-node trees in 400 iterations. In the i-th iteration, a tree generated in the $(i-1)$-th iteration was randomly selected. Next, five random trees were generated from the selected tree by applying a random NNI on an edge $e = (u, v)$ that was randomly generated, where u was an internal node. Here, a NNI just switched one subtree from the u side to v and one subtree from the v side to u if v was not a leaf and just moved a subtree from u to v if v was a leaf.

We computed the eight different distance values between T_0 rooted at Node 1 and the 20,000 trees rooted at Node 1, which are summarized in Fig. 5. This produced two interesting findings. First, the BD distances from T_0 to the random trees range from 0 to 58; the BD, 1-BD and 2-BD correlate well with each other, particularly when the Bourque distances ranged from 0 to 35. However, the distances between a pair of trees can be very different in the three metrics. For example, there are 3367 random trees that are 46 BD away from T_0. The 1-BDs between T_0 and the trees are from 32 to 45 (top left panel, Figure 5).

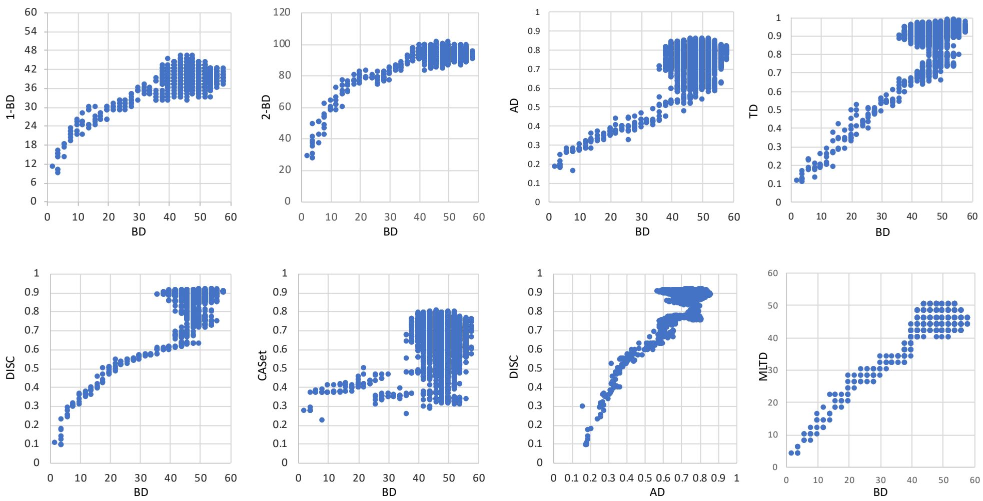


Figure 5 The scatter plots of the Bourque vs the other distance measures between a rooted 1-labeled tree and 20,000 random trees of 30 nodes. BD: Bourque distance; AD: Ancestor distance; CASet: Common Ancestor Set distance; DISC: Distinctly Inherited Set; MLTD: Multi-label tree distance; TD: Triplet-based distance.

Second, AD, DISC, MLTD and TD correlated with BD (and hence 1-BD and 2-BD) surprisingly well with Pearson correlation coefficients (PCC) from 0.38 to 0.543 even though they are defined differently. However, CASet and BD poorly correlated (middle panel, second row) with PCC 0.112.

7 Applications to mutation trees

7.1 The distances between three leukemia mutation trees

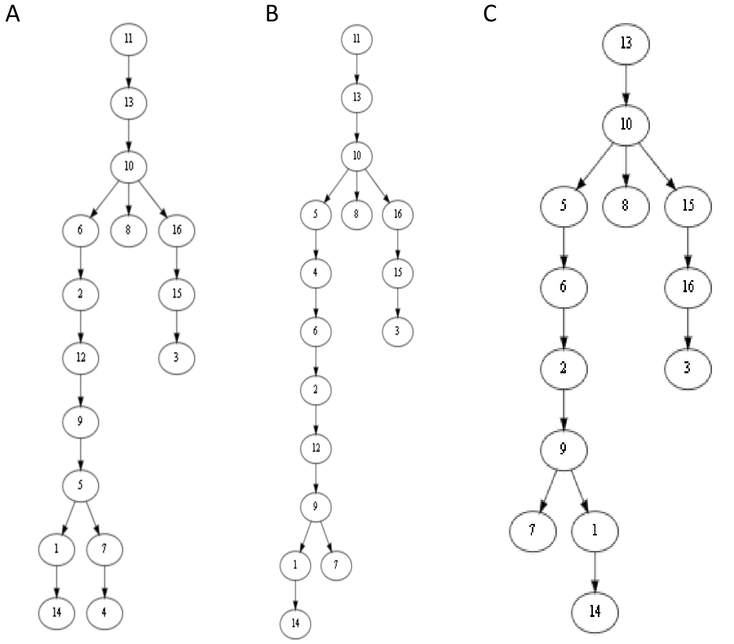


Figure 6 The mutation trees inferred by SCITE [19] (A), B-SCITE [28] (B) and PhISCS [30] (C) from single-cell sequencing data or with the bulk sequencing data for Patient 2 with childhood acute lymphoblastic leukemia that was reported in [16]. The mutation trees contain 16 mutated genes: *ATRNL1* (1), *BDNF_AS* (2), *BRD7P3* (3), *CMTM8* (4), *FAM105A* (5), *FGD4* (6), *INHA* (7), *LINXC00052* (8), *PCDH7* (9), *PLEC* (10), *RIMS2* (11), *RRP8* (12), *SIGLEC10* (13), *TRRAP* (14), *XPO7* (15), *ZC3H3* (16).

Single-cell sequencing data are prone to errors. Mutation trees inferred by different methods from the single-cell sequencing data of a patient are often different in both topology and labels of mutated genes. Fig. 6 shows mutation trees inferred by SCITE [19], B-SCITE [28] and PhISCS [30] for Patient 2, who had childhood acute lymphoblastic leukemia from [16]. Both the SCITE and B-SCITE trees (i.e. Tree A and Tree B) contain 16 mutations, whereas the PhISCS tree (i.e. Tree C) contains just 13 of the 16 mutations.

The pairwise distances between the trees were calculated using the eight distance measures (Table 1). Tree A and Tree B contain the same mutations. The difference between them is mainly the positions of Mutation 4 and Mutation 5 in the long chain on the left. The pairwise distance between them has the smallest value among the three trees for each of the eight measures. Tree B and Tree C have the same topology and are different only in that Mutations 4, 11 and 12 are missing in the latter. For each measure, the distance between Tree B and Tree C is smaller than or nearly equal to the distance between Tree A and Tree

Table 1 Pairwise distances between three mutation trees A, B, and C in Fig. 6 according to different metrics. The union extension of CASet and DISC were used to measure the difference between Tree A (or Tree B) and Tree C [10].

	A & B	A & C	B & C
BD	12	14	14
1-BD	9	26	23
2-BD	28	40	33
MLTD	4	7	5
CASet	0.1079	0.5495	0.5302
DISC	0.2394	0.4135	0.3468
AD	0.1699	0.5499	0.5276
TD	0.3607	0.6393	0.5821

C, consistent with intuition.

7.2 Distances between four simulated mutation trees

Figure 7 presents four simulated mutation trees downloaded from the OncoLib database for which the CASet and DISC disagreed significantly [10]. The pairwise distances between the four trees are given in Table 2. Note that the CASet and DISC distances between T_5 and T_{20} and between T_{14} and T_{26} are different from those reported in [10]. This is because a mutation appearing in a tree node is not an ancestor of another mutation in the same node in our distance calculation. Regardless of the differences between the definitions, our distance computing also shows the disagreement between the CASet and DISC distances. For example, the CASet distance between T_5 and T_{20} is four times as large as the CASet distance between T_{14} and T_{26}, whereas the DISC distance between the former is smaller than the DISC distance between the latter. This disagreement is also observed on the tree pairs $\{T_5, T_{14}\}$ and $\{T_{20}, T_{26}\}$.

Since these four different trees have only one internal edge, the Bourque distance between any two of them is 2. The pairwise 1-BD distances are not much different. However, their differences are reflected in the pairwise 2-BD distances.

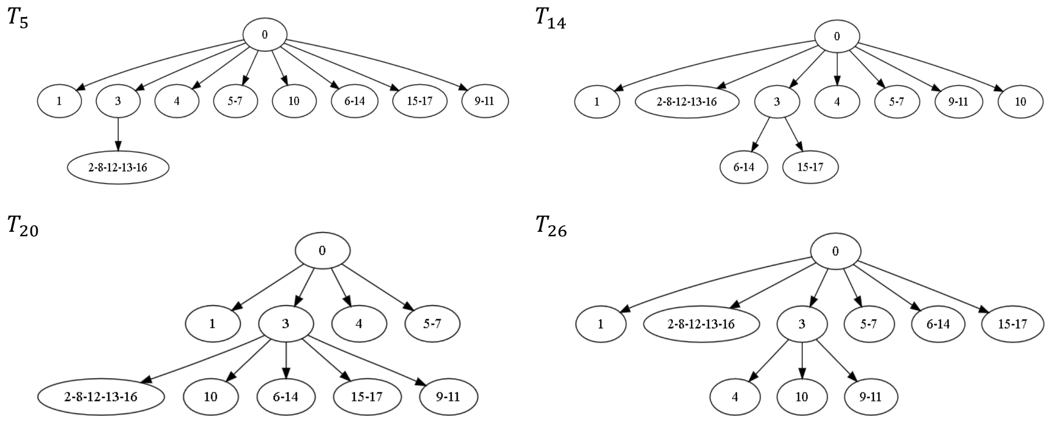


Figure 7 Four simulated mutation trees T_5, T_{14}, T_{20} and T_{26} from the OncoLib database [12].

Table 2 Pairwise distances between trees in Fig. 7 according to the eight distance measures.

	T_5 & T_{14}	T_5 & T_{20}	T_5 & T_{26}	T_{14} & T_{20}	T_{14} & T_{26}	T_{20} & T_{26}
BD	2	2	2	2	2	2
1-BD	11	12	12	12	13	12
2-BD	4	6	5	7	7	8
MLTD	6	10	6	14	10	12
CASet	0.0523	0.1830	0.0523	0.1961	0.0392	0.2157
DISC	0.3807	0.2402	0.3807	0.2483	0.3529	0.3039
AD	0.2500	0.1944	0.2500	0.2222	0.2222	0.2778
TD	0.1961	0.4363	0.2120	0.4669	0.2659	0.4951

8 Conclusions

We have introduced the Bourque and k-Bourque metrics for both unrooted labeled trees and mutation trees. These distances are the generalizations of the RF distance. We demonstrate, through a simulation, that they correlate with the CASet, DISC and AD distance measures for similar trees, but have different distributions of pairwise distances on between all 1-labeled trees with a fixed number of nodes. The advantages of the Bourque metric over CASet and DISC include that it satisfies the triangle inequality and it is computable in linear time. The k-Bourque metrics refine the Bourque metric.

Another contribution is a novel connection between the RF and gNNI metrics on labeled trees. A few theoretical questions arise from this connection between the RF and gNNI and related contributions. Is finding the gNNI distance for labeled trees NP-complete? What is the maximum value of the NNI distance between two binary 1-labeled trees? Can the RF distance be used to define a polynomial time algorithm with approximation ratio < 2 for the gNNi distance?

General mathematical questions also arise from the development of new metrics for comparisons of mutation trees. One is investigating mathematical relationships between the proposed metrics. Another is determining the distributions of pairwise distances between all the 1-labeled trees of the same size. For example, is the distribution Poisson for the Bourque metrics?

Finally, further generalisations of the Bourque distance will be interesting to study in the future, in particular for mutation trees where labels may occur multiple times in different nodes [5]. The motivation for this generalisation comes from the observation that in tumours the same mutations can happen independently in multiple subclones and can also be lost again over time [22].

References

1 Giulia Bernardini, Paola Bonizzoni, and Paweł Gawrychowski. On two measures of distance between fully-labelled trees. *arXiv preprint arXiv:2002.05600*, 2020.

2 Damian Bogdanowicz and Krzysztof Giaro. Matching split distance for unrooted binary phylogenetic trees. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(1):150–160, 2011.

3 Michel Bourque. *Arbes de Steiner et réseaux dont certains sommets sont à localisation variable.* PhD thesis, Thèse (Ph. D.: Informatique)–Université de Montréal, 1978.

4 Samuel Briand, Christophe Dessimoz, Nadia El-Mabrouk, Manuel Lafond, and Gabriela Lobinska. A generalized robinson-foulds distance for labeled trees. In *Proceedings of APBC*, 2020.

5 Simone Ciccolella, Giulia Bernardini, Luca Denti, Paol Bonizzoni, Marco Previtali, and Gianluca Della Vedova. Triplet-based similarity score for fully multi-labeled trees with poly-occurring labels. *bioRxiv*, 2020.

6 Simone Ciccolella, Mauricio Soto Gomez, Murray Patterson, Gianluca Della Vedova, Iman Hajirasouliha, and Paola Bonizzoni. Inferring cancer progression from single cell sequencing while allowing loss of mutations. *bioRxiv*, page 268243, 2018.

7 Douglas E Critchlow, Dennis K Pearl, and Chunlin Qian. The triples distance for rooted bifurcating phylogenetic trees. *Systematic Biology*, 45(3):323–334, 1996.

8 William HE Day. Optimal algorithms for comparing trees with labeled leaves. *Journal of Classification*, 2(1):7–28, 1985.

9 Amit G Deshwar, Shankar Vembu, Christina K Yung, Gun Ho Jang, Lincoln Stein, and Quaid Morris. Phylowgs: reconstructing subclonal composition and evolution from whole-genome sequencing of tumors. *Genome Biology*, 16(1):1–20, 2015.

10 Zach DiNardo, Kiran Tomlinson, Anna Ritz, and Layla Oesper. Distance measures for tumor evolutionary trees. *Bioinformatics*, 36(7):2090–2097, 2020.

11 Jesse Eaton, Jingyi Wang, and Russell Schwartz. Deconvolution and phylogeny inference of structural variations in tumor genomic samples. *Bioinformatics*, 34(13):i357–i365, 2018.

12 Mohammed El-Kebir. Oncolib: Library for tumor heterogeneity. *GitHub repository*, 2018.

13 Mohammed El-Kebir. Sphyr: tumor phylogeny estimation from single-cell sequencing data under loss and error. *Bioinformatics*, 34(17):i671–i679, 2018.

14 Mohammed El-Kebir, Layla Oesper, Hannah Acheson-Field, and Benjamin J Raphael. Reconstruction of clonal trees and tumor composition from multi-sample sequencing data. *Bioinformatics*, 31(12):i62–i70, 2015.

15 Joseph Felsenstein and Joseph Felenstein. *Inferring phylogenies*, volume 2. Sinauer associates Sunderland, MA, 2004.

16 Charles Gawad, Winston Koh, and Stephen R Quake. Dissecting the clonal origins of childhood acute lymphoblastic leukemia by single-cell genomics. *Proceedings of the National Academy of Sciences*, 111(50):17947–17952, 2014.

17 Morris Goodman, John Czelusniak, G William Moore, Alejo E Romero-Herrera, and Genji Matsuda. Fitting the gene lineage into its species lineage, a parsimony strategy illustrated by cladograms constructed from globin sequences. *Systematic Biology*, 28(2):132–163, 1979.

18 Kiya Govek, Camden Sikes, and Layla Oesper. A consensus approach to infer tumor evolutionary histories. In *Proceedings of the 2018 Acm international conference on bioinformatics, computational biology, and health informatics*, pages 63–72, 2018.

19 Katharina Jahn, Jack Kuipers, and Niko Beerenwinkel. Tree inference for single-cell data. *Genome biology*, 17(1):1–17, 2016.

20 Nikolai Karpov, Salem Malikic, Md Khaledur Rahman, and S Cenk Sahinalp. A multi-labeled tree dissimilarity measure for comparing "clonal trees" of tumor progression. *Algorithms for Molecular Biology*, 14(1):17, 2019.

21 Michelle Kendall and Caroline Colijn. Mapping phylogenetic trees to reveal distinct patterns of evolution. *Molecular Biology and Evolution*, 33(10):2735–2743, 2016.

22 Jack Kuipers, Katharina Jahn, Benjamin J Raphael, and Niko Beerenwinkel. Single-cell sequencing data reveal widespread recurrence and loss of mutational hits in the life histories of tumors. *Genome Research*, 27(11):1885–1894, 2017.

23 Shu-Yun Le, Ruth Nussinov, and Jacob V Maizel. Tree graphs of rna secondary structures and their comparisons. *Computers and Biomedical Research*, 22(5):461–473, 1989.

24 Ming Li, John Tromp, and Louxin Zhang. On the nearest neighbour interchange distance between evolutionary trees. *Journal of Theoretical Biology*, 182(4):463–467, 1996.

25 Ming Li and Louxin Zhang. Twist–rotation transformations of binary trees and arithmetic expressions. *Journal of Algorithms*, 32(2):155–166, 1999.

26 Yu Lin, Vaibhav Rajan, and Bernard ME Moret. A metric for phylogenetic trees based on matching. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(4):1014–1022, 2011.

27 Wayne P Maddison. Gene trees in species trees. *Systematic Biology*, 46(3):523–536, 1997.

28 Salem Malikic, Katharina Jahn, Jack Kuipers, S Cenk Sahinalp, and Niko Beerenwinkel. Integrative inference of subclonal tumour evolution from single-cell and bulk sequencing data. *Nature Communications*, 10(1):1–12, 2019.

29 Salem Malikic, Andrew W McPherson, Nilgun Donmez, and Cenk S Sahinalp. Clonality inference in multiple tumor samples using phylogeny. *Bioinformatics*, 31(9):1349–1356, 2015.

30 Salem Malikic, Farid Rashidi Mehrabadi, Simone Ciccolella, Md Khaledur Rahman, Camir Ricketts, Ehsan Haghshenas, Daniel Seidman, Faraz Hach, Iman Hajirasouliha, and S Cenk Sahinalp. Phiscs: a combinatorial approach for subperfect tumor phylogeny reconstruction via integrative use of single-cell and bulk sequencing data. *Genome Research*, 29(11):1860–1877, 2019.

31 G William Moore, M Goodman, and J Barnabas. An iterative approach from the standpoint of the additive hypothesis to the dendrogram problem posed by molecular data sets. *Journal of Theoretical Biology*, 38(3):423–457, 1973.

32 Peter C Nowell. The clonal evolution of tumor cell populations. *Science*, 194(4260):23–28, 1976.

33 Tom MW Nye, Pietro Lio, and Walter R Gilks. A novel algorithm and web-based tool for comparing two alternative phylogenetic trees. *Bioinformatics*, 22(1):117–119, 2006.

34 Victoria Popic, Raheleh Salari, Iman Hajirasouliha, Dorna Kashef-Haghighi, Robert B West, and Serafim Batzoglou. Fast and scalable inference of multi-sample cancer lineages. *Genome Biology*, 16(1):91, 2015.

35 David F Robinson. Comparison of labeled trees with valency three. *Journal of Combinatorial Theory, Series B*, 11(2):105–119, 1971.

36 David F Robinson and Leslie R Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53(1-2):131–147, 1981.

37 Bruce A Shapiro and Kaizhong Zhang. Comparing multiple rna secondary structures using tree comparisons. *Bioinformatics*, 6(4):309–318, 1990.

38 Mike Steel and David Penny. Distributions of tree comparison metrics—some new results. *Systematic Biology*, 42:126–141, 1993.

39 Y Tateno, M Nei, and Tajima F. Accuracy of estimated phylogenetic trees from molecular data. *Journal of Molecular Evolution*, 18:387–404, 1982.

40 Gabriel Valiente. *Algorithms on Trees and Graphs*, volume 2. Springer, New York, USA, 2013.

41 WT Williams and HT Clifford. On the comparison of two classifications of the same set of elements. *Taxon*, 20:519–522, 1971.

42 H Zafar, N Navin, K Chen, and L Nakhleh. Siclonefit: Bayesian inference of population structure, genotype, and phylogeny of tumor clones from single-cell genome sequencing data. *Genome Research*, 29:1847–1859, 2019.

43 K Zhang and D Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18:1245–1262, 1989.

A Appendix: Proofs of Propositions 4 and 6

A.1 Proposition 4

Proposition 4. Let S and T be two labeled trees with s and t nodes, respectively.

(i) If $\mathcal{L}(S) = \mathcal{L}(T)$, $2 \times |s - t| \leq B(S,T) = \mathrm{RF}(S,T)$.

(ii) If $\mathcal{L}(S) \neq \mathcal{L}(T)$, $\max(s,t) - 1 \leq B(S,T) \leq \mathrm{RF}(S,T) = s + t - 2$.

(iii) If $\mathcal{L}(S) \cap \mathcal{L}(T) = \emptyset$, $B(S,T) = \mathrm{RF}(S,T) = s + t - 2$.

(iv) The Bourque metric is a distance metric; in other words, it satisfies the non-negativity, symmetry and the triangle inequality conditions.

Proof.

(i) Since the second term of (2) is non-positive, $B(S,T) \leq |\mathcal{P}(S)\Delta\mathcal{P}(T)| = \mathrm{RF}(S,T)$. Without loss of generality, we may assume $s \geq t$. By the definition of the similarity relation, $\mathcal{L}(S) = \mathcal{L}(T)$ implies that $\{(P,Q) \in \mathcal{P}(S) \times \mathcal{P}(T) \ : \ P \sim Q\} = \emptyset$ and thus $B(S,T) = RF(S,T) = 2|\mathcal{P}(S) \setminus \mathcal{P}(T)| \geq 2(s-t)$, proving the inequality.

(ii) If $\mathcal{L}(S) \neq \mathcal{L}(T)$, $|\mathcal{P}(T)\Delta\mathcal{P}(S)| = |\mathcal{P}(T)| + |\mathcal{P}(S)| = s + t - 2$. Moreover, we have:

$$\begin{aligned}
&\sum_{P\in\mathcal{P}} \min\left(|\{Q' \in \mathcal{P}(S) : Q' \sim P\}|, |\{Q'' \in \mathcal{P}(T) : Q'' \sim P\}|\right) \\
\leq\ & \min\left(\sum_{P\in\mathcal{P}} |\{Q' \in \mathcal{P}(S) : Q' \sim P\}|, \sum_{P\in\mathcal{P}} |\{Q'' \in \mathcal{P}(T) : Q'' \sim P\}|\right) \\
\leq\ & \min(|\mathcal{P}(T)|, |\mathcal{P}(T)|) = \min(s,t) - 1
\end{aligned}$$

and:

$$B(S,T) = s + t - 2 - (\min(s,t) - 1) = \max(s,t) - 1.$$

(iii) If $\mathcal{L}(S) = \mathcal{L}(T)$, the first term becomes $|\mathcal{P}(S)| + |\mathcal{P}(T)|$, which is $s + t - 2$; and the second term is zero. Therefore, the fact is true.

(iv) The non-negativity follows from (i) and (ii). The symmetric property of the Bourque metric follows from the definition of the Bourque distance. The triangle inequality is proved as follows.

Let T_1, T_2 and T_3 be three labeled trees. We consider the following three cases to prove $B(T_1,T_2) \leq B(T_1,T_3) + B(T_3,T_2)$.

Case 1. $\mathcal{L}(T_1) = \mathcal{L}(T_3) = \mathcal{L}(T_2)$. In this case, $B(T_i,T_j) = \mathrm{RF}(T_i,T_j)$. The triangle inequality for these three trees follows from the fact that the RF distance satisfies the triangle inequality.

Case 2. $\mathcal{L}(T_1) \neq \mathcal{L}(T_3)$ and $\mathcal{L}(T_3) \neq \mathcal{L}(T_2)$.

We have $B(T_1,T_2) \leq |\mathcal{P}(T_1)\Delta\mathcal{P}(T_2)| = |\mathcal{P}(T_1) \setminus \mathcal{P}(T_2)| + |\mathcal{P}(T_2) \setminus \mathcal{P}(T_1)|$.

On the other hand, by (ii), $\mathcal{L}(T_i) \neq \mathcal{L}(T_3)$ implies that $B(T_i,T_3) \geq \max\left(|\mathcal{P}(T_i)|, |\mathcal{P}(T_3)|\right)$ for $i = 1,2$. Therefore,

$$\begin{aligned}
B(T_1,T_3) + B(T_1,T_3) &\geq \max\left(|\mathcal{P}(T_1)|, |\mathcal{P}(T_3)|\right) + \max\left(|\mathcal{P}(T_2)|, |\mathcal{P}(T_3)|\right) \\
&\geq |\mathcal{P}(T_1)| + |\mathcal{P}(T_2)| \geq B(T_1,T_2).
\end{aligned}$$

Case 3. $\mathcal{L}(T_1) = \mathcal{L}(T_3) \neq \mathcal{L}(T_2)$ or $\mathcal{L}(T_1) \neq \mathcal{L}(T_3) = \mathcal{L}(T_2)$.

Note that the two conditions are symmetric. Hence, we just need to prove that the triangle inequality holds if the first condition is satisfied.

Let $\mathcal{P}$ be the set of 2-part partitions of $\mathcal{L}(T_1) \cap \mathcal{L}(T_2)$. Since $\mathcal{L}(T_1) = \mathcal{L}(T_3)$, $B(T_1,T_3) = |\mathcal{P}(T_1)\Delta\mathcal{P}(T_3)| = |\mathcal{P}(T_1)| + |\mathcal{P}(T_3)| - 2|\mathcal{P}(T_3) \cap \mathcal{P}(T_1)|$. Since $\mathcal{L}(T_1) \neq \mathcal{L}(T_2)$,

$$\begin{aligned}
&B(T_1,T_2) \\
=\ & |\mathcal{P}(T_1)| + |\mathcal{P}(T_2)| - \sum_{P\in\mathcal{P}} \min\left(|\{Q' \in \mathcal{P}(T_1) : Q' \sim P\}|, |\{Q'' \in \mathcal{P}(T_2) : Q'' \sim P\}|\right). \qquad (A.1)
\end{aligned}$$

Similarly, since $\mathcal{L}(T_2) \neq \mathcal{L}(T_3)$,

$$\begin{aligned} & B(T_1, T_3) + B(T_3, T_2) \\ = \; & |\mathcal{P}(T_1)| + 2|\mathcal{P}(T_3) \setminus \mathcal{P}(T_1)| + |\mathcal{P}(T_2)| \\ & - \sum_{P \in \mathcal{P}} \min\left(|\{Q' \in \mathcal{P}(T_3) : Q' \sim P\}|, |\{Q'' \in \mathcal{P}(T_2) : Q'' \sim P\}|\right). \end{aligned} \tag{A.2}$$

Since

$$\begin{aligned} & \sum_{P \in \mathcal{P}} \min\left(|\{Q' \in \mathcal{P}(T_3) : Q' \sim P\}|, |\{Q'' \in \mathcal{P}(T_2) : Q'' \sim P\}|\right) \\ \leq \; & \sum_{P \in \mathcal{P}} \min\left(|\{Q' \in \mathcal{P}(T_3) \setminus \mathcal{P}(T_1) : Q' \sim P\}|, |\{Q'' \in \mathcal{P}(T_2) : Q'' \sim P\}|\right) \\ & + \sum_{P \in \mathcal{P}} \min\left(|\{Q' \in \mathcal{P}(T_3) \cap \mathcal{P}(T_1) : Q' \sim P\}|, |\{Q'' \in \mathcal{P}(T_2) : Q'' \sim P\}|\right) \\ \leq \; & \sum_{P \in \mathcal{P}} |\{Q' \in \mathcal{P}(T_3) \setminus \mathcal{P}(T_1) : Q' \sim P\}| \\ & + \sum_{P \in \mathcal{P}} \min\left(|\{Q' \in \mathcal{P}(T_3) \cap \mathcal{P}(T_1) : Q' \sim P\}|, |\{Q'' \in \mathcal{P}(T_2) : Q'' \sim P\}|\right) \\ \leq \; & |\mathcal{P}(T_3) \setminus \mathcal{P}(T_1)| + \sum_{P \in \mathcal{P}} \min\left(|\{Q' \in \mathcal{P}(T_1) : Q' \sim P\}|, |\{Q'' \in \mathcal{P}(T_2) : Q'' \sim P\}|\right), \end{aligned}$$

by Eqn. (A.1) and (A.2),

$$\begin{aligned} & B(T_1, T_3) + B(T_3, T_2) \\ = \; & |\mathcal{P}(T_1)| + 2|\mathcal{P}(T_3) \setminus \mathcal{P}(T_1)| + |\mathcal{P}(T_2)| \\ & - \sum_{P \in \mathcal{P}} \min\left(|\{Q' \in \mathcal{P}(T_3) : Q' \sim P\}|, |\{Q'' \in \mathcal{P}(T_2) : Q'' \sim P\}|\right) \\ \geq \; & |\mathcal{P}(T_1)| + |\mathcal{P}(T_2)| - \sum_{P \in \mathcal{P}} \min\left(|\{Q' \in \mathcal{P}(T_1) : Q' \sim P\}|, |\{Q'' \in \mathcal{P}(T_2) : Q'' \sim P\}|\right) \\ = \; & B(T_1, T_2). \end{aligned}$$

The triangle inequality is proved. ◀

A2. Proposition 6

Proposition 6. The k-Bourque distances have the following properties:

(1) For any uniquely labeled trees S and T such that $|V(S)| = |V(T)| = n$, $B_k(S, T) = n \cdot B(S, T)$ for any $k \geq \max(\text{diam}(S), \text{diam}(T))$, where $\text{diam}(X)$ is the diameter of X for $X = S, T$.

(2) $B_k(S, T)$ satisfies the non-negativity, symmetry and triangle inequality conditions for each $k \geq 1$.

Proof.

(1) If $k \geq \max(\text{diam}(S), \text{diam}(T))$, $N_k(u) = S$ for any $u \in V(S)$ and $N_k(v) = T$ for any $v \in V(T)$. This implies that every edge has the same weight $B(S, T)$ and every perfect matching has a weight of $n \times B(S, T)$ in the graph $\text{BG}_k(S, T)$.

(2) Obviously, $B_k(S, T)$ has the non-negativity and symmetry properties for each k. Let S, T and W be three labeled trees. We assume that $s = |V(S)| \geq |V(T)| = t$ and consider three cases to prove that $B_k(S, T) \leq B_k(S, W) + B_k(W, T)$ for $k \geq 1$.

Case 1. $w = |V(W)| \geq s$. Let us assume that:

$$f : \{N_k(v),\ \emptyset_i : v \in V(S), 1 \leq i \leq w - s\} \to \{N_k(v) : v \in V(W)\}$$

is a 1-to-1 function such that $\{(N_k(v), f(N_k(v))), (\emptyset_j, f(\emptyset_i) \ : \ v \in V(S), 1 \leq i \leq w - s\}$ is the minimum weight perfect matching in $\mathrm{BG}_k(S, W)$. Let us also assume that:

$$g : \{N_k(v) : v \in V(W)\} \to \{N_k(v),\ \emptyset_i \ : \ v \in V(T), 1 \leq i \leq w - t\}$$

is a 1-to-1 function such that $\{(N_k(v), g(N_k(v))) \ : \ v \in V(W)\}$ is the minimum weight perfect matching in $\mathrm{BG}_k(W, T)$.
We now define the following:

$$W_{00} = \{v \in V(W) \ : \ N_k(v) = f(\emptyset) \ \&\ g(N_k(v)) = \emptyset\},$$

$$W_{01} = \{v \in V(W) \ : \ N_k(v) = f(\emptyset) \ \&\ g(N_k(v)) \neq \emptyset\},$$

$$W_{10} = \{v \in V(W) \ : \ N_k(v) \neq f(\emptyset) \ \&\ g(N_k(v)) = \emptyset\},$$

$$W_{11} = \{v \in V(W) \ : \ N_k(v) \neq f(\emptyset) \ \&\ g(N_k(v)) \neq \emptyset\}.$$

Clearly, $|W_{10}| + W_{11}| = s$, $|W_{01}| + W_{11}| = t$ and thus $|W_{10}| - |W_{01}| = s - t$.
Let $W_{10} = \{a_1, a_2, \cdots, a_{k'}\}$ and $W_{01} = \{b_1, \cdots, b_k\}$, where $k' = k + s - t$. We then have:

$$\begin{aligned} \{(f^{-1}(N_k(v)), g(N_k(v))) \ : \ v \in W_{11}\} \ &\cup \ \{(f^{-1}(N_k(a_i)), g(N_k(b_i))) \ : \ 1 \leq i \leq k\} \\ &\cup \ \{(f^{-1}(N_k(a_j)), \emptyset) \ : \ k < j \leq k'\} \end{aligned}$$

is a perfect matching in $\mathrm{BG}_k(S, T)$ and its weight is:

$$\begin{aligned} C \ = \ & \sum_{v \in W_{11}} B\left(f^{-1}(N_k(v)), g(N_k(v))\right) + \sum_{1 \leq i \leq k} B\left(f^{-1}(N_k(a_i)), g(N_k(b_i))\right) \\ & + \sum_{k+1 \leq i \leq k'} B\left(f^{-1}(N_k(a_j)), \emptyset\right) \end{aligned}$$

Since the BD satisfies the triangle inequality (Proposition 6),

$$\begin{aligned} & B\left(f^{-1}(N_k(a_i)), g(N_k(b_i))\right) \\ \leq \ & B\left(f^{-1}(N_k(a_i)), N_k(a_i)\right) + B(N_k(a_i), \emptyset) + B(\emptyset, N_k(b_i)) + B\left(N_k(b_i), g(N_k(b_i))\right), 1 \leq i \leq k. \end{aligned}$$

$$\begin{aligned} C \ \leq \ & \sum_{v \in W_{11}} \left[B\left(f^{-1}(N_k(v)), N_k(v)\right) + B(N_k(v), g(N_k(v)))\right] \\ & + \sum_{1 \leq i \leq k} \left[B\left(f^{-1}(N_k(a_i)), N_k(a_i)\right) + B(N_k(a_i), \emptyset) + B(\emptyset, N_k(b_i)) + B(N_k(b_i), g(N_k(b_i)))\right] \\ & + \sum_{k+1 \leq i \leq k'} \left[B\left(f^{-1}(N_k(a_j)), N_k(v)\right) + B(N_k(v), \emptyset)\right] \\ \leq \ & \sum_{v \in V(W)} B\left(f^{-1}(N_k(v)), N_k(v)\right) + \sum_{v \in V(W)} B(N_k(v), g(N_k(v))) \\ = \ & B_k(S, W) + B_k(W, T). \end{aligned}$$

By definition, $B_k(S, T) \leq C$, implying the triangle inequality.

Case 2. $t \geq w$.
Let us assume that

$$f : \{N_k(v) : v \in V(S)\} \to \{N_k(v),\ \emptyset_i : v \in V(W), 1 \leq i \leq s - w\}$$

is a 1-to-1 function such that $\{(N_k(v), f(N_k(v))) \ : \ v \in V(S)\}$ is a minimum weight perfect matching in $\mathrm{BG}_k(S, W)$, and assume that

$$g : \{N_k(v),\ \emptyset_i : v \in V(W), 1 \leq i \leq t - w\} \to \{N_k(v) : v \in V(T)\}$$

is a 1-to-1 function such that $\{(N_k(v), g(N_k(v))), (\emptyset_i, g(\emptyset_i)) \ : \ v \in V(W), 1 \leq i \leq t-w\}$ is the minimum weight perfect matching in $\mathrm{BG}_k(W,T)$. Then,

$$\left\{\left(f^{-1}(N_k(v)), g(N_k(v))\right) \ : \ v \in V(W)\right\} \cup \left\{\left(f^{-1}(\emptyset_i), g(\emptyset_i)\right) \ : \ 1 \leq i \leq t-w\right\}$$
$$\cup \left\{\left(f^{-1}(\emptyset_j), \emptyset_{j-t+w}\right) \ : \ t-w < j \leq s-w\right\}$$

defines a perfect matching in $BG_k(S,T)$ and its weight C can be bounded by:

$$\begin{aligned} C \quad &\leq \quad \sum_{v \in V(W)} \left[B_k\left(f^{-1}(N_k(v)), N_k(v)\right) + B_k(N_k(v), g(N_k(v)))\right] \\ &\quad \sum_{1 \leq i \leq t-w} \left[B_k\left(f^{-1}(\emptyset_i)\,, \emptyset_i\right) + B_k(\emptyset_i, g(\emptyset_i))\right] + \sum_{t-w<j\leq t-w} B_k(f^{-1}(\emptyset_i), \emptyset_{j-t+w}) \\ &= \quad B_k(S,W) + B_k(W,T). \end{aligned}$$

Case 3. $s > w > t$. Let us assume that

$$f : \{N_k(v) : v \in V(S)\} \to \{N_k(v), \emptyset_i : v \in V(W), 1 \leq i \leq s-w\}$$

is a 1-to-1 function such that $\{(N_k(v), f(N_k(v))) \ : \ v \in V(S)\}$ is the minimum weight perfect matching in $\mathrm{BG}_k(S,W)$, and assume that

$$g : \{N_k(v) : v \in V(W)\} \to \{N_k(v), \ \emptyset_i \ : \ v \in V(T), 1 \leq i \leq w-t\}$$

is a 1-to-1 function such that $\{(N_k(v), g(N_k(v))) \ : \ v \in W\}$ is the minimum weight perfect matching in $\mathrm{BG}_k(W,T)$. Then,

$$\{(f^{-1}(N_k(v)), g(N_k(v))), (f^{-1}(\emptyset_j), \emptyset_j) \ : \ v \in V(W), 1 < j \leq s-w\}$$

is a perfect matching in $\mathrm{BG}_k(S,T)$ and its weight is:

$$\begin{aligned} C \quad &= \quad \sum_{v \in V(W)} B(f^{-1}(N_k(v)), g(N_k(v))) + \sum_{1 \leq i \leq s-w} B(f^{-1}(\emptyset_j), \emptyset_j)) \\ &\leq \quad \sum_{v \in V(W)} \left[B(f^{-1}(N_k(v)), N_k(v)) + B(N_k(v), g(N_k(v)))\right] + \sum_{1 \leq i \leq s-w} B(f^{-1}(\emptyset_j), \emptyset_j)) \\ &\leq \quad B_k(S,W) + B_k(W,T), \end{aligned}$$

where the inequality is derived from the triangle inequality. ◀

A3. Measures for comparing mutation trees

The CASet and DISC metrics

Recently, two metrics were introduced for mutation trees [18]. Let M be a label set and T be a rooted tree in which the nodes are uniquely labeled with the parts of a partitions of M. For a node $u \in V(T)$, we use $\ell(u)$ to denote the label of u. For each $m \in M$, we use $\ell^{-}(m)$ to denote the unique node whose label contains m.

Recall that $A_T(u)$ denotes the set of ancestors of u and $u \notin A_T(u)$. For any $m \in M$, define $A_T(m) = \sum_{u \in A_T(\ell^{-}(m))} \ell(u)$. Note that $A_T(m') \cap A_T(m'')$ is equal to the set of their common ancestors for any m' and m'' of M.

Let S and T be two rooted labeled trees S and T whose nodes are uniquely labeled with the elements of M. The *Common Ancestor Set* (CASet) metric between S and T is defined as the average the Jaccard distance between the sets of common ancestors of two labels in S and T [18], i.e.,

$$\mathrm{CASet}(S,T) \triangleq \frac{1}{\binom{m}{2}} \sum_{i,j \in M : i<j} \frac{|(A_S(i) \cap A_S(j)) \triangle (A_T(i) \cap A_T(j))|}{|(A_S(i) \cap A_S(j)) \cup (A_T(i) \cap A_T(j))|},$$

where $A_S(i)$ is the empty set if i is not in the label set of S or it is an element of the label of the root of S. Here, the Jaccard distance between the empty set and itself is 0.

We use $D_S(i,j)$ to denote $A_S(i) \setminus A_S(j)$ for any two labels. The *Distinctly Inherited Set Comparison* (DISC) metric between S and T is defined to be [18]:

$$\text{DISC}(S,T) \triangleq \frac{1}{m(m-1)} \sum_{i,j \in M : i \neq j} \frac{|D_S(i,j) \triangle D_T(i,j)|}{|D_S(i,j) \cup D_T(i,j)|}.$$

In a mutation tree, the nodes are labeled with disjoint subsets of the label set; a label appearing in a tree may not appear in another tree inferred for the same patient. It is not hard to generalize the CASet and DISC in the context of mutation trees [18].

An ancestor difference metric

One reason to introduce the Bourque distance is that every uniquely labeled tree can be uniquely reconstructed from all its node-induced star subtrees. It is not hard to see that every rooted uniquely labeled tree can also be reconstructed from the paths from the root to all other nodes. Hence, the difference between two mutation trees on M can be measured by the Ancestor Difference (AD) metric defined by:

$$\text{AD}(S,T) \triangleq \sum_{m \in M} |A_S(m) \triangle A_T(m)|.$$

The AD metric has been used for comparing mutation trees in [18, 19]. Note that CASet, DISC and AD metrics do not satisfy the triangle inequality in general.

The triplet-based distance

The triplet distance has also been generalized to mutation trees [5]. In a mutation tree, any three labeled nodes induce a labeled tree that has three labeled nodes at most. The triplet-based distance (TD) between two mutation trees S and T with the same label set is defined by:

$$\text{TD}(S,T) = 1 - \frac{|\text{Triplets}(S) \cap \text{Triplets}(T)|}{\max\left(|\text{Triplets}(S)|, |\text{Triplets}(T)|\right)},$$

where Triplets(S) denotes the set of possible subtrees induced by three different labels.

A4. Supplementary Figure S1

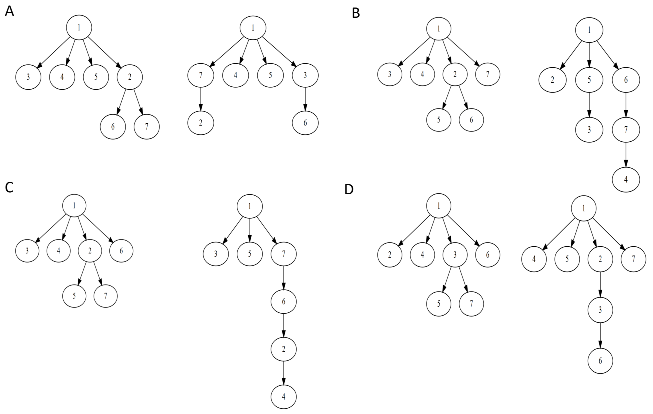


Figure S1 The two trees that have the same AD (A), CASet (B), DISC (C) and TD (D) distance but different DB distances from the 1-labeled star tree centered at Node 1.

Advancing Divide-And-Conquer Phylogeny Estimation Using Robinson-Foulds Supertrees

Xilin Yu[1]
Amazon AWS, Seattle, WA, USA
yuxilin51@gmail.com

Thien Le[2]
Department of EECS, Massachusetts Institute of Technology, Cambridge, MA, USA.
thienle@mit.edu

Sarah Christensen
Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA
sac2@illinois.edu

Erin K. Molloy
Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA
emolloy2@illinois.edu

Tandy Warnow
Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA
warnow@illinois.edu

Abstract

One of the Grand Challenges in Science is the construction of the *Tree of Life*, an evolutionary tree containing several million species, spanning all life on earth. However, the construction of the Tree of Life is enormously computationally challenging, as all the current most accurate methods are either heuristics for **NP**-hard optimization problems or Bayesian MCMC methods that sample from tree space. One of the most promising approaches for improving scalability and accuracy for phylogeny estimation uses divide-and-conquer: a set of species is divided into overlapping subsets, trees are constructed on the subsets, and then merged together using a "supertree method". Here, we present Exact-RFS-2, the first polynomial-time algorithm to find an optimal supertree of two trees, using the Robinson-Foulds Supertree (RFS) criterion (a major approach in supertree estimation that is related to maximum likelihood supertrees), and we prove that finding the RFS of three input trees is **NP**-hard. We also present GreedyRFS (a greedy heuristic that operates by repeatedly using Exact-RFS-2 on pairs of trees, until all the trees are merged into a single supertree). We evaluate Exact-RFS-2 and GreedyRFS, and show that they have better accuracy than the current leading heuristic for RFS.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases supertrees, divide-and-conquer, phylogeny estimation

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.15

Related Version A full version of the paper is available at https://www.biorxiv.org/content/10.1101/2020.05.16.099895.

Supplementary Material Exact-RFS-2 and GreedyRFS are available in open source form on Github at https://github.com/yuxilin51/GreedyRFS.

Funding *Sarah Christensen*: Ira and Debra Cohen Fellowship.
Erin K. Molloy: NSF Graduate Research Fellowship DGE-1144245 and the Ira and Debra Cohen Fellowship.
Tandy Warnow: US National Science Foundation grants 1458652 1513629, and 1535977.

[1] Corresponding author. This research was performed while Xilin Yu was a graduate student in the Department of Computer Science at the University of Illinois.

[2] This work was performed while Thien Le was an undergraduate at the University of Illinois

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti; Article No. 15; pp. 15:1–15:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications.

1 Introduction

Supertree construction (i.e., the combination of a collection of trees, each on a potentially different subset of the species, into a tree on the full set of species) is a natural algorithmic problem that has important applications to computational biology; see [7] for a 2004 book on the subject and [25, 38, 14, 18, 17, 16, 9, 10] for some of the recent papers on this subject. Supertree methods are particularly important for large-scale phylogeny estimation, where it can be used as a final step in a divide-and-conquer pipeline [45]: the species set is divided into two or more overlapping subsets, unrooted leaf-labelled trees are constructed (possibly recursively) on each subset, and then these subset trees are combined into a tree on the full dataset, using the selected supertree method. Furthermore, provided that optimal supertrees are computed, divide-and-conquer pipelines can be provably *statistically consistent* under stochastic models of evolution: i.e., as the amount of input data (e.g., sequence lengths when estimating gene trees, or number of gene trees when estimating species trees) increases, the probability that the true tree is returned converges to 1 [24, 44].

Unfortunately, the most accurate supertree methods are typically local-search heuristics for **NP**-hard optimization problems [4, 29, 25, 38, 33, 27, 34, 16], and are computationally intensive on large datasets. However, divide-and-conquer strategies, especially recursive ones, may only need to apply supertree methods to two trees at a time, and hence the computational complexity of supertree estimation given two trees is of interest. One optimization problem where optimal supertrees can be found on two trees is the **NP**-hard Maximum Agreement Supertree (SMAST) problem (also known as the Agreement Supertree Taxon Removal problem), which removes a minimum number of leaves so that the reduced trees have an agreement supertree [17, 14]. Similarly, the Maximum Compatible Supertree (SMCT) problem, which removes a minimum number of leaves so that the reduced trees have a compatibility supertree [5, 6], can also be solved in polynomial time on two trees (and note that SMAST and SMCT are identical when the input trees are fully resolved). Because SMAST and SMCT remove taxa, methods for these optimization problems are not *true supertree methods*, because they do not return a tree on the entire set of taxa. However, solutions to SMAST and SMCT could potentially be used as *constraints* for other supertree methods, where the deleted leaves are added into the computed SMAST or SMCT trees, so as to optimize the desired criterion.

When restricting to methods that return trees on the full set of taxa, much less seems to be understood about finding supertrees on two trees. However, if the two input trees are compatible (i.e., there is a supertree that equals or refines each tree when restricted to the respective leaf set), then finding that compatibility supertree is solvable in polynomial time, using (for example) the well known BUILD algorithm [1], but more efficient algorithms exist (e.g., [6, 3]).

Since compatibility is a strong requirement (rarely seen in biological datasets), optimization problems are more relevant. One optimization problem worth discussing is the Maximum Agreement Supertree Edge Contraction problem (which takes as input a set of rooted trees and seeks a minimum number of edges to collapse so that an agreement supertree exists). This problem is **NP**-hard, but the decision problem can be solved in $O((2k)^p kn^2)$ time when the input has k trees and p is the allowed number of number of edges to be collapsed [14]. Note that the algorithm for AST-EC proposed by [14] may be exponential even for two trees, when the number of edges that must be collapsed is $\Omega(n)$ (e.g., imagine two caterpillar trees, where one is obtained from the other by moving the left-most leaf to the rightmost position).

In sum, while supertree methods are important and well studied, when restricted to the major optimization problems that do not remove taxa, polynomial time methods do not seem to be available, even for the special case where the input contains just two trees. This restriction has consequences for large-scale phylogeny estimation, as without good supertree methods, divide-and-conquer pipelines are not guaranteed to be statistically consistent, are not fast, and do not have good scalability [44].

In this paper we examine the well known Robinson-Foulds Supertree (RFS) problem [2], which seeks a supertree that minimizes the total Robinson-Foulds [30] distance to the input trees. Although RFS is **NP**-hard [20], it has several desirable properties: it is closely related to maximum likelihood supertrees [36] and, as shown very recently, has good theoretical performance for species tree estimation in the presence of gene duplication and loss [23]. Because of its importance, there are several methods for RFS supertrees, including PluMiST [18], MulRF [8], and FastRFS [42]. A comparison between FastRFS and other supertree methods (MRL [25], ASTRAL, ASTRID [41], PluMiST, and MulRF) on simulated and biological supertree datasets showed that FastRFS matched or improved on the other methods with respect to topological accuracy and RFS criterion scores [42]. Hence, FastRFS is currently the leading method for the RFS optimization problem.

The main contributions of this paper are:

- We prove that RFS is solvable in $O(n^2|X|)$ time for two trees, where n is the number of leaves and X is the number of shared leaves (Theorem 2) and **NP**-hard for three or more trees (Lemma 10).
- We present Exact-RFS-2, a polynomial time algorithm for the RFS problem when given only two source trees, and explore its performance on simulated data, both within a natural divide-and-conquer pipeline and within a greedy heuristic (Section 3). We show that Exact-RFS-2 outperforms FastRFS [42] on two trees, the current most accurate method for RFS, and that GreedyRFS is better than FastRFS for small to moderate numbers of source trees (Section 4).
- We prove that divide-and-conquer pipelines using Exact-RFS-2 are statistically consistent methods for phylogenetic tree estimation (both gene trees and species trees) under standard sequence evolution models (Theorem 12).
- We establish equivalence between RFS and some other supertree problems (Lemma 1).
- We show critical differences between RFS and SMAST/SMCT problems, that establish that methods for SMAST or SMCT cannot provably be used to constrain the search for RFS supertrees (Lemma 23 in Appendix in the full version on bioRxiv).

The remainder of the paper is organized as follows. In Section 2, we provide terminology and define the optimization problems we consider. We present the Exact-RFS-2 algorithm and establish theory related to the algorithm in Section 3. Our experimental performance study is presented in Section 4, and we conclude in Section 5 with a discussion of trends and future research directions.

2 Terminology and Problem Statements

We let $[N] = \{1, 2, \ldots, N\}$ and $\mathcal{A} = \{T_i \mid i \in [N]\}$ denote the input to a supertree problem, where each T_i is a phylogenetic tree on leaf set $L(T_i) = S_i \subseteq S$ (where $L(t)$ denotes the leaf set of t) and the output is a tree T where $L(T)$ is the set of all species that appear as a leaf in at least one tree in $\mathcal{A}$, which we will assume is all of S. We use the standard supertree terminology, and refer to the trees in $\mathcal{A}$ as "source trees" and the set $\mathcal{A}$ as a "profile". For a tree T, let $V(T)$ and $E(T)$ denote the set of vertices and edges of T, respectively.

Robinson-Foulds Supertree

Each edge e in a tree T defines a bipartition $\pi_e := [A|B]$ of the leaf set, and each tree is defined by the set $C(T) := \{\pi_e \mid e \in E(T)\}$. The *Robinson-Foulds distance* [30] (also called the bipartition distance) between trees T and T' with the same leaf set is $\mathrm{RF}(T,T') := |C(T) \backslash C(T')| + |C(T') \backslash C(T)|$. We extend the definition of RF distance to allow for T and T' to have different leaf sets as follows: $RF(T,T') := RF(T|_X, T'|_X)$, where X is the shared leaf set and $t|_X$ denotes the homeomorphic subtree of t induced by X. Letting $\mathcal{T}_S$ denote the set of all phylogenetic trees such that $L(T) = S$ and $\mathcal{T}_S^B$ denote the binary trees in $\mathcal{T}_S$, then a Robinson-Foulds supertree [2] of a profile $\mathcal{A}$ is a binary tree

$$T_{\mathrm{RFS}} = \operatorname*{argmin}_{T \in \mathcal{T}_S^B} \sum_{i \in [N]} \mathrm{RF}(T, T_i).$$

We let $\mathrm{RF}(T,\mathcal{A}) := \sum_{i\in[N]} \mathrm{RF}(T,T_i)$ denote the *RFS score* of T with respect to profile $\mathcal{A}$. Thus, the **Robinson-Foulds Supertree problem** takes as input the profile $\mathcal{A}$ and seeks a Robinson-Foulds (RF) supertree for $\mathcal{A}$, which we denote by $\mathrm{RFS}(\mathcal{A})$.

Split Fit Supertree

The Split Fit (SF) Supertree problem was introduced in [46], and is based on optimizing the number of shared splits (i.e., bipartitions) between the supertree and the source trees. For two trees T, T' with the same leaf set, the *split support* is the number of shared bipartitions, i.e., $\mathrm{SF}(T,T') := |C(T) \cap C(T')|$. For trees with different leaf sets, we restrict them to the shared leaf set before calculating the split support. The Split Fit supertree for a profile $\mathcal{A}$ of source trees, denoted $\mathrm{SFS}(\mathcal{A})$, is a tree $T_{\mathrm{SFS}} \in \mathcal{T}_S^B$ such that

$$T_{\mathrm{SFS}} = \operatorname*{argmax}_{T \in \mathcal{T}_S^B} \sum_{i \in [N]} \mathrm{SF}(T, T_i).$$

Thus, the split support score of T with respect to $\mathcal{A}$ is $\mathrm{SF}(T,\mathcal{A}) := \sum_{i\in[N]} \mathrm{SF}(T,T_i)$. The **Split Fit Supertree (SFS) problem** takes as input the profile $\mathcal{A}$ and seeks a Split Fit supertree (the supertree with the maximum split support score), which we denote by $\mathrm{SFS}(\mathcal{A})$.

Nomenclature for variants of RFS and SFS problems

- The relaxed versions of the problems where we do not require the output to be binary (i.e., we allow $T \in \mathcal{T}_S$) are named Relax-RFS and Relax-SFS.
- We append "-N" to the name to indicate that we assume there are N source trees. If no number is specified then the number of source trees is unconstrained.
- We append "-B" to the name to indicate that the source trees are required to be binary; hence, we indicate that the source trees are allowed to be non-binary by not appending -B.

Thus, the RFS problem with two binary input trees is RFS-2-B and the relaxed SFS problem with three (not necessarily binary) input trees is Relax-SFS-3.

Other notation

For any $v \in V(T)$, we let $N_T(v)$ denote the set of neighbors of v in T. A tree T' is a *refinement* of T iff T can be obtained from T' by contracting a set of edges. Two bipartitions π_1 and π_2 of the same leaf set are said to be *compatible* if and only if there exists a tree T such that $\pi_i \in C(T), i = 1, 2$. A bipartition $\pi = [A|B]$ restricted to a subset R is $\pi|_R = [A \cap R|B \cap R]$. For a graph G and a set F of vertices or edges, we use $G + F$ to represent the graph obtained from adding the set F of vertices or edges to G, and $G - F$ is defined for deletions, similarly.

3 Theoretical Results

In this section we establish the main theoretical results, with detailed proofs provided in [47] or in the Appendix in the full version on bioRxiv [48].

3.1 Solving RFS and SFS on two trees

▶ **Lemma 1.** *Given an input set $\mathcal{A}$ of source trees, a tree $T \in \mathcal{T}_S^B$ is an optimal solution for RFS($\mathcal{A}$) if and only if it is an optimal solution for SFS($\mathcal{A}$).*

The main result of this paper is Theorem 2 (correctness is proved later within the main body of the paper, and the running time is established in the Appendix):

▶ **Theorem 2.** *Let $\mathcal{A} = \{T_1, T_2\}$ with S_i the leaf set of T_i ($i = 1, 2$) and $X := S_1 \cap S_2$. The problems RFS-2-B($\mathcal{A}$) and SFS-2-B($\mathcal{A}$) can be solved in $O(n^2|X|)$ time, where $n := \max\{|S_1|, |S_2|\}$.*

3.1.1 Exact-RFS-2: Polynomial time algorithm for RFS-2-B and SFS-2-B

The input to Exact-RFS-2 is a pair of binary trees T_1 and T_2. Let X denote the set of shared leaves. At a high level, Exact-RFS-2 constructs a tree T_{init} that has a central node that is adjacent to every leaf in X and to the root of every "rooted extra subtree" (a term we define below under "Additional notation") so that T_{init} contains all the leaves in S. It then modifies T_{init} by repeatedly refining it to add specific desired bipartitions, to produce an optimal Split Fit (and optimal Robinson-Foulds) supertree (Figure 3). The bipartitions that are added are defined by a maximum independent set in a bipartite "weighted incompatibility graph" we compute.

Additional notation

Let 2^X denote the set of all bipartitions of X; any bipartition that splits a single leaf from the remaining $|X| - 1$ leaves will be called "trivial" and the others will be called "non-trivial". Let $C(T_1, T_2, X)$ denote $C(T_1|_X) \cup C(T_2|_X)$, and let Triv and NonTriv denote the sets of trivial and non-trivial bipartitions in $C(T_1, T_2, X)$, respectively. We refer to $T_i|_X, i = 1, 2$ as **backbone trees** (Figure 2). Recall that we suppress degree-two vertices when restricting a tree T_i to a subset X of the leaves; hence, every edge e in $T_i|_X$ will correspond to an edge or a path in T (see Fig. 1 for an example). We will let $P(e)$ denote the path associated to edge e, and let $w(e) := |P(e)|$ (the number of edges in $P(e)$). Finally, for $\pi \in C(T_i|_X)$, we define $e_i(\pi)$ to be the edge that induces π in $T_i|_X$ (Fig. 1).

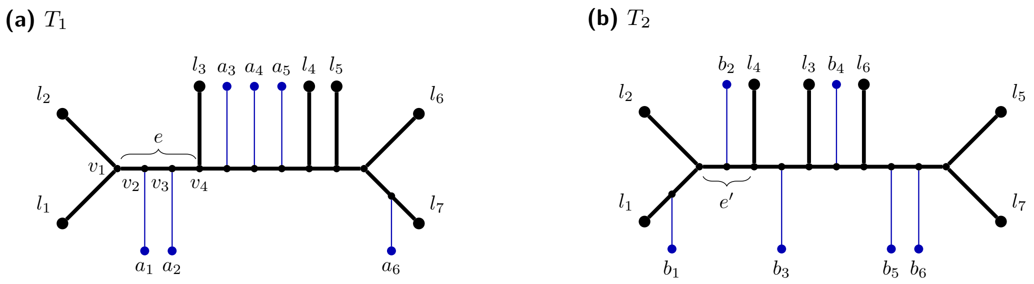


Figure 1 T_1 and T_2 depicted in (a) and (b) have an overlapping leaf set $X = \{l_1, l_2, \ldots, l_7\}$. Each of $a_1, \ldots, a_6$ and $b_1, \ldots, b_6$ can represent a multi-leaf extra subtree. For $e \in T_1|_X$ as shown, $P(e)$ is the path from v_1 to v_4, so $w(e) = 3$. Using indices to represent the shared leaves, let $\pi = [12|34567]$; then $e_1(\pi) = e$ and $e_2(\pi) = e'$. $\mathcal{TR}(e) = \{a_1, a_2\}$, $\mathcal{TR}(e') = \{b_2\}$, so $\mathcal{TR}^*(\pi) = \{a_1, a_2, b_2\}$. Let $A = \{1, 2, 3\}$, $B = \{4, 5, 6, 7\}$. Ignoring the trivial bipartitions, we have $\mathcal{BP}(A) = \{[12|34567]\}$ and $\mathcal{BP}(B) = \{[1234|567], [12345|67], [12346|57]\}$. $\mathcal{TRS}(A) = \{a_1, a_2, b_1, b_2\}$ and $\mathcal{TRS}(B) = \{a_6, b_4, b_5, b_6\}$.

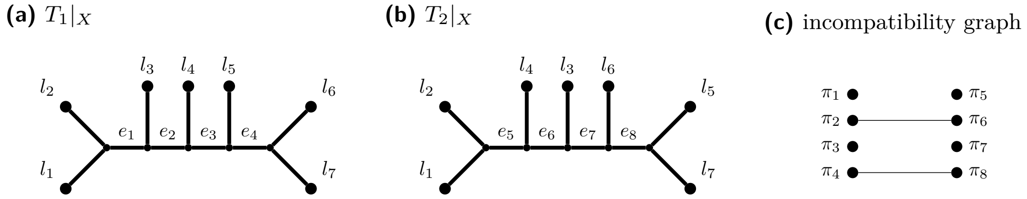


Figure 2 We show (a) $T_1|_X$, (b) $T_2|_X$, and (c) their incompatibility graph, based on the trees T_1 and T_2 in Figure 1 (without the trivial bipartitions). Each π_i is the bipartition induced by e_i, and the weights for $\pi_1, \ldots, \pi_8$ are $3, 4, 1, 1, 2, 2, 2, 3$, in that order. We note that π_1 and π_5 are the same bipartition, but they have different weights as they are induced by different edges; similarly for π_3 and π_7. The maximum weight independent set in this graph has all the isolated vertices ($\pi_1, \pi_3, \pi_5, \pi_7$) and also π_2, π_8, and so has total weight 15.

The next concept we introduce is the set of **extra subtrees**, which are rooted subtrees of T_1 and T_2, formed by deleting X and all the edges and vertices on paths between vertices in X (i.e., we delete $T_i|_X$ from T_i). Each component in $T_i - T_i|_X$ is called an **extra subtree** of T_i, and note that the extra subtree t is naturally seen as rooted at the unique vertex $r(t)$ that is adjacent to a vertex in $T_i|_X$. Thus, $\text{Extra}(T_i) = \{t \mid t \text{ is a component in } T_i - T_i|_X\}$.

We can now define the initial tree T_{init} computed by Exact-RFS-2: T_{init} has a center node that is adjacent to every $x \in X$ and also to the root $r(t)$ for every extra subtree $t \in \text{Extra}(T_1) \cup \text{Extra}(T_2)$. Note that T_{init} has a leaf for every element in S, and that $T_{\text{init}}|_{S_i}$ is a contraction of T_i, formed by collapsing all the edges in the backbone tree $T_i|_X$.

We say that an extra subtree t is **attached to edge** $e \in E(T_i|_X)$ if the root of t is adjacent to an internal node of $P(e)$, and we let $\mathcal{TR}(e)$ denote the set of such extra subtrees attached to edge e. Similarly, if $\pi \in C(T_1, T_2, X)$, we let $\mathcal{TR}^*(\pi)$ refer to the set of extra subtrees that attach to edges in a backbone tree that induce π in either $T_1|_X$ or $T_2|_X$. For example, if both trees T_1 and T_2 contribute extra subtrees to π, then $\mathcal{TR}^*(\pi) := \bigcup_{i\in[2]} \mathcal{TR}(e_i(\pi))$.

For any $Q \subseteq X$, we let $\mathcal{BP}_i(Q)$ denote the set of bipartitions in $C(T_i|_X)$ that have one side being a strict subset of Q, and we let $\mathcal{TRS}_i(Q)$ denote the set of extra subtrees associated with these bipartitions. In other words, $\mathcal{BP}_i(Q) := \{[A|B] \in C(T_i|_X) \mid A \subsetneq Q \text{ or } B \subsetneq Q\}$, and $\mathcal{TRS}_i(Q) := \bigcup_{\pi\in\mathcal{BP}_i(Q)} \mathcal{TR}(e_i(\pi))$. Intuitively, $\mathcal{TRS}_i(Q)$ denotes the set of extra subtrees in T_i that are "on the side of Q". By Corollary 14, which appears in the Appendix, for any $\pi =$

Algorithm 1 Exact-RFS-2: Computing a Robinson-Foulds supertree of two trees (see Figure 3).

```
    Input: two binary trees $T_1$, $T_2$ with leaf sets $S_1$ and $S_2$ where $S_1 \cap S_2 = X \neq \emptyset$
    Output: a binary supertree $T$ on leaf set $S = S_1 \cup S_2$ that maximizes the split support score
 1: compute $C(T_1|_X)$ and $C(T_2|_X)$
 2: for each $\pi = [A|B] \in C(T_1, T_2, X)$ do
 3:     for $i \in [2]$ do
 4:         compute $\mathcal{TR}(e_i(\pi))$, $w(e_i(\pi))$
 5:     compute $\mathcal{BP}(A)$, $\mathcal{BP}(B)$, $\mathcal{TRS}(A)$, $\mathcal{TRS}(B)$, and $\mathcal{TR}^*(\pi)$,
 6: construct $T$ as a star tree with leaf set $X$ and center vertex $\hat{v}$ and with the root of each $t \in$
    Extra$(T_1) \cup$ Extra$(T_2)$ connected to $\hat{v}$ by an edge                ▷ let $T_{\text{init}} = T$
 7: construct the weighted incompatibility graph $G$ of $T_1|_X$ and $T_2|_X$
 8: compute the maximum weight independent set $I^*$ in $G$
 9: let $I$ be the set of bipartitions associated with vertices in $I^*$
10: for each $\pi = [\{a\}|B] \in$ Triv do
11:     detach all extra subtrees in $\mathcal{TR}^*(\pi)$ from $\hat{v}$ and attach them onto $(\hat{v}, a)$ such that $\mathcal{TR}(e_1(\pi))$ are
    attached first with their ordering matching their attachments on $e_1(\pi)$ and $\mathcal{TR}(e_2(\pi))$ are attached
    to the right of all subtrees in $\mathcal{TR}(e_1(\pi))$ with the ordering of them also matching their attachments
    on $e_2(\pi)$
                                                                   ▷ let $\tilde{T} = T$ after for loop
12: $H(\hat{v}) =$ NonTriv, set $sv(\pi) = \hat{v}$ for all $\pi \in$ NonTriv
13: for each $\pi \in$ NonTriv $\cap I$ (in any order) do
14:     $T \leftarrow$ Refine$(T, \pi, H, sv)$                              ▷ let $T^* = T$ after for loop
15: arbitrarily refine $T$ to make it a binary tree
16: return $T$
```

$[A|B] \in C(T_i|_X)$, $\mathcal{BP}_i(A) \cup \mathcal{BP}_i(B)$ is the set of bipartitions in $C(T_i|_X)$ that are compatible with π. Finally, let $\mathcal{BP}(Q) = \mathcal{BP}_1(Q) \cup \mathcal{BP}_2(Q)$, and $\mathcal{TRS}(Q) = \mathcal{TRS}_1(Q) \cup \mathcal{TRS}_2(Q)$. We give an example for these terms in Figure 1.

The *incompatibility graph* of a set of trees, each on the same set of leaves, has one vertex for each bipartition in any tree (and note that bipartitions can appear more than once) and edges between bipartitions if they are incompatible (see [28]). We compute a **weighted incompatibility graph** for the pair of trees $T_1|_X$ and $T_2|_X$, in which the weight of the vertex corresponding to bipartition π appearing in tree $T_i|_X$ is $w(e_i(\pi))$, as defined previously. Thus, if a bipartition is common to the two trees, it produces two vertices in the weighted incompatibility graph, and each vertex has its own weight (Figure 2).

We divide $\mathcal{C} = C(T_1) \cup C(T_2)$ into two sets: $\Pi_1 = \{[A|B] \in \mathcal{C} \mid A \cap X \neq \emptyset \text{ and } B \cap X \neq \emptyset\}$, and $\Pi_2 = \{[A|B] \in \mathcal{C} \mid A \cap X = \emptyset \text{ or } B \cap X = \emptyset\}$. Intuitively, Π_1 is the set of bipartitions from the input trees that are induced by edges in the minimal subtree of T_1 or T_2 spanning X, and Π_2 are all the other input tree bipartitions. We define $p_1(\cdot)$ and $p_2(\cdot)$ on trees $T \in \mathcal{T}_S$ by:

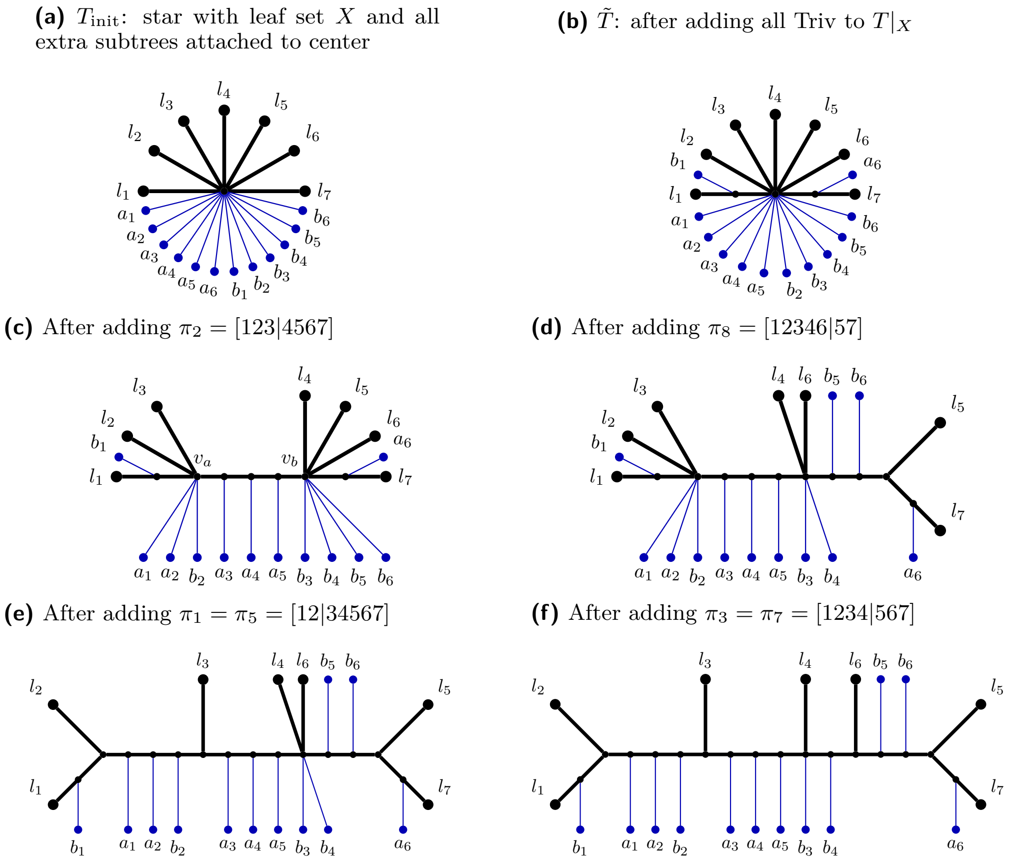
$$p_1(T) = \sum_{i \in [2]} |C(T|_{S_i}) \cap C(T_i) \cap \Pi_1|, \quad p_2(T) = \sum_{i \in [2]} |C(T|_{S_i}) \cap C(T_i) \cap \Pi_2|.$$

Note that $p_1(T)$ and $p_2(T)$ decompose the split support score of T into the score contributed by bipartitions in Π_1 and the score contributed by bipartitions in Π_2; thus, the split support score of T with respect to T_1, T_2 is $p_1(T) + p_2(T)$.

As we will show, the two scores can be maximized independently and we can use this observation to refine T_{init} so that it achieves the optimal total score.

Overview of Exact-RFS-2

Exact-RFS-2 (Algorithm 1) has four phases. In the pre-processing phase (lines 1–5), it computes the weight function w and the mappings $\mathcal{TR}, \mathcal{TR}^*, \mathcal{BP}$, and $\mathcal{TRS}$ for use in latter parts of Algorithm 1 and Algorithm 2. In the initial construction phase (line 6), it constructs a tree T_{init} (as described earlier), and we note that T_{init} maximizes $p_2(\cdot)$ score (Lemma 3). In the refinement phase (lines 7–14), it refines T_{init} so that it attains the maximum $p_1(\cdot)$ score. In the last phase (line 15), it arbitrarily refines T to make it binary. The refinement phase begins with the construction of a weighted incompatibility graph G of $T_1|_X$ and $T_2|_X$ (see Figure 2). It then finds a maximum weight independent set of G that defines a set $I \subseteq C(T_1, T_2, X)$ of compatible bipartitions of X. Finally, it uses these bipartitions of X in I to refine T_{init} to achieve the optimal $p_1(\cdot)$ score, by repeatedly applying Algorithm 2 for each $\pi \in I$ (and we note that the order does not matter). See Figure 3 for an example of Exact-RFS-2 given two input source trees.


(a) T_{init}: star with leaf set X and all extra subtrees attached to center

(b) $\tilde{T}$: after adding all Triv to $T|_X$

(c) After adding $\pi_2 = [123|4567]$

(d) After adding $\pi_8 = [12346|57]$

(e) After adding $\pi_1 = \pi_5 = [12|34567]$

(f) After adding $\pi_3 = \pi_7 = [1234|567]$

Figure 3 Algorithm 1 working on T_1 and T_2 from Figure 1 as source trees; the indices of leaves in $X = \{l_1, l_2, \ldots, l_7\}$ represent the leaves and the notation of $\pi_1, \ldots, \pi_8$ is from Figure 2. In (a) to (f), the $p_X(\cdot)$ score of the trees are $14, 16, 20, 23, 27, 29$, in that order. We explain how the algorithm obtain the tree in (c) from $\tilde{T}$ by adding $\pi_2 = [123|4567]$ to the backbone of $\tilde{T}$. Let $A = \{l_1, l_2, l_3\}$ and $B = \{l_4, l_5, l_6, l_7\}$. The center vertex c of $\tilde{T}$ is split into two vertices v_a, v_b with an edge between them. Then all neighbors of c between c and A are made adjacent to v_a while the neighbors between c and B are made adjacent to v_b. All neighbors of c which are roots of extra subtrees are moved around such that all extra subtrees in $\mathcal{TR}^*(\pi_2)$ are attached onto (v_a, v_b); all extra subtrees in $\mathcal{TRS}(A) = \{a_1, a_2, b_2\}$ are attached to v_a and all extra subtrees in $\mathcal{TRS}(B) = \{b_4, b_5, b_6\}$ are attached to v_b. We note that in this step, b_3 can attach to either v_a or v_b because it is not in $\mathcal{TRS}(A)$ or $\mathcal{TRS}(B)$. However, when obtaining the tree in (d) from the tree in (c), b_3 can only attach to the left side because for $A' = \{l_1, l_2, l_3, l_4, l_6\}$, $[124|3567] \in \mathcal{BP}(A')$ and thus $b_3 \in \mathcal{TRS}(A')$.

Algorithm 2 refines the given tree T on leaf set S with bipartitions on X from $C(T_1, T_2, X) \setminus C(T|_X)$. Given bipartition $\pi = [A|B]$ on X, Algorithm 2 produces a refinement T' of T such that $C(T'|_{S_i}) = C(T|_{S_i}) \cup \{\pi' \in C(T_i) \mid \pi'|_X = \pi\}$ for both $i = 1, 2$. To do this, we first

Algorithm 2 Refine.

```
Input: a tree $T$ on leaf set $S$, a nontrivial bipartition $\pi = [A|B]$ of $X$, two data structures $H$ and $sv$
Output: a tree $T'$ which is a refinement of $T$ such that for both $i = 1, 2$, $C(T'|_{S_i}) = C(T|_{S_i}) \cup \{\pi' \in C(T_i) \mid \pi'|_X = \pi\}$
$v \leftarrow sv(\pi)$
$T' \leftarrow T + v_a + v_b + (v_a, v_b)$
compute $N_A := \{u \in N_T(v) \mid \exists a \in A$ s.t. $u$ can reach $a$ in $T - v\}$ and $N_B := \{u \in N_T(v) \mid \exists b \in B$ s.t. $u$ can reach $b$ in $T - v\}$.
for each $u \in N_A \cup N_B$ do
    if $u \in N_A$ then connect $u$ to $v_a$
    else connect $u$ to $v_b$
detach all extra subtrees in $\mathcal{TR}^*(\pi)$ from $v$ and attach them onto $(v_a, v_b)$ such that $\mathcal{TR}(e_1(\pi))$ are attached first with their ordering matching their attachments on $e_1(\pi)$ and $\mathcal{TR}(e_2(\pi))$ are attached to the right of all subtrees in $\mathcal{TR}(e_1(\pi))$ with the ordering of them also matching their attachments on $e_2(\pi)$
for each $t \in \mathcal{TRS}(A)$ do
    if $t$ is attached to $v$, detach it and attach to $v_a$
for each $t \in \mathcal{TRS}(B)$ do
    if $t$ is attached to $v$, detach it and attach to $v_b$
for each remaining extra subtree attached to $v$ do
    detach it from $v$ and attach it to either $v_a$ or $v_b$
$H(v_a) \leftarrow \emptyset, H(v_b) \leftarrow \emptyset$
for each $\pi' \in H(v)$ do
    if $\pi' \in \mathcal{BP}(A)$ then
        $sv(\pi') = v_a$, $H(v_a) \leftarrow H(v_a) \cup \{\pi'\}$
    else if $\pi' \in \mathcal{BP}(B)$ then
        $sv(\pi') = v_b$, $H(v_b) \leftarrow H(v_b) \cup \{\pi'\}$
    else
        discard $\pi'$
return $T' = T' - v$
```

find the unique vertex v such that no component of $T - v$ has leaves from both A and B. We create two new vertices v_a and v_b with an edge between them. We divide the neighbor set of v into three sets: N_A is the set of neighbors that split v from leaves in A, N_B is the set of neighbors that split v from leaves in B, and N_{other} contains the remaining neighbors. Then, we make vertices in N_A adjacent to v_a and vertices in N_B adjacent to v_b. We note that $N_{\text{other}} = \emptyset$ if $X = S$ and thus there are no extra subtrees. In the case where $X \neq S$, N_{other} contains the roots of the extra subtrees adjacent to v and we handle them in four different cases to make T' include the desired bipartitions:

- those vertices that root extra subtrees in $\mathcal{TR}^*(\pi)$ are moved onto the edge (v_a, v_b) (by subdividing the edge to create new vertices, and then making these vertices adjacent to the new vertices)
- those vertices that root extra subtrees in $\mathcal{TRS}(A)$ are made adjacent to v_a
- those that root extra subtrees in $\mathcal{TRS}(B)$ are made adjacent to v_b
- the remaining vertices can be made adjacent to either v_a or v_b

Algorithms 1 and 2 also use two data structures (functions) H and sv: (1) For a given node $v \in V(T)$, $H(v) \subseteq C(T_1, T_2, X)$ is the set of bipartitions of X that can be added to $T|_X$ by refining $T|_X$ at v, and (2) Given $\pi \in C(T_1, T_2, X)$, $sv(\pi) = v$ means $\exists T'$, a refinement of T at v, so that $C(T'|_X) = C(T|_X) \cup \{\pi\}$.

▶ **Lemma 3.** *For any tree $T \in \mathcal{T}_S$, $p_2(T) \leq |\Pi_2|$. In particular, let T_{init} be the tree defined in line 6 of Algorithm 1. Then, $p_2(T_{\text{init}}) = |\Pi_2|$.*

Lemma 3 formally states that the tree T_{init} we build in line 6 of Exact-RFS-2 (Algorithm 1) maximizes the $p_2(\cdot)$ score. This lemma is true because there are only $|\Pi_2|$ bipartitions that can contribute to $p_2(\cdot)$ and T_{init} contains all of them by construction. We define the function $w^* : \Pi \to \mathbb{N}_{\geq 0}$ as follows:

$$w^*(\pi) = \begin{cases} 0 & \text{if } \pi \notin C(T_1, T_2, X), \\ w(e_1(\pi)) & \text{if } \pi \in C(T_1|_X) \setminus C(T_2|_X), \\ w(e_2(\pi)) & \text{if } \pi \in C(T_2|_X) \setminus C(T_1|_X), \\ \sum_{i \in [2]} w(e_i(\pi)) & \text{else.} \end{cases}$$

For any set F of bipartitions, we let $w^*(F) = \sum_{\pi \in F} w^*(\pi)$.

▶ **Lemma 4.** *Let $\pi = [A|B] \in \Pi$. Let $T \in \mathcal{T}_S$ be any tree with leaf set S such that $\pi \notin C(T|_X)$ but π is compatible with $C(T|_X)$. Let T' be a refinement of T such that for all $\pi' \in C(T'|_{S_i}) \setminus C(T|_{S_i})$ for some $i \in [2]$, $\pi'|_X = \pi$. Then, $p_1(T') - p_1(T) \leq w^*(\pi)$.*

▶ **Lemma 5.** *For any compatible set $F \subseteq \Pi$, let $T \in \mathcal{T}_S$ be any tree with leaf set S such that $C(T|_X) = F$. Then $p_1(T) \leq w^*(F)$.*

Lemma 4 shows that $w^*(\pi)$ represents the maximum potential increase in $p_1(\cdot)$ as a result of adding bipartition π to $T|_X$. The proof of Lemma 4 follows the idea that for any bipartition π of X, there are at most $w^*(\pi)$ edges in either T_1 or T_2 whose induced bipartitions become π when restricted to X. Therefore, by only adding π to $T|_X$, at most $w^*(\pi)$ more bipartitions get included in $C(T|_{S_1})$ or $C(T|_{S_2})$ so that they contribute to the increase of $p_1(T)$. The proof of Lemma 5 uses Lemma 4 repeatedly by adding the compatible bipartitions to the tree in an arbitrary order.

▶ **Proposition 6.** *Let $\tilde{T}$ be the tree constructed after line 11 of Algorithm 1, then $p_1(\tilde{T}) = w^*(\mathrm{Triv})$.*

The proof naturally follows by construction (Line 8 of Algorithm 1), and implies that the algorithm adds the trivial bipartitions of X (which are all in I) to $T|_X$ so that $p_1(T)$ reaches the full potential of adding those trivial bipartitions.

▶ **Lemma 7.** *Let T be a supertree computed within Algorithm 1 at line 14 immediately before a refinement step. Let $\pi = [A|B] \in \mathrm{NonTriv} \cap I$. Let T' be a refinement of T obtained from running Algorithm 2 with supertree T, bipartition π, and the auxiliary data structures H and sv. Then, $p_1(T') - p_1(T) = w^*(\pi)$.*

The idea for the proof of Lemma 7 is that for any non-trivial bipartition $\pi \in I$ of X to be added to $T|_X$, Algorithm 2 is able to split the vertex correctly and move extra subtrees around in a way such that each bipartition in T_1 or T_2 that is induced by an edge in $P(e_1(\pi))$ or $P(e_2(\pi))$, which is not in $T|_{S_1}$ or $T|_{S_2}$ before the refinement, becomes present in $T|_{S_1}$ or $T|_{S_2}$ after the refinement. Since there are exactly $w^*(\pi)$ such bipartitions, they increase $p_1(\cdot)$ by $w^*(\pi)$.

▶ **Proposition 8.** *Let G be the weighted incompatibility graph on $T_1|_X$ and $T_2|_X$, and let I be the set of bipartitions associated with vertices in I^*, which is a maximum weight independent set of G. Let F be any compatible subset of $C(T_1, T_2, X)$. Then $w^*(I) \geq w^*(F)$.*

We now restate and prove Theorem 2:

▶ **Theorem 2.** *Let $\mathcal{A} = \{T_1, T_2\}$ with S_i the leaf set of T_i $(i = 1, 2)$ and $X := S_1 \cap S_2$. The problems RFS-2-B($\mathcal{A}$) and SFS-2-B($\mathcal{A}$) can be solved in $O(n^2|X|)$ time, where $n := \max\{|S_1|, |S_2|\}$.*

Proof. First we claim that $p_1(T^*) \geq p_1(T)$ for any tree $T \in \mathcal{T}_S$, where T^* is defined as from line 14 of Algorithm 1. Fix arbitrary $T \in \mathcal{T}_S$ and let $F = C(T|_X)$. Then by Lemma 5, $p_1(T) \leq w^*(F)$. We know that $w^*(\pi) = 0$ for any $\pi \notin C(T_1, T_2, X)$, so $w^*(F) = w^*(F \cap C(T_1, T_2, X))$ and thus $p_1(T) \leq w^*(F \cap C(T_1, T_2, X))$. Since $F \cap C(T_1, T_2, X)$ is a compatible subset of $C(T_1, T_2, X)$, we have $w^*(F \cap C(T_1, T_2, X)) \leq w^*(I)$ by Proposition 8. Then $p_1(T) \leq w^*(I)$. Since $\text{Triv} \subseteq C(T_1|_X) \cap C(T_2|_X) \subseteq I$, we have $I = (\text{NonTriv} \cap I) \cup (\text{Triv} \cap I) = (\text{NonTriv} \cap I) \cup \text{Triv}$. Therefore, by Proposition 6 and Lemma 7, we have

$$p_1(T^*) = p_1(\tilde{T}) + \sum_{\pi \in \text{NonTriv} \cap I} w^*(\pi) = w^*(\text{Triv}) + w^*(\text{NonTriv} \cap I) = w^*(I).$$

Therefore, $p_1(T^*) = w^*(I) \geq p_1(T)$.

From Lemma 3 and the fact that a refinement of a tree never decreases $p_1(\cdot)$ and $p_2(\cdot)$, we also know that $p_2(T^*) \geq p_2(T_{\text{init}}) \geq p_2(T)$ for any tree $T \in \mathcal{T}_S$. Since for any $T \in \mathcal{T}_S$, $\text{SF}(T, \mathcal{A}) = p_1(T) + p_2(T)$, T^* achieves the maximum split support score with respect to $\mathcal{A}$ among all trees in $\mathcal{T}_S$. Thus, T^* is a solution to Relax–SFS-2-B (Corollary 9). If T^* is not binary, Algorithm 1 arbitrarily resolves every high degree node in T^* until it is a binary tree and then returns a tree that achieves the maximum split support score among all binary trees of leaf set S. See the Appendix for the running time analysis. ◀

▶ **Corollary 9.** *Let $\mathcal{A} = \{T_1, T_2\}$ with S_i the leaf set of T_i ($i = 1, 2$) and $X := S_1 \cap S_2$. Relax–SFS-2-B can be solved in $O(n^2|X|)$ time, where $n := \max\{|S_1|, |S_2|\}$.*

▶ **Lemma 10.** *RFS-3, SFS-3, and Relax–SFS-3 are all* **NP***-hard.*

The proof for this lemma can be found in the Appendix.

3.2 DACTAL-Exact-RFS-2

Let Φ be a model of evolution (e.g., GTR) for which statistically consistent methods exist, and we have some data (e.g., sequences) generated by the model and wish to construct a tree. We construct an initial estimate of the tree, and we select an edge e in the tree. The deletion of e and its endpoints creates four subtrees, and we let P be the set of the p nearest leaves to e taken from each subtree (including all leaves that tie for closest in each subtree). We define the subsets be $A \cup P$ and $B \cup P$, where $\pi_e = [A|B]$), and we re-estimate trees on these subsets and then combine the trees together using Exact-RFS-2. We call this the DACTAL-Exact-RFS-2 pipeline, due to its similarity to the DACTAL pipeline [24]. The DACTAL pipeline differs from the DACTAL-Exact-RFS-2 pipeline only in that it computes four trees (each containing the set P and otherwise being leaf-disjoint) and then combines the overlapping subset trees using the Strict Consensus Merger technique, and was proven statistically consistent when the subset trees are computed using statistically consistent methods.

Before we prove that DACTAL-Exact-RFS-2 can enable statistically consistent pipelines, we begin with some definitions. Given a tree T and an internal edge e in T, the deletion of the edge e and its endpoints defines four subtrees. A **short quartet around** e is a set of four leaves, one from each subtree, selected to be closest to the edge. Note that due to ties, there can be multiple short quartets around some edges. The set of short quartets for a tree T is the set of all short quartets around the edges of T. The **short quartet trees of** T is the set of quartet trees on short quartets induced by T. It is well known that the short quartet trees of a tree T define T, and furthermore T can be computed from this set in polynomial time [11, 12, 13].

▶ **Lemma 11.** *Let T be a binary tree on leaf set S and let $A, B \subseteq S$. Let $T_A = T|_A$ and $T_B = T|_B$ (i.e., T_A and T_B are induced subtrees). If every short quartet tree is induced in T_A or in T_B, then T is the unique compatibility supertree for T_A and T_B and Exact-2-RFS(T_A, T_B) $= T$.*

Proof. Because T_A and T_B are induced subtrees of T, it follows that T is a compatibility supertree for T_A and T_B. Furthermore, because every short quartet tree appears in at least one of these trees, T is the unique compatibility supertree for T_A and T_B (by results from [12, 13], mentioned above). Finally, because T is a compatibility supertree, the RFS score of T with respect to T_A, T_B is 0, which is the best possible. Since Exact-2-RFS solves the RFS problem on two binary trees, Exact-2-RFS returns T given input T_A and T_B. ◀

Thus, Exact-2-RFS is guaranteed to return the true tree when given two correct trees that have sufficient overlap (in that all short quartets are included). We continue with proving that these pipelines are statistically consistent.

▶ **Theorem 12.** *The DACTAL-Exact-RFS-2 pipeline is a statistically consistent method for estimating the unrooted tree topology under any model Φ for which statistically consistent unrooted tree topology estimation methods exist.*

Proof. The proof is very similar to the proof given for the original DACTAL pipeline in [24]. Let Φ be the stochastic evolution model. To establish statistical consistency of the DACTAL-Exact-RFS-2 pipeline (see above), we need to prove that as the amount of data increases the unrooted tree topology that is returned by the pipeline converge to the true unrooted tree topology. That is, we will show that for any $\epsilon > 0$, there is an amount of data so that the probability of returning the true tree topology given that amount of data is at least $1 - \epsilon$. Hence, let F be the method used to compute the starting tree, let G be the method used to compute the subset trees, and let $\epsilon > 0$ be given. Because F is statistically consistent under Φ, it follows that there is an amount of data so that the starting tree computed by F will have the true tree topology T with probability at least $1 - \epsilon/2$. Now consider the decomposition into two sets produced by the algorithm produced by deleting edge e, applied to a tree with the true unrooted tree topology. Note that for any $p \geq 1$, all the leaves appearing in any short quartet around e are placed in the set P. Now, subset trees are computed using G on $A \cup P$ and $B \cup P$, where $\pi_e = [A|B]$, which we will refer to as T_A and T_B, respectively. Since G is statistically consistent, for a large enough amount of data, T_A and T_B will have the true tree topology on their leaf sets ($T|_{L(T_A)}$ and $T|_{L(T_B)}$, respectively) with probability at least $1 - \epsilon/2$. When T_A and T_B are equal to the true trees on their leaf sets, then every short quartet tree of T is in T_A or T_B, so that by Lemma 11, T is the only compatibility supertree for T_A and T_B. Thus, under these conditions, Exact-2-RFS(T_A, T_B) returns T. Hence, for a large enough amount of data, Exact-2-RFS(T_A, T_B) returns T with probability at least $1 - \epsilon$, completing our proof. ◀

Hence, DACTAL+Exact-2-RFS is statistically consistent under all standard molecular sequence evolution models and also under the MSC+GTR model [43, 31] where gene trees evolve within species trees under the multi-species coalescent model (which addresses gene tree discordance due to incomplete lineage sorting [19]) and then sequences evolve down each gene tree under the GTR model.

Note that all that is needed for F and G to guarantee that the pipeline is statistically consistent is that they should be statistically consistent under Φ. However, for the sake of improving empirical performance, F should be fast so that it can run on the full dataset but G can be more freely chosen, since it will only be run on smaller datasets. Indeed, the user

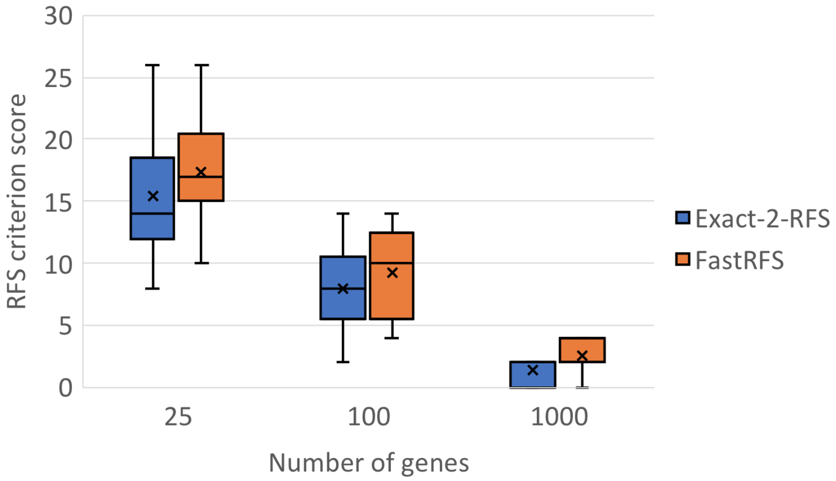


Figure 4 Results for Experiment 1: Exact-2-RFS has better RFS criterion scores than FastRFS (lower is better) in ILS-based species tree estimation (using ASTRAL-III [49], for 501 species with varying numbers of genes).

can specify the size of the subsets that are analyzed, with smaller datasets enabling the use of more computationally intensive methods.

For example, when estimating trees under the GTR [40] model, F could be a fast but statistically consistent distance-based method such as neighbor joining [32] and G could be RAxML [35], a leading maximum likelihood method. For the MSC+GTR model, F and G could be polynomial time summary methods (i.e., methods that estimate the species tree by combining gene trees), with F being ASTRID [41] (a very fast summary method) and G being ASTRAL [21, 22, 49], which is slower than ASTRID but often more accurate. However, if the subsets are chosen to be very small, then other choices for G include StarBeast2 [26], a Bayesian method for co-estimating gene trees and species trees under the MSC+GTR model.

4 Experiments and Results

We performed two experiments: Experiment 1, where we used Exact-2-RFS within a divide-and-conquer strategy for large scale phylogenomic species tree estimation where gene trees differ from the species tree due to Incomplete Lineage Sorting (ILS), and Experiment 2, where we used Exact-2-RFS as part of a greedy heuristic, GreedyRFS, for large-scale supertree estimation.

4.1 Experiment 1: Phylogenomic species tree estimation

In this experiment, the input is a set of gene trees that can differ from the species tree due to Incomplete Lineage Sorting [19], ASTRAL [21, 22, 49] is used to construct species trees on the two overlapping subsets in the DACTAL pipeline described above, and the two overlapping estimated species trees are then merged together using either Exact-2-RFS or FastRFS. Because the divide-and-conquer strategy produces two source trees, the RFS criterion score for Exact-2-RFS cannot be worse than the score obtained by FastRFS; here we examine the degree of improvement. The simulation protocol produced datasets with high variability (especially for small numbers of genes), so that there was substantial range in the optimal criterion scores for 25 and 100 genes (Fig. 4).

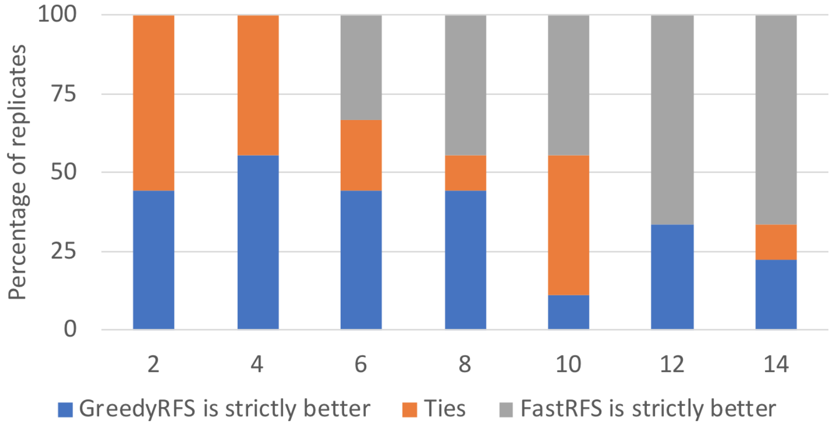


Figure 5 Results for Experiment 2: The percentage of datasets (y-axis) that each method (FastRFS and GreedyRFS) ties with or is strictly better than the other in terms of RFS criterion score is shown for varying numbers of source trees (x-axis), based on nine replicate supertree 500-leaf 20% scaffold datasets (from [39]).

On average, Exact-2-RFS produces better RFS scores than FastRFS for all numbers of genes (Fig. 4), showing that divide-and-conquer pipelines are improved using Exact-2-RFS compared to FastRFS.

4.2 Experiment 2: Exploring GreedyRFS for supertree estimation

We developed GreedyRFS, a greedy heuristic that takes a profile $\mathcal{A}$ as input, and then merges pairs of trees until all the trees are merged into a single tree. The choice of which pair to merge follows the technique used in SuperFine [39] for computing the Strict Consensus Merger, which selects the pair that maximizes the number of shared taxa between the two trees (other techniques could be used, potentially with better accuracy [15]). Thus, GreedyRFS is identical to Exact-2-RFS when the profile has only two trees.

We use a subset of the SMIDgen [37] datasets with 500 species and varying numbers of source trees (each estimated using maximum likelihood heuristics) that have been used to evaluate supertree methods in several studies [37, 39, 25, 38, 42]. See Appendix (in the full version of the paper on bioRxiv) for full details of this study.

We explored the impact of changing the number of source trees. The result for two source trees is predicted by theory (i.e., GreedyRFS is the same as Exact-2-RFS for two source trees, and so is guaranteed optimal for this case), but even when the number of source trees was greater than two, GreedyRFS dominated FastRFS in terms of criterion score, provided that the number of source trees was not too large (Fig. 5).

This establishes that the advantage in criterion score is not limited to the case of two source trees, suggesting that using Exact-2-RFS within GreedyRFS (or some other heuristics) may be useful for supertree estimation more generally.

5 Conclusions

The main contribution of this paper is Exact-2-RFS, a polynomial time algorithm for the Robinson-Foulds Supertree (RFS) of two trees that enables divide-and-conquer pipelines to be provably statistically consistent under sequence evolution models (e.g., GTR [40] and

MSC+GTR [31]). Our experimental study showed that Exact-2-RFS dominates the leading RFS heuristic, FastRFS, when used within divide-and-conquer species tree estimation using genome-scale datasets, a problem of increasing importance in biology. We also showed that a greedy heuristic using Exact-2-RFS produced better criterion scores than FastRFS when the number of source trees was small to moderate, showing the potential for Exact-2-RFS to be useful in other settings. Overall, our study advances the theoretical understanding of several important supertree problems and also provides a new method that should improve scalability of phylogeny estimation methods.

This study suggests several directions for future work. For example, although we showed that Exact-2-RFS produced better RFS criterion scores than FastRFS when used in divide-and-conquer species tree estimation (and similarly GreedyRFS was better than FastRFS for small numbers of source trees in supertree estimation), additional studies are needed to explore its performance, including additional datasets (both simulated and biological datasets) and other leading supertree methods. Similarly, other heuristics using Exact-2-RFS besides GreedyRFS should be developed and studied. Finally, our study explored accuracy rather than computational aspects; hence, a comparison between methods with respect to running time would also help inform the choice of method, especially for large datasets, and should be studied.

References

1 Alfred V. Aho, Yehoshua Sagiv, Thomas G. Szymanski, and Jeffrey D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal on Computing*, 10(3):405–421, 1981.

2 Mukul S Bansal, J Gordon Burleigh, Oliver Eulenstein, and David Fernández-Baca. Robinson-Foulds supertrees. *Algorithms for Molecular Biology*, 5(1):18, 2010.

3 Julien Baste, Christophe Paul, Ignasi Sau, and Celine Scornavacca. Efficient FPT algorithms for (strict) compatibility of unrooted phylogenetic trees. *Bulletin of Mathematical biology*, 79(4):920–938, 2017.

4 Bernard R Baum. Combining trees as a way of combining data sets for phylogenetic inference, and the desirability of combining gene trees. *Taxon*, pages 3–10, 1992.

5 Vincent Berry and François Nicolas. Maximum agreement and compatible supertrees. In *Annual Symposium on Combinatorial Pattern Matching*, pages 205–219. Springer, 2004.

6 Vincent Berry and François Nicolas. Improved parameterized complexity of the maximum agreement subtree and maximum compatible tree problems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(3):289–302, 2006.

7 Olaf RP Bininda-Emonds. *Phylogenetic supertrees: combining information to reveal the tree of life.* Springer Science & Business Media, 2004.

8 Ruchi Chaudhary, David Fernández-Baca, and John Gordon Burleigh. MulRF: a software package for phylogenetic analysis using multi-copy gene trees. *Bioinformatics*, 31(3):432–433, 2014.

9 James A Cotton and Mark Wilkinson. Majority-rule supertrees. *Systematic biology*, 56(3):445–452, 2007.

10 Leonardo De Oliveira Martins, Diego Mallo, and David Posada. A Bayesian supertree model for genome-wide species tree reconstruction. *Systematic biology*, 65(3):397–416, 2016.

11 Péter Erdős, Michael A Steel, Laszlo A Szekely, and Tandy J Warnow. Local quartet splits of a binary tree infer all quartet splits via one dyadic inference rule. *Computers and Artifical Intelligence*, 16(2):217–227, 1997.

12 Péter L Erdős, Michael A Steel, László A Székely, and Tandy J Warnow. A few logs suffice to build (almost) all trees (I). *Random Structures & Algorithms*, 14(2):153–184, 1999.

13 Péter L Erdös, Michael A Steel, László A Székely, and Tandy J Warnow. A few logs suffice to build (almost) all trees (II). *Theoretical Computer Science*, 221(1-2):77–118, 1999.

14 David Fernández-Baca, Sylvain Guillemot, Brad Shutters, and Sudheer Vakati. Fixed-parameter algorithms for finding agreement supertrees. *SIAM Journal on Computing*, 44(2):384–410, 2015.

15 Markus Fleischauer and Sebastian Böcker. Collecting reliable clades using the greedy strict consensus merger. *PeerJ*, 4:e2172, 2016.

16 Markus Fleischauer and Sebastian Böcker. Bad Clade Deletion supertrees: a fast and accurate supertree algorithm. *Molecular biology and evolution*, 34(9):2408–2421, 2017.

17 Sylvain Guillemot and Vincent Berry. Fixed-parameter tractability of the maximum agreement supertree problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(2):342–353, 2010.

18 Anne Kupczok. Split-based computation of majority-rule supertrees. *BMC evolutionary biology*, 11(1):205, 2011.

19 Wayne P Maddison. Gene trees in species trees. *Systematic Biology*, 46(3):523–536, 1997.

20 FR McMorris and Michael A Steel. The complexity of the median procedure for binary trees. In *New Approaches in Classification and Data Analysis*, pages 136–140. Springer, 1994.

21 Siavash Mirarab, Rezwana Reaz, Md S Bayzid, Théo Zimmermann, M Shel Swenson, and Tandy Warnow. ASTRAL: genome-scale coalescent-based species tree estimation. *Bioinformatics*, 30(17):i541–i548, 2014. Special issue for ECCB (European Conference on Computational Biology), 2014.

22 Siavash Mirarab and Tandy Warnow. ASTRAL-II: coalescent-based species tree estimation with many hundreds of taxa and thousands of genes. *Bioinformatics*, 31(12):i44–i52, 2015. Special issue for ISMB 2015.

23 Erin K. Molloy and Tandy Warnow. FastMulRFS: fast and accurate species tree estimation under generic gene duplication and loss models. *Bioinformatics*, 2020. To appear, special issue for ISMB 2020; preprint available at https://www.biorxiv.org/content/10.1101/835553v3.full.

24 Serita Nelesen, Kevin Liu, Li-San Wang, C Randal Linder, and Tandy Warnow. DACTAL: divide-and-conquer trees (almost) without alignments. *Bioinformatics*, 28(12):i274–i282, 2012. Special issue for ISMB 2012.

25 Nam Nguyen, Siavash Mirarab, and Tandy Warnow. MRL and SuperFine+MRL: new supertree methods. *Algorithms for Molecular Biology*, 7(1):3, 2012.

26 Huw A Ogilvie, Joseph Heled, Dong Xie, and Alexei J Drummond. Computational performance and statistical accuracy of *BEAST and comparisons with other methods. *Systematic Biology*, 65(3):381–396, 2016.

27 Roderic DM Page. Modified mincut supertrees. In *Proceedings WABI (International Workshop on Algorithms in Bioinformatics)*, pages 537–551. Springer-Verlag, 2002.

28 Cynthia Phillips and Tandy J Warnow. The asymmetric median tree—a new model for building consensus trees. *Discrete Applied Mathematics*, 71(1-3):311–335, 1996.

29 Mark A Ragan. Phylogenetic inference based on matrix representation of trees. *Molecular Phylogenetics and Evolution*, 1(1):53–58, 1992.

30 David F Robinson and Leslie R Foulds. Comparison of phylogenetic trees. *Mathematical biosciences*, 53(1-2):131–147, 1981.

31 Sebastien Roch, Michael Nute, and Tandy Warnow. Long-branch attraction in species tree estimation: Inconsistency of partitioned likelihood and topology-based summary methods. *Systematic Biology*, 68(2):281–297, September 2018. doi:10.1093/sysbio/syy061.

32 Naruya Saitou and Masatoshi Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, 4(4):406–425, 1987.

33 Charles Semple and Mike Steel. A supertree method for rooted trees. *Discrete Applied Mathematics*, 105(1-3):147–158, 2000.

34 Sagi Snir and Satish Rao. Quartets MaxCut: a divide and conquer quartets algorithm. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 7(4):704–718, 2010.

35 Alexandros Stamatakis. RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30(9):1312–1313, 2014.

36 Mike Steel and Allen Rodrigo. Maximum likelihood supertrees. *Systematic biology*, 57(2):243–250, 2008.

37 M Shel Swenson, François Barbançon, Tandy Warnow, and C Randal Linder. A simulation study comparing supertree and combined analysis methods using SMIDGen. *Algorithms for Molecular Biology*, 5(1):8, 2010.

38 M Shel Swenson, Rahul Suri, C Randal Linder, and Tandy Warnow. An experimental study of Quartets MaxCut and other supertree methods. *Algorithms for Molecular Biology*, 6(1):7, 2011.

39 M Shel Swenson, Rahul Suri, C Randal Linder, and Tandy Warnow. SuperFine: fast and accurate supertree estimation. *Systematic Biology*, 61(2):214, 2011.

40 Simon Tavaré. Some probabilistic and statistical problems in the analysis of DNA sequences. In R.M. Miura, editor, *Lectures on mathematics in the life sciences–DNA sequences*, volume 17, pages 57–86. American Mathematical Society, Providence, RI, 1986.

41 Pranjal Vachaspati and Tandy Warnow. ASTRID: accurate species trees from internode distances. *BMC genomics*, 16(10):S3, 2015.

42 Pranjal Vachaspati and Tandy Warnow. FastRFS: fast and accurate Robinson-Foulds Supertrees using constrained exact optimization. *Bioinformatics*, 33(5):631–639, 2016.

43 Tandy Warnow. Concatenation analyses in the presence of incomplete lineage sorting. *PLOS Currents Tree of Life*, 2015. doi:10.1371/currents.tol.8d41ac0f13d1abedf4c4a59f5d17b1f7.

44 Tandy Warnow. Divide-and-conquer tree estimation: Opportunities and challenges. In *Bioinformatics and Phylogenetics: Seminal contributions of Bernard Moret*, pages 121–150. Springer, 2019.

45 Mark Wilkinson and James A Cotton. Supertree methods for building the tree of life: divide-and-conquer approaches to large phylogenetic problems. In T. R. Hodkinson and J. A. N. Parnell, editors, *Reconstructing the Tree of Life: Taxonomy and Systematics of Large and Species Rich Taxa*, pages 61–75. CRC Press, 2007. Systematics Association special volume 72.

46 Mark Wilkinson, James A. Cotton, Chris Creevey, Oliver Eulenstein, Simon R. Harris, Francois-Joseph Lapointe, Claudine Levasseur, James O. McInerney, Davide Pisani, and Joseph L. Thorley. The shape of supertrees to come: Tree shape related properties of fourteen supertree methods. *Systematic Biology*, 54(3):419–431, 2005.

47 Xilin Yu. Computing Robinson-Foulds supertree for two trees. Master's thesis, University of Illinois at Urbana-Champaign, Urbana, IL, 2019. Available online at http://hdl.handle.net/2142/105698.

48 Xilin Yu, Thien Le, Sarah Christensen, Erin K Molloy, and Tandy Warnow. Advancing divide-and-conquer phylogeny estimation. *bioRxiv*, 2020. doi:10.1101/2020.05.16.099895.

49 Chao Zhang, Maryam Rabiee, Erfan Sayyari, and Siavash Mirarab. ASTRAL-III: polynomial time species tree reconstruction from partially resolved gene trees. *BMC Bioinformatics*, 19(6):153, 2018. Special issue for RECOMB-CG 2017.

Disk Compression of k-mer Sets

Amatur Rahman
Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA
aur1111@psu.edu

Rayan Chikhi
Department of Computational Biology, C3BI USR 3756 CNRS, Institut Pasteur, Paris, France
rayan.chikhi@pasteur.fr

Paul Medvedev
Pennsylvania State University, University Park, PA, USA
pzm11@psu.edu

Abstract

K-mer based methods have become prevalent in many areas of bioinformatics. In applications such as database search, they often work with large multi-terabyte-sized datasets. Storing such large datasets is a detriment to tool developers, tool users, and reproducibility efforts. General purpose compressors like gzip, or those designed for read data, are sub-optimal because they do not take into account the specific redundancy pattern in k-mer sets. In our earlier work (Rahman and Medvedev, RECOMB 2020), we presented an algorithm UST-Compress that uses a spectrum-preserving string set representation to compress a set of k-mers to disk. In this paper, we present two improved methods for disk compression of k-mer sets, called ESS-Compress and ESS-Tip-Compress. They use a more relaxed notion of string set representation to further remove redundancy from the representation of UST-Compress. We explore their behavior both theoretically and on real data. We show that they improve the compression sizes achieved by UST-Compress by up to 27 percent, across a breadth of datasets. We also derive lower bounds on how well this type of compression strategy can hope to do.

2012 ACM Subject Classification Applied computing → Computational biology

Keywords and phrases de Bruijn graphs, compression, k-mer sets, spectrum-preserving string sets

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.16

Supplementary Material Software available at `http://github.com/medvedevgroup/ESSCompress`.

Funding PM and AR were supported by NSF awards 1453527 and 1439057.
Amatur Rahman: AR is supported by NIH Computation, Bioinformatics, and Statistics training program.
Rayan Chikhi: INCEPTION project (PIA/ANR-16-CONV-0005).

1 Introduction

Many of today's bioinformatics analyses are powered by tools that are k-mer based. These tools first reduce the input sequence data, which may be of various lengths and type, to a set of short fixed length strings called k-mers. K-mer based methods are used in a broad range of applications, including genome assembly [4], metagenomics [38], genotyping [36, 14], variant calling [34], and phylogenomics [25]. They have also become the basis of a recent wave of database search tools [32, 33, 35, 15, 7, 5, 26, 13, 23], surveyed in [22]. K-mer based methods are not new, but only recently they have started to be applied to terabyte-sized datasets. For example, the dataset used to test the BIGSI database search index, which is composed of 31-mers from 450,000 microbial genomes [7], takes about 12 TB to store in compressed form.

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti; Article No. 16; pp. 16:1–16:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Storing such large datasets is a detriment to tool developers, tool users, and reproducibility efforts. For tool developers, development time is significantly increased when having to manage such large files. Due to the iterative nature of the development process, these files do not typically just sit in one place, but instead get created/moved/recreated many times. For tool users, the time it takes for the tools to write these files to disk and load them into memory is non-negligible. In addition, as we scale to even larger datasets, storage costs start to play a larger factor. Finally, for reproducibility efforts, storing and moving terabytes of data across networks can be detrimental.

To minimize these negative effects, disk compression of k-mer sets is a natural solution. By disk compression, we refer to a compressed representation that, while supporting decompression, does not support any other querying of the compressed data. Compressed representations that allow for membership queries [10] are important in their own right, but are sub-optimal when only storage is required. Most k-mer sets are currently stored on disk in one of two ways. In the situation where the set of k-mers comes from k-mer counting reads, one can simply compress the reads themselves using one of many read compression tools [17, 16, 24]. This approach requires the substantial overhead of running a k-mer counter as part of decompression, but it is often used in the absence of better options. The second approach is to gzip/bzip the output of the k-mer counter [19, 30, 21, 27, 37]. As we showed in [29], both of these approaches are space-inefficient by at least an order-of-magnitude. This is not surprising, as neither of these approaches was designed specifically for disk compression of k-mer sets.

Disk compression tailor-made for k-mer sets was first considered in our earlier work [29]. The idea was based on the concept of *spectrum-preserving string sets (SPSS)*, introduced in [9, 29, 8]. In [8], the concept of SPSS is introduced under the name *simplitigs.* A set of strings S is said to be a SPSS representation of a set of k-mers K iff 1) the set of k-mers contained in S is exactly K, 2) S does not contain duplicate k-mers, and 3) each string in S is of length $\geq k$. The weight of an SPSS is the number of characters it contains. For example, if $K = \{ACG, CGT, CGA\}$, then $\{ACGT, CGA\}$ would be an SPSS of weight 7; also K itself would be an SPSS of K of weight 9. On the other hand, $\{CGACGT\}$ is not an SPSS, because it contains $GAC \notin K$. Intuitively, a low weight SPSS can be constructed by gluing together k-mers in K, with each glue operation reducing the weight by $k-1$. In [29], we proposed the following simple compression strategy, called UST-Compress. We find a low-weight SPSS S, using a greedy algorithm called UST, and compress S to disk using a generic nucleotide compression algorithm (e.g. MFC [28]). UST-Compress achieved significantly better compression sizes than the two approaches mentioned above.

UST-Compress was not designed to be the best possible disk compression algorithm but only to demonstrate one of the possible applications of the SPSS concept. When the goal is specifically disk compression, we are no longer bound to store a set of strings with exactly the same k-mers as K, as long as a decompression algorithm can correctly recover K. The main idea of this paper is to replace the SPSS with a more relaxed string set representation, over the alphabet $\{A, C, G, T, [,], +, -\}$. Our approach is loosely inspired by the notion of elastic-degenerate strings [18]. It attempts to remove even more duplicate $(k-1)$-mers from the representation than SPSS does, using the extra alphabet characters as placeholders for nearby repetitive $(k-1)$-mers. For the above example, our representation would be $ACG[+A]T$, where the " + " is interpreted as a placeholder for the $k-1$ characters before the open bracket (i.e. CG). After replacing the " + ", we get $ACG[CGA]T$ and we split the string by cleaving out the substring within brackets; i.e., we get $ACGT$ and CGA.

Based on this idea, we present two algorithms for the disk compression of k-mer sets, ESS-Compress and ESS-Tip-Compress. We explore the behavior of these algorithms both theoretically and on real data. We give a lower bound on how well this type of algorithm can compress. We show that they improve the compression sizes achieved by UST-Compress by 10-27% across a breadth of datasets. The two algorithms present a trade-off between time/memory and compression size, which we explore in our results. The two algorithms are freely available open source tools on `http://github.com/medvedevgroup/ESSCompress`.

2 Preliminaries

2.1 Basic definitions

Strings: The *length* of string x is denoted by $|x|$. A string of length k is called a *k-mer*. We assume k-mers are over the DNA alphabet. A string over the alphabet $\{A, C, G, T, [,], +, -\}$ is said to be *enriched*. We use $\cdot$ as the string concatenation operator. For a set of strings S, $weight(S) = \sum_{x \in S} |x|$ denotes the total count of characters. We define $suf_k(x)$ (respectively, $pre_k(x)$) to be the last (respectively, first) k characters of x. We define $cutPre_k(x) = suf_{|x|-k}(x)$ as x with the prefix removed. When the subscript is omitted from pre, suf, and $cutPre$, we assume it is $k-1$. A string x is *canonical* if it is the lexicographically smaller of x and its reverse complement.

For x and y with $suf(x) = pre(y)$, we define *gluing* x and y as $x \odot y = x \cdot cutPre(y)$. For $s \in \{0, 1\}$, we define $orient(x, s)$ to be x if $s = 0$ and to be the reverse complement of x if $s = 1$. We say that x_0 and x_1 have a *(s_0, s_1)-oriented-overlap* if $suf(orient(x_0, 1 - s_0)) = pre(orient(x_1, s_1))$. Intuitively, such an overlap exists between two strings if we can orient them in such a way that they are glueable. For example, AAC and TTG have a $(0, 0)$-oriented overlap.

Bidirected de Bruijn graphs: A *bidirected graph* G is a pair (V, E) where the set V are called vertices and E is a set of edges. An edge e is a set of two pairs, $\{(u_0, s_0), (u_1, s_1)\}$, where $u_i \in V$ and $s_i \in \{0, 1\}$, for $i \in \{0, 1\}$. Note that this differs from the notion of an edge in an undirected graph, where $E \subseteq V \times V$. Intuitively, every vertex has two sides, and an edge connects to a side of a vertex (see Figure 1 for examples). An edge is a *loop* if $u_0 = u_1$. Given a non-loop edge e that is incident to a vertex u, we denote $side(u, e)$ as the side of u to which it is incident. We say that a vertex u is a *dead-end* if it has exactly one side to which no edges are incident. A *bidirected DNA graph* is a bidirected graph G where every vertex u has a string label $lab(u)$, and for every edge $e = \{(u_0, s_0), (u_1, s_1)\}$, there is a (s_0, s_1)-oriented-overlap between $lab(u_0)$ and $lab(u_1)$ (see Figure 1 for examples). G is said to be *overlap-closed* if there is an edge for every such overlap. Let K be a set of canonical k-mers. The node-centric *bidirected de Bruijn graph*, denoted by $dBG(K)$, is the overlap-closed bidirected DNA graph where the vertices and their labels correspond to K. In this paper, we will assume that $dBG(K)$ is not just a single cycle; such a case is easy to handle in practice but is a space-consuming corner-case in all the analyses.

Paths and spellings: A sequence $p = (u_0, e_1, u_1, \ldots, e_n, u_n)$ is a *path* iff 1) for all $1 \le i \le n$, e_i is incident to u_{i-1} and to u_i, 2) for all $1 \le i \le n-1$, $side(u_i, e_i) = 1 - side(u_i, e_{i+1})$, and 3) all the u_is are different. A path can also be any single vertex. Vertices $u_1, \ldots, u_{n-1}$ are called *internal* and u_0 and u_n are called *endpoints*. We call u_0 to be the *initiator* vertex of p. We say that p is *normalized* if for every e_i, $side(u_{i-1}, e_i) = 1$ and $side(u_i, e_i) = 0$; intuitively, the path uses edges like in a directed graph. The *spelling* of a normalized path p is defined as $spell(p) = lab(u_0) \odot \cdots \odot lab(u_n)$. If P is a set of normalized paths, then $spell(P) = \bigcup_{p \in P} spell(p)$.

Unitigs and the compacted de Bruijn graph: A path in $dbG(K)$ is a *unitig* if all its vertices have in- and out-degrees of 1, except that the first vertex can have any in-degree and the last vertex can have any out-degree. A single vertex is also a unitig. A unitig is *maximal* if it is not a sub-path of another unitig. It was shown in [12] that if $dBG(K)$ is not a cycle, then the set of maximal unitigs forms a unique decomposition of the vertices in $dBG(K)$ into vertex-disjoint paths. The bidirected *compacted de Bruijn graph* of K, denoted by $cdBG(K)$, is the overlap-closed bidirected DNA graph where the vertices are the maximal unitigs of $dBG(K)$, and the labels of the vertices are the spellings of the unitigs. In practice, this graph can be efficiently constructed from K using the BCALM2 tool [12, 11].

Spanning out-forest: Given a directed graph D, an *out-tree* is a subgraph in which every vertex except one, called the root, has in-degree one, and, when the directions on the edges are ignored, is a tree. An *out-forest* is a collection of vertex-disjoint out-trees. An out-forest is *spanning* if it covers all the vertices of D.

2.2 Path covers and UST-Compress

A *vertex-disjoint normalized path cover* Ψ of $cdBG(K)$ is a set of normalized paths such that every vertex is in exactly one path and no path visits a vertex more than once; we will sometimes use the shorter term *path cover* to mean the same thing. There is a close relationship between SPSS representations of K and path covers, shown in [29]. In particular, a path cover Ψ induces the SPSS $spell(\Psi)$. An example of a path cover is one where every vertex of $cdBG(K)$ is in its own path, and the corresponding SPSS is the set of all maximal unitig sequences. Figures 1 and 2 show examples of path covers. The number of paths in Ψ (denoted as $|\Psi|$) and the weight of the induced SPSS is closely related:

$$weight(spell(\Psi)) = |K| + |\Psi|(k-1) \tag{1}$$

This relationship also translates to the number of edges in Ψ; by its definition, the number of edges in Ψ is simply the number of vertices in $cdBG(K)$ minus $|\Psi|$.

The idea of our previous algorithm UST-Compress [29] is to find a path cover Ψ_{UST} with as many edges as possible. Having more edges reduces the number of paths, which in turn reduces the weight of the corresponding SPSS and the size of the final compressed output. We can understand this intuitively as follows. Edges in $cdBG(K)$ connect unitigs whose endpoints have the same $(k-1)$-mer (after accounting for reverse complements). For every edge we add to our path cover, we glue these two unitigs and remove one duplicate instance of the $(k-1)$-mer from the corresponding SPSS. Note however that Ψ_{UST} does not remove all duplicate $(k-1)$-mers from the SPSS, because Ψ can only have two edges incident on a vertex, one from each side, and hence a unitig can only be glued at most twice. If a unitig has edges to more than two other unitigs, then some of the adjacent unitigs would include the duplicate $(k-1)$-mer in the SPSS. The idea of our paper is to exploit the redundancy due to those remaining edges an thus further reduce the size of the representation.

3 ESS-Compress

3.1 Main algorithm

Our starting point is a set of canonical k-mers K, the graph $cdBG(K)$, and a vertex-disjoint normalized path cover Ψ of $cdBG(K)$ returned by UST.[1] To develop the intuition for our

[1] Though we did not explain it in [29], UST always returns normalized paths. It flips any vertex that is in the wrong orientation on its path, by reverse complementing its label, without affecting anything else.

algorithm, we first consider a simple example (Figure 1A). In this example, we see a vertex-disjoint path cover Ψ composed of two paths, ψ^p and ψ^c. There is an edge between an internal vertex (=unitig [2]) u^p of ψ^p and the initiator vertex u^c of ψ^c. Such an edge is an example of an absorption edge. ESS-Compress constructs an enriched string representation of K, as shown in the figure. The basic idea is that u^p and u^c share a common $(k-1)$-mer (i.e. GT). We can cut out this common portion from the string representing u^c and replace it with a special marker character "+". We can then include u^c inside of the representation of u^p by surrounding u^c with brackets. The marker character " + " is a placeholder for the $k-1$ nucleotides right before the opening bracket. To decompress the enriched string, we first replace the marker to get $TCGT[GTAA]T$ and then cleave out the bracketed string to get $\{TCGTT, GTAA\}$. This exactly recovers the SPSS representation of ψ^p and ψ^c.

Formally, we say that an edge in $cdBG(K)$ is an *absorption edge* iff 1) it connects two unitigs u^p and u^c, on two distinct paths ψ^p and ψ^c, respectively, 2) u^p is an internal vertex, and 3) u^c is an initiator vertex. We refer to u^p and ψ^p as *parents* and u^c and ψ^c as *children*; we also say that ψ^p and u^p absorb ψ^c and u^c. [3]

Figure 1B-D shows the other cases, corresponding to the possible orientation of the absorption edge. The logic is the same, but we need to introduce a second marker character " $-$ " that is a placeholder for the reverse complement of the last $k-1$ characters right before the opening bracket. In each of these cases, we add 3 extra characters (two brackets and one marker) and remove $k-1$ nucleotide characters.

Next, observe that a single parent path can absorb multiple children paths, as illustrated in Figure 2A. Also, observe that a single parent unitig can absorb more than one child path, as shown in Figure 2B. As in the previous example, we save $k-1-3=k-4$ characters for every absorbed edge.

These absorptions can be recursively combined, as shown in Figure 2C. Because we require a parent unitig to be an internal vertex and a child unitig to be an initiator vertex, the same unitig cannot be both parent and child. Therefore, ESS-Compress can construct a representation recursively, without any conflicts. The recursion tree is reflected in the nesting structure of the brackets in the enriched string.

However, we must be careful to avoid cycles in the recursion. We define the *absorption digraph* D_A as the directed graph whose vertex set is the set of paths Ψ and an edge $(\psi^p \rightarrow \psi^c)$ if ψ^p absorbs ψ^c. For every edge in D_A, we also associate the corresponding bidirected edge between u^p and u^c in $cdBG(K)$. We would like to select a subset of edges F along which to perform absorptions, so as to avoid cycles in D_A and to make sure a path cannot be absorbed by more than one other path. We would also try to choose as many edges as possible, since each absorption saves $k-4$ characters. To achieve these goals, ESS-Compress defines F as a spanning out-forest in D_A with the maximum number of edges. We postpone the algorithm to find F to Section 3.2.

The high-level pseudo-code of ESS-Compress is shown in Algorithm 1 and illustrated in Figure 3. The recursive algorithm to create the enriched representation using F as a guide is shown in Algorithm 2. It follows the intuition we just developed. It starts from the paths that will not be absorbed (i.e. the roots in F). For a path ψ^p, it first computes the enriched representations of all the child paths (Lines 3 to 9). It then integrates them into

[2] Note that the vertices of this graph (i.e. $cdBG(K)$) correspond to maximal unitigs in the non-compacted graph (i.e. $dBG(K)$). We will generally use "vertex" and "unitig" interchangeably, to refer to a vertex in $cdBG(K)$. We never use "unitig" to refer to a type of path in $cdBG(K)$.

[3] In our code, we actually allow a slightly broader definition of absorption. In particular, we also allow an edge to be absorbing if u^p is an initiator and $s^p = 1$, or if u^p is an initiator and $|lab(u^p)| \geq 2k-2$. For the sake of simplicity, we do not consider this edge case in the paper.

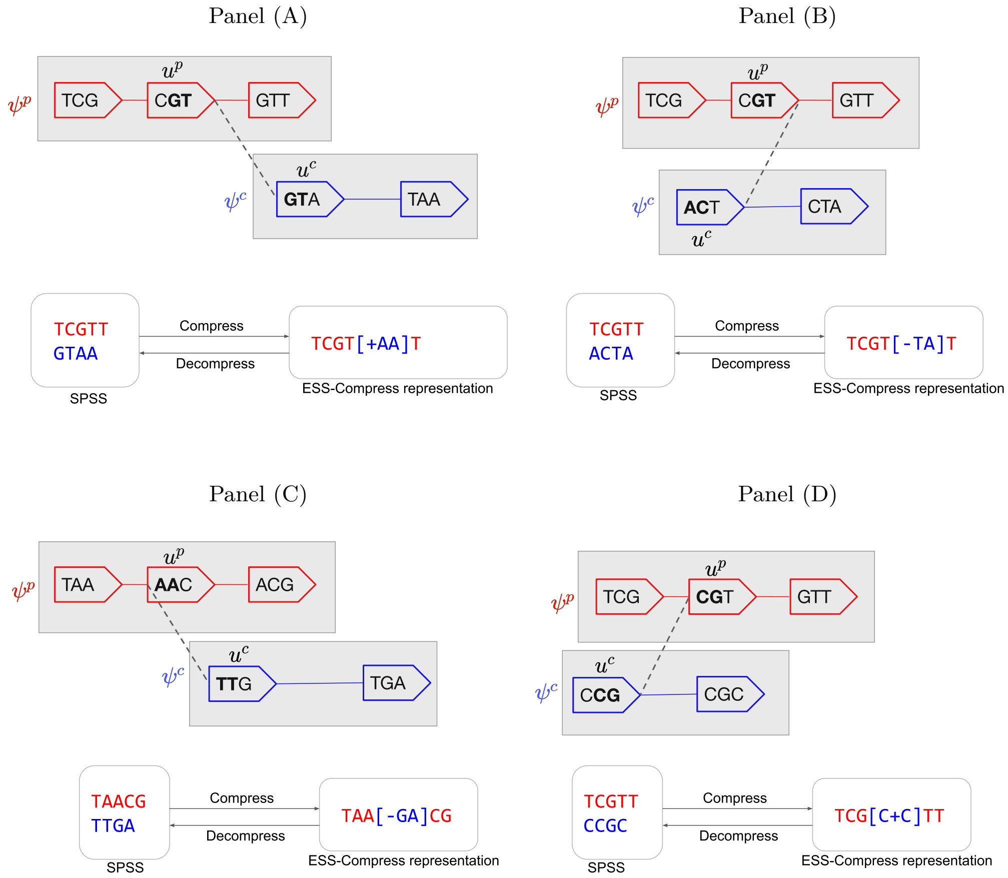


Figure 1 Examples of the four types of absorption. Each panel shows the edges along two paths: ψ^p (red vertices inside a shaded rectangle) and ψ^c (blue vertices inside a shaded rectangle) and an absorption edge $e = \{(u^p, s^p), (u^c, s^c)\}$ (dashed line) between the parent unitig u^p and the child unitig u^c. The graph being shown in each panel is $cdBG(K)$, but only the absorption edge and the edges of ψ^p and ψ^c are shown. In this simple example, the unitigs of $dBG(K)$ are just paths made of single vertices, and hence the vertices of $cdBG(K)$ have labels of length $k = 3$. Each vertex is shown as a pointed rectangle with its label inside; we use the convention that the "zero" side of a vertex is the flat side on the left, and the "one" side is the pointy side on the right. At the bottom left of each panel, we show the spectrum-preserving string set (SPSS) $spell(\{\psi^p, \psi^c\})$. At the bottom right, we show the enriched representation generated by our algorithm. Depending on the value of s^p and s^c, four different cases can arise. When $s^p = 1, s^c = 0$ (shown in (A)), $pre(lab(u^c))$ is replaced with marker "+", as it is same as $suf(lab(u^p))$. When $s^p = 1, s^c = 1$ (shown in (B)), $pre(lab(u^c))$ is replaced by "$-$", as it is same as the reverse complement of $suf(lab(u^p))$. When $s^p = 0, s^c = 0$ (shown in (C)), $pre(lab(u^c))$ is replaced with "$-$", as it is the same as the reverse complement of $pre(lab(u^p))$. When $s^p = 0, s^c = 1$ (shown in (D)), $suf(lab(u^c))$ is replaced with "+", as it is the same as $pre(lab(u^p))$.

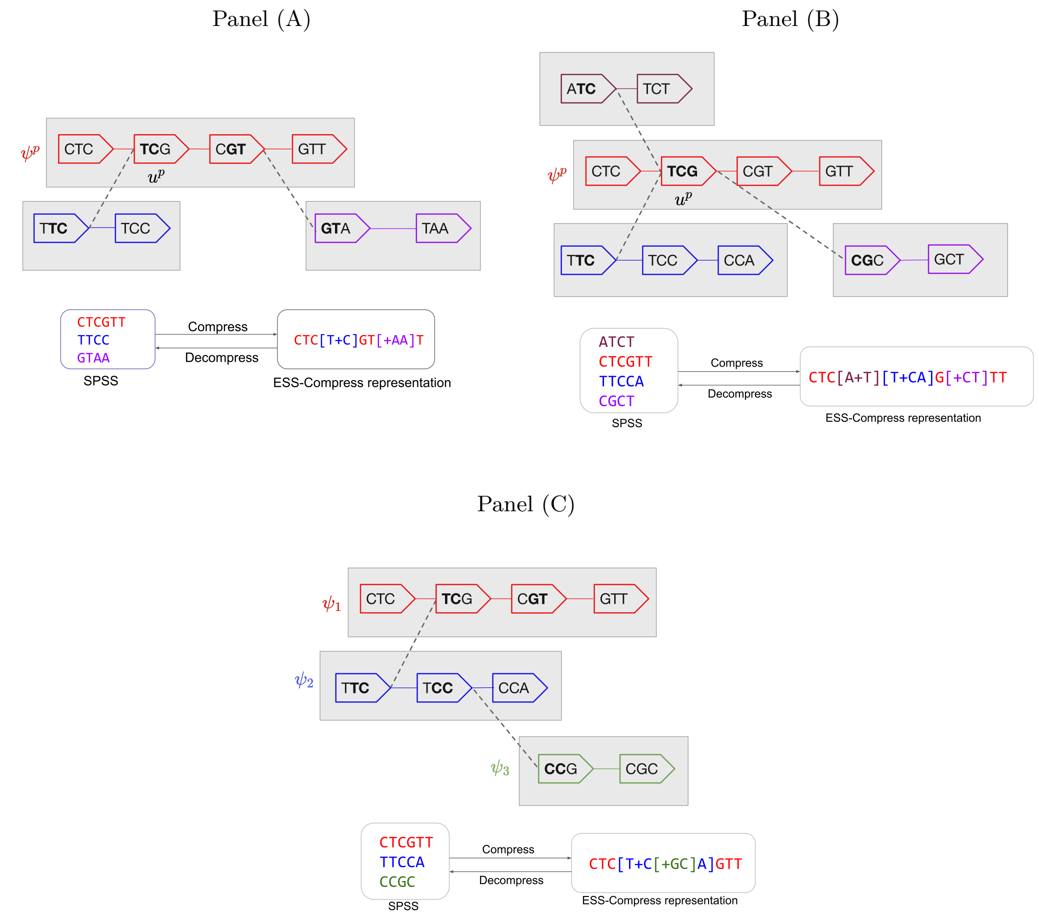


Figure 2 More complex absorption examples. In (A), one path absorbs multiple paths. In (B), one unitig u^p absorbs multiple paths. In (C), one path (ψ_1) absorbs another (ψ_2) which itself absorbs another (ψ_3). This is a recursive absorption, showing how a path can be both a child and a parent.

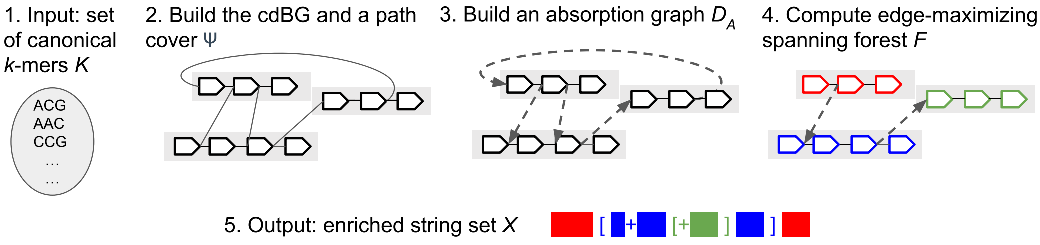

1. Input: set of canonical k-mers K
ACG
AAC
CCG
…
…
2. Build the cdBG and a path cover Ψ
3. Build an absorption graph D_A
4. Compute edge-maximizing spanning forest F
5. Output: enriched string set X
[+ [+]]

Figure 3 Visual overview of the steps in Algorithm 1.

Algorithm 1 ESS-Compress (K)
Input: a set of canonical k-mers K
Output: a set of enriched strings X.

```
Construct $cdBG(K)$
Run UST to get a path cover $\Psi$
Run DFS algorithm to get $F$, a spanning out-forest of the absorption graph $D_A$
$X \leftarrow \emptyset$
for each path $\psi$ which is a root in $F$ do
    add Spell-Path-Enrich ($\psi$, null) to $X$
end for
return $X$
```

the appropriate locations of $spell(\psi^p)$ (Lines 10 to 14). It then uses a marker to replace the redundant sequence in the spelling of ψ^p, with respect to ψ^p's own parent (Lines 17 to 31). To decide which marker to use, it receives as a parameter the absorption edge e_D that was used to absorb ψ^p.

Decompression is done by a recursive algorithm DEC that takes as input an enriched string x and a $(k-1)$-mer called *markerReplacement*. Initially, DEC is called independently on every enriched string $x \in$ ESS-Compress(K), with $markerReplacement = null$. We call the characters of x which are not enclosed within brackets *outer*. The brackets themselves are not considered outer characters. DEC first replaces any occurrence of an outer " + " (respectively, " − ") with *markerReplacement* (respectively, the reverse complement of *markerReplacement*). It then outputs all the outer characters as a single string. Then, for every top-level open/close bracket pair in x, it calls DEC recursively on the sequence in between the brackets, and passes as *markerReplacement* the rightmost $k-1$ outer characters to the left of the open bracket.

3.2 Algorithm to choose absorption edges

Let D be any directed graph and consider the problem of finding a spanning out-forest with the maximum number of edges. We call this the problem of finding an *edge-maximizing spanning out-forest*. This problem is a specific instance of the maximum weight out-forest problem [3], which allows for weights to be placed on the edges. As we show in this section, there is an optimal algorithm for our problem that is simpler than the algorithm for arbitrary weights described in [3].

Our algorithm first decomposes D into strongly connected components, and builds $SC(D)$, the strongly connected component digraph of D. In $SC(D)$, the vertices are the strongly connected components of D, and there is an edge from component c_1 to c_2 if there is an edge in D from some vertex in c_1 to some vertex in c_2. For every component that is a source in $SC(D)$, we pick an arbitrary vertex from it (in D) and put it into a "starter" set. Then, we perform a depth-first search (DFS) traversal of D, but whenever we start a new tree, we initiate it with a vertex from the starter set, if one is available. We remove the vertex from the starter set once it is used to initiate a tree. We then output the DFS forest F.

We will prove that F is a spanning out-forest of D with the maximum number of edges.

▶ **Lemma 1** (Correctness of edge-maximizing spanning out-forest algorithm)**.** *Let D be a directed graph, let F be the spanning out-forest returned by our algorithm run on D, and let n_{sc} be the number of source components in $SC(D)$. Then, the number of out-trees in F is n_{sc} and this is the smallest possible for any spanning out-forest. Also, the number of edges in F is the maximum possible for any spanning out-forest.*

Algorithm 2 Spell-Path-Enrich(ψ,e_D)

Input: a path ψ corresponding to the sequence of unitigs $u_0, \ldots, u_n$. If ψ is itself absorbed, then the absorption edge e_D.

Output: an enriched string representation of ψ and all its descendent paths in F.

```
for $i = 0$ to $n$ do                                        ▷ for each unitig in $\psi$
    Use $u^p$ to denote the $i^{\text{th}}$ unitig of $\psi$.
    $ins_0$ = “”                        ▷ absorbed enriched strings to insert at the end
    $ins_1$ = “”                        ▷ absorbed enriched strings to insert after prefix
    for each unitig $u^c$ absorbed by $u^p$ in $F$ do
        Let $e = \{(u^p, s^p), (u^c, s^c)\}$ be the corresponding absorption edge in $cdBG(K)$
        Let $\psi^c \in \Psi$ be the path containing $u^c$.
        $ins_{s^p} \leftarrow ins_{s^p} \cdot$ Spell-Path-Enrich($\psi^c, e$)
    end for
    if $i = 0$ then                                  ▷ if $u^p$ is the first unitig in $\psi$
        $enrichedStr[i] \leftarrow pre(lab(u^p)) \cdot ins_0 \cdot cutPre(lab(u^p)) \cdot ins_1$
    else
        $enrichedStr[i] \leftarrow ins_0 \cdot cutPre(lab(u^p)) \cdot ins_1$
    end if
end for
$x \leftarrow$ concatenate $enrichedStr[i]$, in increasing order of $i$
if $e_D \neq null$ then                                   ▷ if $\psi$ is not a root in $F$
    /* Perform marker replacement, following Figure 1 */
    Let $\{(u^p, s^p), (u^c, s^c)\} = e_D$
    if ($s^p$ xor $s^c$) $= 1$ then
        $marker$ = “ + ”
    else
        $marker$ = “ − ”
    end if
    if $s^c = 1$ then
        In $x$, replace $suf(lab(u^c))$ with $marker$
    else
        In $x$, replace $pre(lab(u^c))$ with $marker$
    end if
    $x \leftarrow$ “[” $\cdot\, x \,\cdot$ “]”
end if
return $x$
```

Proof. Consider any spanning out-forest of D. If it has less than n_{sc} out-trees, then by the pigeonhole principle, there are two source components c_1 and c_2 with vertices v_1 and v_2, respectively, belonging to the same out-tree. This is a contradiction, since c_1 and c_2 are source components and hence there cannot be a path between them. Hence, any spanning out-forest must have at least n_{sc} out-trees. Now, consider F. Every vertex in D is reachable from one of the vertices in the starter set, by its construction. There are n_{sc} starter vertices, so F will have at most n_{sc} out-trees. Since any spanning out-forest must have at least n_{sc} out-trees, F will have n_{sc} out-trees and it will be the minimum achievable. Also, in any spanning out-forest, the number of edges is the number of vertices minus the number of out-trees; hence F will have the the maximum number of edges of any spanning out-forest. ◀

3.3 The weight of the ESS-Compress representation

In this section, we derive a formula for the weight of the ESS-Compress representation and explore the potential benefits of some variations of ESS-Compress.

▶ **Theorem 2.** *Let K be a set of canonical k-mers, and let Ψ be a vertex-disjoint normalized path cover of $cdBG(K)$ that is used by ESS-Compress(K). Let n_{sc} be the number of sources in the strongly connected component graph of the absorption graph D_A. Let X be the solution returned by ESS-Compress(K). Then*

$$weight(X) = |K| + 3|\Psi| + n_{sc}(k-4)$$

Proof. If we unroll the recursion of ESS-Compress, then there are exactly $|\Psi|$ runs of Spell-Path-Enrich, one for each $\psi \in \Psi$. For each call, we let n_ψ be the number of characters in the returned string that are added non-recursively (i.e. everything except ins_0 and ins_1). Considering the structure of the recursion and accounting for characters in this way, we have that $weight(X) = \sum_{\psi\in\Psi} n_\psi$.

Prior to marker replacement (Line 17, the non-recursive part of x is $spell(\psi)$). When ψ is a root in the absorption forest F, then the marker absorption stage is not executed and so $n_\psi = |spell(\psi)|$. Otherwise, the marker absorption phase (Lines 17 to 31) removes $k-1$ characters, adds 1 new marker character, and adds two new bracket characters. Hence, $n_\psi = |spell(\psi)| - (k-1) + 3 = |spell(\psi)| - (k-4)$. By Lem. 1, F contains n_{sc} roots. Hence,

$$\begin{aligned} weight(X) &= \sum_{\psi\in\Psi} n_\psi = \sum_{\psi \text{ is a root}} |spell(\psi)| + \sum_{\psi \text{ is not a root}} |spell(\psi)| - (k-4) \\ &= \sum_{\psi\in\Psi} |spell(\psi)| - (k-4)(|\Psi| - n_{sc}) = |K| + 3|\Psi| - n_{sc}(k-4) \end{aligned}$$

The last equality follows by applying Equation (1) from Section 2. ◀

We can use Thm. 2 to better understand ESS-Compress. The weight depends on the choice of Ψ. The Ψ returned by UST has, empirically, almost the minimum $|\Psi|$ possible [29]. This (almost) minimizes the $3|\Psi|$ term in Thm. 2. However, this may not necessarily lead to the lowest total weight, because there is an interplay between Ψ and n_{sc}, as follows. Let Ψ' be a vertex-disjoint normalized path cover with $|\Psi'| > |\Psi|$. Its paths are shorter, on average, than Ψ's. There may now be edges of $cdBG(K)$ that become absorption edges, that were not with Ψ. For example, an edge between two unitigs which are internal in Ψ is not, by our definition, an absorption edge. With the shorter paths in Ψ', one of these unitigs may become an initiator vertex, making the edge absorbing. This may in turn improve connectivity in D_A and decrease n_{sc}, counterbalancing the increase in $|\Psi'|$. Nevertheless, ESS-Compress does not consider alternative path covers and always uses the one returned by UST.

Another aspect of ESS-Compress that could be changed is the definition of absorption edge. We restrict absorption edges to be between an initiator unitig and an internal unitig; however, one could in principle also define ways to absorb between an endpoint unitig and an internal unitig, or between two internal unitigs. This could potentially decrease n_{sc} by increasing the number of absorption edges, though it would likely need more complicated and space-consuming encoding schemes.

How much could be gained by modifying the path cover and the absorption rules that ESS-Compress uses? We can answer this by observing that n_{sc} cannot be less than C, the number of connected components of the undirected graph underlying $cdBG(K)$. At the same time, in [29] we gave an algorithm to compute an instance-specific lower bound β on the number of paths in any vertex-disjoint path cover. Putting this together, we conclude that regardless of which path cover is used and which subset of $cdBG(K)$ edges are allowed to be absorbing, the weight of a ESS-Compress representation cannot be lower than:

$$|K| + 3\beta + C(k-4) \tag{2}$$

As we will see in the results, the weight of ESS-Compress is never more than 2% higher than this lower bound, which is why we did not pursue these other possible optimizations to ESS-Compress. We note, however, that the above is not a general lower bound and does not rule out the possibility of lower-weight string set representations that beat ESS-Compress.

4 ESS-Tip-Compress: a simpler alternative

ESS-Compress is designed to achieve a low compression size but can require a large memory stack due to its recursive structure. The memory during compression and decompression is proportional to the depth of this stack, which is the depth of the out-forest F. If F were to be more shallow, then the memory would be reduced. In this section, we describe ESS-Tip-Compress, a simpler, faster, and lower-memory technique that can be used when compression speed/memory are prioritized. It is centered on dead-end vertices in the compacted graph, which usually correspond to tips in the uncompacted dBG and are typically due to sequencing errors, endpoints of transcripts, or coverage gaps. ESS-Tip-Compress is based on the observation that a large chunk of the graph is dead-end vertices (at least for sequencing data), and limiting absorption to only them can yield much of the benefits of a more sophisticated algorithm.

First, we find a vertex-disjoint normalized path cover Ψ that is forced to have each dead-end vertex in its own dedicated path (i.e. its path only contains the vertex itself). This can be done easily by running UST on the graph obtained from $cdBG(K)$ by removing all dead-end vertices. Next, we select the absorption forest F as follows. For each dead-end vertex v, we identify a non-dead-end vertex u which is connected to v via an edge e. In the rare case that such a u does not exist, we skip v. Otherwise, we add $(u \to v)$ to F. We can assume without loss of generality that $side(u, e) = 1 - side(v, e)$ because if that is not the case, than we can replace $lab(v)$ by its reverse complement and thereby change the side to which e is incident. For any paths that remain uncovered by F, we add them as roots of their own tree. Finally, we run a slightly modified version of Spell-Path-Enrich, using this Ψ and this F.

We modify Spell-Path-Enrich as follows. First, observe that F has max depth of 2 vertices. Hence, the parenthesis generated by Spell-Path-Enrich are never nested. Second, observe that the marker value is always " + ", because $side(u, e) = 1 - side(v, e)$ for all absorption edges in F. These observations allow us to reduce the number of extra characters we need for each absorption down to 2, instead of 3 (we omit the implementation details).

5 Empirical Results

We evaluated our methods on one small bacterial dataset, two metagenomic datasets from NIH human microbiome project, and RNA-seq reads from both human and plant (Table 1). To obtain the set of k-mers K from these datasets, we ran the DSK k-mer-counter [30] with $k = 31$ and filtered out low-frequency k-mers (<5 for whole human and <2 for the other datasets). We then constructed $cdBG(K)$ using BCALM2. The last three columns in Table 1 show the properties of the graph: number of vertices, number of dead-end vertices and total percentage of isolated vertices. We ran all our experiments single-threaded on

Table 1 Dataset characteristics.

Dataset	Source	Read Length (bp)	# reads	# distinct 31-mers	# unitigs	% dead-end unitigs	% isolated unitigs
R. sphaeroides	GAGE [31]	101	2,050,868	5,908,467	442,681	47%	8%
Human RNA-seq	SRR957915	101	49,459,840	101,017,526	7,665,682	40%	13%
Gingiva metagenome	SRS014473	101	55,419,548	101,872,420	5,678,516	36%	15%
Soybean RNA-seq	SRR11458718	125	83,594,116	111,206,789	3,659,969	28%	12%
Tongue metagenome	SRS011086	101	81,664,789	165,159,726	11,358,233	37%	11%
Whole human	ERR174310	101	207,579,467	2,319,022,432	51,094,913	14%	18%

a server with an Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.10GHz processor with 64 cores and 512 GB of memory. We used `/usr/bin/time` to measure time and memory. Detailed steps to reproduce our experiments are available at `https://github.com/medvedevgroup/ESSCompress/tree/master/experiments`.

The output of our tools was compressed with MFC. Note that MFC is not optimized for non-nucleotide characters, but such characters are rare in our string sets (< 0.1 bits per k-mer). We compared our tools against four other approaches. The first is UST-Compress, which we showed in our previous work to outperform other disk compressors [29]. The second is to strip the read FASTA files of all non-sequence information and compress them using MFC. The third is to simply write one distinct k-mer per line to a file and compress it using MFC. The fourth is the BOSS method, as implemented in [1]. BOSS is a succinct implementation of a de Bruijn graph [6]. Though it is designed to answer membership queries, it also achieved the closest compression size to UST-Compress in our previous study [29]. As in [29], we compressed BOSS's binary output using LZMA. We confirmed the correctness of all evaluated tools, including our own, on the datasets.

We did not explore the possibility of replacing UST in our pipeline with ProphAsm [2]. ProphAsm is an alternative algorithm to compute an SPSS called simplitigs, but we showed in [29] that the UST SPSS representation is nearly optimal, with only 2-3% difference to the lower bound of weight. Since ProphAsm computes the same kind of representation, it is impossible for it to improve result beyond 2-3%. We also did not compare against other k-mer membership data structures because in our previous paper [29], we showed that UST-Compress and BOSS achieve a better compression ratio on the tested datasets.

String set properties

We first measure the weights and sizes of our ESS-Compress and ESS-Tip-Compress, shown in Table 2. ESS-Compress uses 13-42% less characters than UST. ESS-Tip-Compress was worse than ESS-Compress (6-13% larger), but still better than UST-Compress (3-38% smaller). The lower bound computed by Eq. 2 is very close to the weight of ESS-Compress (within 1.7%, Table 2), indicating that the alternate strategies explored in Section 3.3 would not be useful on these datasets.

Compression size

Table 3 shows the final compression sizes, after the string sets are compressed with MFC. ESS-Compress outperforms the second best tool (which is usually UST-Compress) by 4-27%. It outperforms the naive strategies (i.e. read FASTA or one k-mer per line) by an order-of-magnitude. Interestingly, it outperforms ESS-Tip-Compress by only 1-8%; this can be attributed to the large number of dead-end vertices (Table 1).

Table 2 The weights and sizes of various string set representations. The rightmost column shows the lower bound computed by Equation (2) in Section 3.3. The weight of ESS-Compress was verified to be the same as predicted by Theorem 2.

	UST		ESS-Tip-Compress		ESS-Compress		Eq. 2 lower bound
Dataset	# strings	#char/ k-mer	# strings	#char/ k-mer	# strings	#char/ k-mer	#char/ k-mer
R. sphaeroides	240,562	2.22	61,909	1.38	36,456	1.29	1.28
Human RNA-seq	4,098,389	2.22	1,834,945	1.60	1,098,938	1.42	1.39
Gingiva metagenome	3,095,476	1.91	1,499,270	1.48	917,388	1.33	1.32
Soybean RNA-seq	1,806,078	1.49	1,137,350	1.32	515,244	1.17	1.17
Tongue metagenome	6,030,814	2.10	2,664,422	1.53	1,327,701	1.33	1.32
Whole human	22,072,219	1.32	21,320,263	1.28	10,321,275	1.15	1.14

Table 3 The compression sizes, as measured in bits per k-mer in the compressed output. All string representations (i.e. not BOSS) are compressed using MFC in the final step. Since BOSS is a binary representation, we use LZMA for the final compression step.

Dataset	Read FASTA	One k-mer per line	BOSS	UST-Compress	ESS-Tip-Compress	ESS-Compress
R. sphaeroides	45.4	28.4	6.55	3.93	2.90	2.87
Human RNA-seq	45.8	31.7	6.89	4.14	3.43	3.33
Gingiva metagenome	48.0	32.4	10.64	3.76	3.22	3.05
Soybean RNA-seq	43.0	33.1	5.97	2.83	2.66	2.55
Tongue metagenome	48.1	33.3	3.59	4.07	3.32	3.07
Whole human	31.9	48.2	4.65	2.49	2.46	2.40

We observe that our improvement in weight (Table 2) does not directly translate to improvement after compression with MFC (Table 3). For ESS-Compress, the average improvement in weight over UST is 30% but the improvement in bits is 17%. We attribute this to the fact that MFC works by exploiting redundant regions, based on their context. Thus, the redundant sequence that ESS-Compress removes is likely the sequence that was more compressible by MFC and hence MFC looses some of its effectiveness.

We also verified that ESS-Compress can successfully compress datasets of varying k-mer sizes (between 21 and 71) and low-frequency thresholds (2,3, and 4). Figure 4 shows compressed sizes of human RNA-seq data in bits/k-mer as well as their weights compared to the lower bounds. The weight of ESS-Compress closely matches the lower bound across all parameters ($<$ 2.4% gap), but the weight and compression size increase for larger k and lower thresholds.

Decompression and compression time and memory

The cost of decompression is important since it is incurred every time the dataset is used for analysis. For both ESS-Compress and ESS-Tip-Compress, the decompression memory is $<$ 1 GB (Table 5) the time is $<$ 10 minutes for the large whole human dataset and $<$ 1.5 minutes for the other datasets (Table 4). Both of these are dominated by the MFC portion.

Compression is typically done only once, but the time and memory use can still be important in some applications. Tables 5 and 6 show the compression time and memory

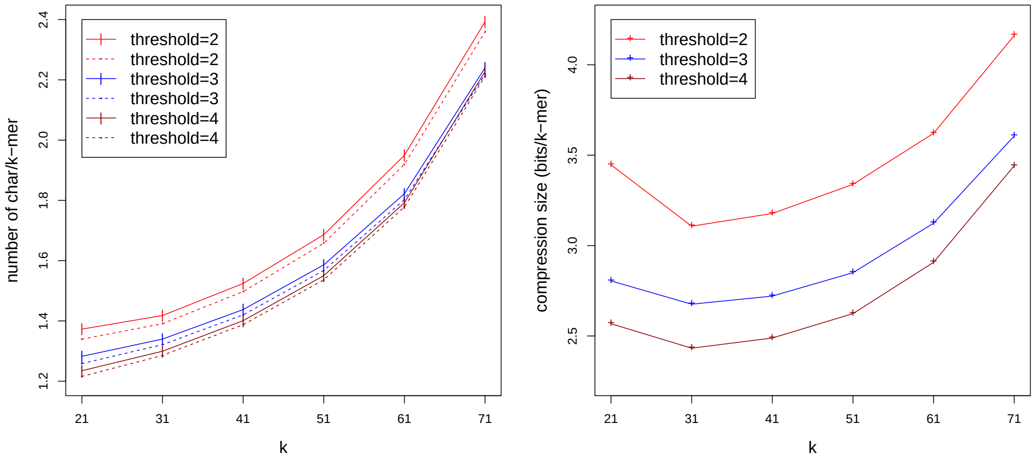


Figure 4 Compression performance of ESS-Compress when varying k and the low-frequency filter threshold, on Human RNA-seq dataset. In the left panel, solid lines represent the weight of the ESS-Compress representation, compared against the lower bound, represented by the dashed lines. In the right panel, compressed sizes are shown in bits/k-mer.

Table 4 Decompression time in *seconds*. The time is broken down into the portion taken by MFC to decompress the binary file into an enriched string set and the portion taken by our core algorithm to decompress the enriched string set into an SPSS. Note that BOSS does not implement decompression (because it is a membership data structure) so it is not included.

Dataset	UST-Compress	ESS-Tip-Compress			ESS-Compress		
	MFC-D	MFC-D	Core	Total	MFC-D	Core	Total
R. sphaeroides	3	2	1	4	2	1	3
Human RNA-seq	40	41	19	60	34	17	51
Gingiva metagenome	37	38	16	54	30	15	45
Soybean	31	33	13	46	29	13	42
Tongue metagenome	62	61	28	89	49	25	74
Whole human	302	337	259	596	303	250	553

usage. For UST-Compress, the time is dominated by the *cdBG* construction step (i.e. BCALM2). For ESS-Compress, the time and memory are significantly increased beyond that. Here, the advantage of ESS-Tip-Compress stands out. Its run time is nearly the same as UST-Compress, and its memory, while close to UST-Compress, is significantly lower than ESS-Compress.

Note that MFC is one of many DNA sequence compressors that can be used with our algorithms. MFC is known to achieve superior compression ratios but is slower for compression/decompression than other competitors [20]. We recommend using MFC since it was not the time or memory bottleneck during compression, in our datasets.

Table 5 Peak memory usage for compression and decompression. Decompression takes far less memory than compression, so compression memory is shown in *GB* and decompression memory in *MB*. Decompression memory is split in the same manner as the running time in Table 4.

Dataset	Compression (GB)				Decompression (MB)				
	BOSS	UST-Compress	ESS-Tip-Compress	ESS-Compress	UST-Compress	ESS-Tip-Compress		ESS-Compress	
					MFC-D	MFC-D	Core	MFC-D	Core
R. sphaeroides	2	3	3	3	509	513	3	513	4
Human RNA-seq	4	3	3	6	515	515	3	515	38
Gingiva metagenome	4	2	2	5	515	515	3	515	4
Soybean	4	2	2	3	515	515	3	515	12
Tongue metagenome	4	2	2	9	515	515	3	515	6
Whole human	5	12	11	42	515	515	3	515	735

Table 6 Compression time, measured in *minutes*. The column for BOSS includes the time for k-mer counting the reads using KMC [19], the time to run BOSS construction, and the time to run LZMA. The total time in UST-Compress, ESS-Tip-Compress and ESS-Compress include the time to compute *cdBG* from the reads using BCALM, which is same for all three. The columns labelled *core* refer to Algorithm 1. ESS-Tip-Compress core uses the specific instance of Algorithm 1 defined in Section 4.

Dataset	BOSS	BCALM	UST-Compress			ESS-Tip-Compress			ESS-Compress		
			UST	MFC	Total	Core	MFC	Total	Core	MFC	Total
R. sphaeroides	0.2	0.4	0.1	0.1	1	0.1	0.0	1	0.2	0.0	1
Human RNA-seq	4.0	6.6	1.6	0.8	9	1.3	0.7	9	5.0	0.6	12
Gingiva metagenome	4.3	5.5	1.2	0.7	7	1.0	0.7	7	3.4	0.6	10
Soybean	5.7	9.6	0.8	0.6	11	0.7	0.7	11	2.4	0.5	13
Tongue metagenome	7.4	8.7	1.6	0.8	11	1.9	1.1	12	7.6	0.9	17
Whole human	95	106	11	7	124	10	6	122	40	7	152

6 Discussion

In this paper, we presented a disk compression algorithm for k-mer sets called ESS-Compress. ESS-Compress is based on the strategy of representing a set of k-mers as a set of longer strings with as few total characters as possible. Once this string set is constructed, it is compressed using a generic nucleotide compressor such as MFC. On real data, ESS-Compress uses up to 42% less characters than the previous best algorithm UST-Compress. After MFC compression, ESS-Compress uses up to 27% less bits than UST-Compress.

We also presented a second algorithm ESS-Tip-Compress. It is simpler than ESS-Compress and does not achieve as good of compression sizes. However, the difference is less than 8% on our data, and it has the advantage of being about twice as fast and using significantly less memory during compression. For many users, this may be a desirable trade-off.

Our algorithms can also be used to compress information associated with the k-mers in K, such as their counts. Every k-mer in K corresponds to a unique location in the enriched string set. The counts can then be ordered sequentially, in the same order as the k-mers appear in the string set, and stored in a separate file. This file can then be compressed/decompressed separately using a generic compressor. After decompression of the enriched string set, the order of k-mers in the output SPSS will be the same as in the counts file.

We discussed several potential improvements to ESS-Compress, such as allowing more edges in the compacted de Bruijn graph to be absorbing or exploring the space of all path covers. We also gave a lower bound to what such improvements could achieve and showed they cannot gain more than 2% in space on our datasets. This makes these improvement of little interest, unless we encounter datasets where the gap is much larger.

ESS-Compress works by removing redundant $(k-1)$-mers from the string set, but a more general strategy could be to somehow remove ℓ-mer duplicates, for all $\ell_{min} \leq \ell \leq k-1$. Such a strategy would require novel algorithms but would still be unable to reduce the characters per k-mer below one. On our datasets, this amounts to at most a 30% improvement in characters, which would be further reduced after MFC compression. It is also not clear if a 30% improvement in characters is even possible, since this kind of strategy would require a more sophisticated encoding scheme with more overhead.

Another direction to achieve lower compression sizes is to look beyond string set approaches. We observe, for example, that the large improvement of ESS-Compress compared to UST-Compress, measured in the weight of the string set, was significantly reduced when measured in bits after MFC compression. This indicates that some of the work done by ESS-Compress duplicates the work done by MFC on UST, which is itself designed to remove redundancy in the input. Thus, generic compressors such as MFC could potentially be modified to work directly on k-mer sets.

We believe that the biggest opportunity for improving the two algorithms of this paper are the compression time and memory. The time is dominated by the initial step of running BCALM2 to find unitigs. It may be possible to avoid this step by running UST directly on the non-compacted graph. Such an approach was taken in [8], and it would be interesting to see if it ends up improving on the memory and run-time of BCALM2. The memory usage, on the other hand, can likely be optimized with better software engineering. The current implementation of Algorithm 2 is done in a memoized bottom-up manner. Instead, a top down iterative implementation can reduce memory usage by directly writing to disk as soon as a vertex is processed. A "max-depth" option in Algorithm 2 could also be used to limit the depth of the recursion, thereby controlling memory at the cost of the compression ratio.

References

1 URL: `https://github.com/cosmo-team/cosmo/tree/VARI`.

2 URL: `https://github.com/prophyle/prophasm`.

3 Jørgen Bang-Jensen and Gregory Z Gutin. *Digraphs: theory, algorithms and applications.* Springer Science & Business Media, 2008.

4 Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A Gurevich, Mikhail Dvorkin, Alexander S Kulikov, Valery M Lesin, Sergey I Nikolenko, Son Pham, Andrey D Prjibelski, et al. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology*, 19(5):455–477, 2012.

5 Timo Bingmann, Phelim Bradley, Florian Gauger, and Zamin Iqbal. COBS: a compact bit-sliced signature index. *arXiv preprint arXiv:1905.09624*, 2019.

6 Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de bruijn graphs. In *Proceedings of the 12th International Conference on Algorithms in Bioinformatics*, volume 7534 of *LNCS*, page 225–235. Springer, 2012.

7 Phelim Bradley, Henk C den Bakker, Eduardo PC Rocha, Gil McVean, and Zamin Iqbal. Ultrafast search of all deposited bacterial and viral genomic data. *Nature biotechnology*, 37(2):152, 2019.

8 Karel Břinda, Michael Baym, and Gregory Kucherov. Simplitigs as an efficient and scalable representation of de Bruijn graphs. *bioRxiv*, 2020. `doi:10.1101/2020.01.12.903443`.

9 Karel Břinda. *Novel computational techniques for mapping and classifying Next-Generation Sequencing data.* PhD thesis, Université Paris-Est, November 2016. `doi:10.5281/zenodo.1045317`.

10 Rayan Chikhi, Jan Holub, and Paul Medvedev. Data structures to represent sets of k-long DNA sequences. *arXiv:1903.12312 [cs, q-bio]*, 2019.

11 Rayan Chikhi, Antoine Limasset, Shaun Jackman, Jared T Simpson, and Paul Medvedev. On the representation of de Bruijn graphs. In *Research in Computational Molecular Biology, RECOMB 2014*, volume 8394 of *Lecture Notes in Computer Science*, pages 35–55. Springer, 2014.

12 Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de Bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12):i201–i208, 2016.

13 Temesgen Hailemariam Dadi, Enrico Siragusa, Vitor C Piro, Andreas Andrusch, Enrico Seiler, Bernhard Y Renard, and Knut Reinert. DREAM-Yara: An exact read mapper for very large databases with short update time. *Bioinformatics*, 34(17):i766–i772, 2018.

14 Luca Denti, Marco Previtali, Giulia Bernardini, Alexander Schönhuth, and Paola Bonizzoni. MALVA: genotyping by Mapping-free ALlele detection of known VAriants. *iScience*, 18:20–27, 2019.

15 R. S. Harris and P. Medvedev. Improved Representation of Sequence Bloom Trees. *Bioinformatics*, 36(3):721–727, 2020.

16 Mikel Hernaez, Dmitri Pavlichin, Tsachy Weissman, and Idoia Ochoa. Genomic Data Compression. *Annual Review of Biomedical Data Science*, 2, 2019.

17 Morteza Hosseini, Diogo Pratas, and Armando Pinho. A survey on data compression methods for biological sequences. *Information*, 7(4):56, 2016.

18 Costas S Iliopoulos, Ritu Kundu, and Solon P Pissis. Efficient pattern matching in elastic-degenerate texts. In *International Conference on Language and Automata Theory and Applications*, pages 131–142. Springer, 2017.

19 Marek Kokot, Maciej Długosz, and Sebastian Deorowicz. KMC 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, 2017.

20 Kirill Kryukov, Mahoko Takahashi Ueda, So Nakagawa, and Tadashi Imanishi. Nucleotide Archival Format (NAF) enables efficient lossless reference-free compression of DNA sequences. *bioRxiv*, page 501130, 2018.

21 Guillaume Marçais and Carl Kingsford. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, 27(6):764–770, 2011.

22 Camille Marchet, Christina Boucher, Simon J Puglisi, Paul Medvedev, Mikaël Salson, and Rayan Chikhi. Data structures based on k-mers for querying large collections of sequencing datasets. *bioRxiv*, page 866756, 2019.

23 Camille Marchet, Zamin Iqbal, Daniel Gautheret, Mikaël Salson, and Rayan Chikhi. Reindeer: efficient indexing of k-mer presence and abundance in sequencing datasets. *bioRxiv*, 2020.

24 Ibrahim Numanagić, James K Bonfield, Faraz Hach, Jan Voges, Jörn Ostermann, Claudio Alberti, Marco Mattavelli, and S Cenk Sahinalp. Comparison of high-throughput sequencing data compression tools. *Nature methods*, 13(12):1005, 2016.

25 Brian D Ondov, Todd J Treangen, Páll Melsted, Adam B Mallonee, Nicholas H Bergman, Sergey Koren, and Adam M Phillippy. Mash: fast genome and metagenome distance estimation using MinHash. *Genome biology*, 17(1):132, 2016.

26 Prashant Pandey, Fatemeh Almodaresi, Michael A Bender, Michael Ferdman, Rob Johnson, and Rob Patro. Mantis: A fast, small, and exact large-scale sequence-search index. *Cell systems*, 7(2):201–207, 2018.

27 Prashant Pandey, Michael A Bender, Rob Johnson, and Rob Patro. Squeakr: an exact and approximate k-mer counting system. *Bioinformatics*, 34(4):568–575, 2017.

28 Armando J Pinho and Diogo Pratas. MFCompress: a compression tool for FASTA and multi-FASTA data. *Bioinformatics*, 30(1):117–118, 2013.

29 Amatur Rahman and Paul Medvedev. Representation of k-mer sets using spectrum-preserving string sets. In *Research in Computational Molecular Biology - 24th Annual International Conference, RECOMB 2020, Padua, Italy, May 10-13, 2020, Proceedings*, volume 12074 of *Lecture Notes in Computer Science*, pages 152–168. Springer, 2020.

30 Guillaume Rizk, Dominique Lavenier, and Rayan Chikhi. DSK: k-mer counting with very low memory usage. *Bioinformatics*, 29(5):652–653, 2013.

31 Steven L Salzberg, Adam M Phillippy, Aleksey Zimin, Daniela Puiu, Tanja Magoc, Sergey Koren, Todd J Treangen, Michael C Schatz, Arthur L Delcher, Michael Roberts, et al. GAGE: A critical evaluation of genome assemblies and assembly algorithms. *Genome research*, 22(3):557–567, 2012.

32 B. Solomon and C. Kingsford. Fast search of thousands of short-read sequencing experiments. *Nature biotechnology*, 34(3):300–302, 2016.

33 B. Solomon and C. Kingsford. Improved Search of Large Transcriptomic Sequencing Databases Using Split Sequence Bloom Trees. *Journal of Computational Biology*, 25(7):755–765, 2018.

34 Daniel S Standage, C Titus Brown, and Fereydoun Hormozdiari. Kevlar: a mapping-free framework for accurate discovery of de novo variants. *iScience*, 18:28–36, 2019.

35 Chen Sun, Robert S. Harris, Rayan Chikhi, and Paul Medvedev. AllSome Sequence Bloom Trees. In *Research in Computational Molecular Biology - 21st Annual International Conference, RECOMB 2017, Hong Kong, China, May 3-7, 2017, Proceedings*, volume 10229 of *Lecture Notes in Computer Science*, pages 272–286, 2017.

36 Chen Sun and Paul Medvedev. Toward fast and accurate snp genotyping from whole genome sequencing data for bedside diagnostics. *Bioinformatics*, 35(3):415–420, 2018.

37 Isaac Turner, Kiran V Garimella, Zamin Iqbal, and Gil McVean. Integrating long-range connectivity information into de bruijn graphs. *Bioinformatics*, 34(15):2556–2565, 2018.

38 Derrick E Wood and Steven L Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome biology*, 15(3):R46, 2014.

Near-Linear Time Edit Distance for Indel Channels

Arun Ganesh
Department of Electrical Engineering and Computer Sciences, UC Berkeley, CA, USA
arunganesh@berkeley.edu

Aaron Sy
Department of Electrical Engineering and Computer Sciences, UC Berkeley, CA, USA
raaronsy@gmail.com

Abstract

We consider the following model for sampling pairs of strings: s_1 is a uniformly random bitstring of length n, and s_2 is the bitstring arrived at by applying substitutions, insertions, and deletions to each bit of s_1 with some probability. We show that the edit distance between s_1 and s_2 can be computed in $O(n \ln n)$ time with high probability, as long as each bit of s_1 has a mutation applied to it with probability at most a small constant. The algorithm is simple and only uses the textbook dynamic programming algorithm as a primitive, first computing an approximate alignment between the two strings, and then running the dynamic programming algorithm restricted to entries close to the approximate alignment. The analysis of our algorithm provides theoretical justification for alignment heuristics used in practice such as BLAST, FASTA, and MAFFT, which also start by computing approximate alignments quickly and then find the best alignment near the approximate alignment. Our main technical contribution is a partitioning of alignments such that the number of the subsets in the partition is not too large and every alignment in one subset is worse than an alignment considered by our algorithm with high probability. Similar techniques may be of interest in the average-case analysis of other problems commonly solved via dynamic programming.

2012 ACM Subject Classification Theory of computation → Dynamic programming

Keywords and phrases edit distance, average-case analysis, dynamic programming, sequence alignment

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.17

Related Version A preprint of the paper is available at `https://arxiv.org/abs/2007.03040`.

Funding *Arun Ganesh*: Supported in part by NSF Award CCF-1535989.
Aaron Sy: Supported in part by NSF Award CCF-1535989.

Acknowledgements We thank Satish Rao for suggesting the problem and for pointing out the connection to alignment heuristics used in practice. We thank Nir Yosef for helpful discussions on models for indels used in computational biology. We thank the anonymous reviews for their helpful feedback regarding the presentation of the results.

1 Introduction

Edit distance is an important string similarity measure whose computation has applications in many fields including computational biology. Its simplest variant is the Levensthein distance, which is the minimum number of insertions, deletions, or substitutions required to turn the first string into the second. A textbook dynamic programming algorithm computes the edit distance between two length n strings in $O(n^2)$ time (see e.g. Section 6.3 of [12]), and the best known worst-case exact algorithm runs in $O(\frac{n^2}{\ln^2 n})$ time [23]. Assuming the Strong Exponential Time Hypothesis, Backurs and Indyk showed that no $O(n^{2-\epsilon})$ time algorithm exists [7] for any $\epsilon > 0$, and Bringmann and Künnemann extended this result to the special case of bitstrings, suggesting that these algorithms are near-optimal [10].

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti; Article No. 17; pp. 17:1–17:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In many practical applications, a quadratic runtime is prohibitively expensive. For example, it was once estimated that using the textbook algorithm to align the full genomes of a human and a mouse (although not a very practical problem) would take 95 CPU years [14]. When the edit distance is small, one can do better. An immediate result is that if the edit distance between two length n strings is at most d, it can be computed in time $O(nd)$ (by considering only entries in the dynamic programming table which are distance at most d from entries indexed (i, i) for some i), and Landau et al. give a more nuanced algorithm which finds the edit distance in time $O(n + d^2)$ [22]. However, when e.g. aligning the sequences of two different species the edit distance can still be as large as $\Omega(n)$, so these results do not offer substantial improvements over the textbook algorithm.

Motivated by this and the aforementioned lower bounds, there have been many efforts to design faster algorithms. Many worst-case approximation algorithms exist for the problem (e.g. [9, 5, 6, 11]). However, most results give super-constant approximation ratios, and even the known constant approximation ratios are perhaps too large for practical applications. For example, popular knowledge suggests that a 3-approximation algorithm[1] for edit distance when applied to genome sequences is not guaranteed to determine that humans are more closely related to dogs than chickens.

However, there is good reason to believe that in biological applications, the subquadratic lower bound is not applicable. Roughly speaking, the lower bounds of [7, 10] say that every part of one string must be compared to every part of another string in order to compute the edit distance exactly. In practice, this should rarely be true. e.g. when aligning two genomes, there is good reason to believe that the beginning of the first genome only needs to be compared to the beginning of the second genome. Observations like this motivate the need for *average-case analysis* of edit distance algorithms. There are already several results on average-case analyses of edit distance. For example, [4] gives an approximation algorithm when the inputs are chosen adversarially but then perturbed, [17] gives an exact algorithm when the inputs are compressible, and [21] gives an approximation algorithm when one of the input strings satisfies a pseudo-randomness condition. Note that all these results require losing an approximation factor (which as mentioned before is undesirable) and/or for fairly specific conditions (such as compressibility) to hold for the input.

1.1 Our Contribution

In this paper, we consider a model for average-case analysis of edit distance called the *indel channel* which is motivated by biological applications. In this model, we generate a random bitstring of length n as our first string (using bitstrings simplifies the presentation, and the results generalize easily to larger alphabets), and then at each position in the string randomly apply each of the three types of mutations (insertion, deletion, substitution) independently with some probability to get the second string. We let $ID(n)$ denote the distribution of pairs of strings and sets of mutations generated by this model. This model of random string mutation is popular as an extension of the CFN model for biological mutations in computational biology, and problems based on the indel channel have been defined and studied in the areas of sequence alignment [15], phylogenetic reconstruction [13, 2, 3, 16], and trace reconstruction [19, 24, 18]. We show that for pairs of strings generated by this model, we can compute their exact edit distance in near-linear time with high probability:

[1] The approximation ratio proven by [11] is 1680, though they conjecture their algorithm is actually a $(3 + \epsilon)$-approximation.

▶ **Theorem 1** (Informal). *Let s_1 be a uniformly random bitstring of length n and s_2 be the bitstring generated by applying substitution, insertion, and deletion to each bit of s_1 each uniformly at random and with probability at most some constant. Then with high probability we can compute the edit distance between s_1 and s_2 in $O(n \ln n)$ time.*

Our Techniques. Our algorithm is simple, using only the dynamic programming algorithm as a primitive. The high-level approach is as follows: While we cannot use the dynamic programming algorithm to compute the edit distance between the two strings and get a near-linear time algorithm, we can repeatedly use it to compute the edit distance between two substrings of length $k \ln n$, where k is a (sufficiently large) constant. Under the indel channel, a substring of length $k \ln n$ of the first string s_1 and the corresponding substring of the second string s_2 have low edit distance compared to two random substrings with high probability. So by computing the edit distance between two substrings of length $k \ln n$, we can determine if the correct alignment places these two substrings close to each other.

We can now use this as a primitive to find an alignment of the two strings that is an approximation of the "canonical" alignment, i.e. the alignment corresponding to the insertions and deletions caused by indel channel. If we know bit i of s_1 is aligned with bit j of s_2, then with high probability there are only $O(\ln n)$ indices in s_2 that bit $i + k \ln n$ of s_1 can be aligned with. Even if we only have an estimate for where bit i of s_1 is aligned with in s_2 that is $O(\ln n)$ bits off, with high probability the number of indices bit $i + k \ln n$ of s_1 can be aligned with is still $O(\ln n)$. So, once we have computed an approximate alignment for the first i bits of s_1, we can iteratively extend the approximate alignment by using a small number of edit distance computations on bitstrings of length $O(\ln n)$ to determine approximately where bit $i + k \ln n$ of s_1 should be aligned. We note that some past works studying the indel channel in phylogenetic reconstruction use the trivial "diagonal" alignment (e.g. [13, 16]) as an approximate alignment.

Once we have an approximate alignment, our algorithm is straightforward: Use the dynamic programming algorithm, but only compute entries in the dynamic programming table which are close to the approximate alignment. We show that with high probability, the best alignment is close to the canonical alignment suggested by the indel channel, which is close to our approximate alignment, giving the correctness of this algorithm. To show this statement holds, we would like to use the fact that that any alignment which differs significantly from the canonical alignment is better than the canonical alignment with probability decaying exponentially in the difference between the two alignments. However, there are too many alignments for us to conclude by combining this fact with a union bound. Instead, we construct a partition $\mathcal{B}$ of the alignments such that for each element B of the partition $\mathcal{B}$, the alignments in B are structurally similar. Roughly speaking, this lets us argue for each B that with probability much smaller than $1/|\mathcal{B}|$ all alignments in B are not optimal. We can then take a union bound over the subsets in $\mathcal{B}$ instead of over all alignments to get the desired statement.

We note that techniques similar to finding an approximate alignment and then computing the DP table restricted to entries near this alignment are used in heuristics in practice such as BLAST [1], FASTA [25], and MAFFT [20]. Our analysis thus can be viewed as theoretical support for these kinds of heuristics.

The rest of the paper is as follows: In Section 2, we define the indel channel model formally, give some simple probability facts that are useful, and define some terms that appear frequently in the analysis. In Section 3, as a warm-up we show that in the substitution-only case, the optimal alignment is close to the diagonal. In Section 4 we describe and analyze our algorithm for finding an approximate alignment. In Section 5, we extend the analysis from Section 3 to the general case, completing the proof of Theorem 1.

2 Preliminaries and Definitions

To simplify the presentation, we will often treat possibly non-integer numbers like $\ln n, k \ln n$ and $n/k \ln n$ as integers without explicitly rounding them first. The correctness of all proofs in the paper is unaffected by replacing these quantities by their rounded versions (e.g. $\lceil \ln n \rceil$) where appropriate.

2.1 Problem Setup

In this section, we describe the model used to generate the pairs of correlated strings and formally state our main result. We start by sampling a uniformly random bitstring $s_1 \sim \{0,1\}^n$. We pass s_1 through an indel channel to arrive at a new bitstring s_2. When passed through the indel channel, for the jth bit of s_1, $b_j := (s_1)_j$:

- b_j is substituted, i.e. flips, with probability p_s.
- b_j is deleted, with probability
  - p_d if the previous bit b_{j-1} was not deleted,
  - $q_d > p_d$ if the previous bit b_{j-1} was deleted. (This is similar but not equivalent to deleting a geometric number of bits whenever a deletion occurs)

  That is, whenever a bit b_j is deleted, an additional number of bits equal to roughly a geometric random variable with mean $1/(1-q_d)$ are deleted to the right of b_j.
- An insertion event occurs with probability p_i, inserting a uniformly random bit string $t \sim \{0,1\}^I$ with length $I \sim \text{Geo}\,(1-q_i)$ (I has mean $1/(1-q_i)$) to the right of b_j. Inserted bits are not further acted upon by the indel channel.

We call each of these edits, and use $\mathcal{E}$ to denote the set of edits occurring in the indel channel. Each mutation happens independently for each bit and across different bits. As mentioned before, this definition of the indel channel is chosen to parallel models in both the computer science theory and computational biology communities that account for splicing in/out entire subsequences rather than individual sites (e.g. see [15] for an example of a model for mutation which uses geometric indel lengths; of course, setting $q_d = p_d, q_i = 0$ gives the setting where only single bits are spliced in/out). We require that

$$p_s \le \rho_s, \qquad p_d \le \rho_d, \frac{1-\rho_d}{1-q_d} \le \rho'_d, \qquad p_i \le \rho_i, \frac{1}{1-q_i} \le \rho'_i. \tag{1}$$

for some small constants $\{\rho\} := \{\rho_s, \rho_d, \rho'_d, \rho_i, \rho'_i\}$. Our lemma/theorem statements will implicity assume (1) holds, and our proofs will specify certain inequalities which must hold for the values $\{\rho\}$, thus specifying a range of values for the mutation probabilities for which our algorithm is proven to work. We do not attempt to the optimize the values of $\{\rho\}$ for which our algorithm works, but will state the exact inequalities that need to hold for $\{\rho\}$ when it is convenient to do so. We use $ID(n)$ to denote the distribution of tuples $(s_1, s_2, \mathcal{E})$ arrived at by this process for some p, q values - we often make statements about $(s_1, s_2, \mathcal{E}) \sim ID(n)$ which apply for any realization of the p, q values satisfying the constraints given by $\{\rho\}$, in which case we will not specify what these values are. Given $(s_1, s_2, \mathcal{E}) \sim ID(n)$ we want to compute the edit distance $ED(s_1, s_2)$ between s_1 and s_2 as quickly as possible. For simplicity, we specifically use the Levenshtein distance in our proofs, but they can easily be generalized to other sets of penalties for edits. We now formally restate Theorem 1 as our main result:

▶ **Theorem 2.** *Assuming* (1) *holds for certain constants* $\{\rho\}$*, there exists a (deterministic) algorithm running in time* $O(n \ln n)$ *that computes* $ED(s_1, s_2)$ *for* $(s_1, s_2, \mathcal{E}) \sim ID(n)$ *with probability* $1 - n^{-\Omega(1)}$.

2.2 Probability Facts

We start with some basic probability facts appearing in our analysis. We denote the number of ways to sort $a+b+c$ elements into three groups of size a, b, c, i.e. trinomial, by $\binom{a+b+c}{a,b,c}$. This of course equals $\frac{(a+b+c)!}{a!b!c!}$. When the trinomial appears, we use Stirling's approximation to bound its value:

▶ **Fact 3** (Stirling's approximation). $\sqrt{2\pi}n^{n+1/2}e^{-n} \leq n! \leq en^{n+1/2}e^{-n}$.

We do not aim to optimize constants, so we will use the following standard simplified Chernoff bound in our proofs:

▶ **Fact 4** (Chernoff bound). *Let $X_1 \dots X_n$ be independent Bernoulli random variables and $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathbb{E}[X]$. Then for $0 < \epsilon < 1$:*

$$Pr[X \geq (1+\epsilon)\mu] \leq e^{-\frac{\epsilon^2\mu}{3}}, \qquad Pr[X \leq (1-\epsilon)\mu] \leq e^{-\frac{\epsilon^2\mu}{2}}.$$

We will also use the following simplified negative binomial tail bound:

▶ **Fact 5** (Negative binomial tail bound). *Let $X \sim NBinom(n, p)$, i.e. X is a random variable equal to the number of probability p success events needed before n successes are seen. Then for $k > 1$:*

$$Pr[X \geq kn/p] \leq e^{-\frac{kn(1-1/k)^2}{2}}.$$

Proof. This follows from noticing that $Pr[X \geq kn/p] = Pr[Binom(kn/p, p) < n]$ and applying a Chernoff bound with $\epsilon = 1 - 1/k$. ◀

We'll chain together these facts to get a tail bound for a binomial number of geometric random variables:

▶ **Lemma 6.** *Consider $X \sim NBinom(m, q)$ where $m = \sum_{i=1}^{t} m_i$, $m_i \sim Bern(p_i)$, i.e. X is a random variable obtained by first sampling m, the sum of t independent Bernoullis, and then sampling $X \sim NBinom(m, q)$. Then for $1 < k \leq 4$, $\mu = \sum_{i=1}^{t} p_i$:*

$$Pr\left[X \geq k \cdot \frac{\mu}{q}\right] \leq e^{-\frac{(\sqrt{k}-1)^2\mu}{3}} + e^{-\frac{k\mu(1-1/\sqrt{k})^2}{2}}.$$

A proof is given in Appendix A.

2.3 Definitions

In this section we give definitions that simplify the presentation. There are many identical definitions for solutions to the edit distance problem - we will define solutions as paths through the dependency graph as doing so simplifies the presentation of the analysis.

▶ **Definition 7.** *Consider the **dependency graph** of the edit distance dynamic programming table: For two strings s_1, s_2 of length n_1, n_2, the dependency graph of s_1, s_2 has vertices (i, j) for $i \in \{0, 1, \dots n_1\}, j \in \{0, 1, \dots n_2\}$ and directed edges from (i, j) to $(i+1, j), (i, j+1)$ and $(i+1, j+1)$ for $i \in [n_1], j \in [n_2]$ if these vertices exist. The edges $\{(i-1, j-1), (i, j)\}$ where $(s_1)_i = (s_2)_j$ have weight 0 and all other edges have weight 1. The **edit distance** between s_1 and s_2, denoted $ED(s_1, s_2)$, is the (weighted) shortest path from $(0, 0)$ to (n_1, n_2).*

For completeness we recall the standard dynamic programming algorithm for edit distance and its restriction to a subset of indices.

▶ **Fact 8** (Textbook Algorithm). *The edit distance between s_1, s_2 of length n_1, n_2 can be computed in $O(n_1 n_2)$ time by using e.g. the $O(|V| + |E|)$-time*[2] *dynamic programming algorithm for shortest paths in a DAG. In addition, if we know the shortest path in the dependency graph is contained in vertex set V', we can compute the edit distance in $O(|V'|)$ time by applying the dynamic program "restricted to V'" i.e. by applying it to the dependency graph after deleting all vertices not in V'.*

▶ **Definition 9.** *An* ***alignment*** *(of two strings s_1, s_2) is any path $A = \{(i_1 = 0, j_1 = 0), (i_2, j_2) \dots (i_{L-1}, j_{L-1}), (i_L = n_1, j_L = n_2)\}$ from $(0, 0)$ to (n_1, n_2) in the dependency graph of s_1, s_2. Denote the set of all alignments by $\mathcal{A}$.*

For convenience, we will abuse notation and sometimes use A to also denote the cost of alignment A, e.g. using $A \geq A'$ to denote that the cost of A is at least the cost of A'.

▶ **Definition 10.** *For $(s_1, s_2, \mathcal{E}) \sim ID(n)$, the* ***canonical alignment*** *of s_1, s_2, denoted A^*, is informally the alignment corresponding to $\mathcal{E}$. More formally, A^* starts at $(0, 0)$, and for each row i of the dependency graph, if the first vertex in A^* in this row is (i, j), we extend A^* as follows according to $\mathcal{E}$:*

- *If no insertion or deletion occurs on the ith bit, we include the edge $\{(i, j), (i + 1, j + 1)\}$.*
- *If an insertion of I bits occurs on the ith bit and no deletion occurs, we include the path $\{(i, j), (i, j + 1), \dots (i, j + I), (i + 1, j + I + 1)\}$*
- *If a deletion and no insertion occurred on the ith bit, we include the edge $\{(i, j), (i+1, j)\}$.*
- *If an insertion of I bits occurred and a deletion, we include the path $\{(i, j), (i, j + 1), \dots (i, j + I), (i + 1, j + I)\}$.*

The definition of (canonical) alignments depends on the pair of strings s_1, s_2, but throughout the paper usually it will be clear that the pair of strings being referred to is sampled from $ID(n)$, so for brevity's sake we may refer to a canonical alignment without referring to strings, letting the strings be implicit.

Note that the canonical alignment is not necessarily the optimal alignment (in fact, even in the substitution-only case, the substitutions cause the optimal alignment to be one including insertions and deletions with high probability). However, alignments which differ sufficiently from the canonical alignment should not perform better than the canonical alignment with high probability. For alignments which aren't the canonical alignment, we characterize their differences from the canonical alignment in terms of where they break from the canoncial alignment.

▶ **Definition 11.** *Fix a canonical alignment A^*, and let A be any alignment. A* ***break*** *of A (from A^*) is any subpath $(\{(i_1, j_1), (i_2, j_2) \dots (i_L, j_L)\})$ of A such that (i_1, j_1) and (i_L, j_L) are in A^* but none of (i_2, j_2) to (i_{L-1}, j_{L-1}) are in A^*. The* ***length*** *of the break is the value $i_L - i_1$.*

For $(s_1, s_2, \mathcal{E}) \sim ID(n)$, a break of alignment A from (i_1, j_1) to (i_L, j_L) is ***long*** *if its length is at least $k \ln n$ (for a constant k to be specified later) and* ***short*** *otherwise*[3]*. An alignment is* ***good*** *if it has no long breaks and* ***bad*** *if it has at least one long break.*

[2] Note that the dependency graph has $|E| = O(|V|)$.

[3] Note that in the definition of length, we use $i_L - i_1$ and ignore j_1, j_L. This is because with high probability, for all i, i' such that $i' > i + k \ln n$, if the canonical alignment goes through (i, j) and (i', j'), $j' - j$ will be within a constant factor of $i' - i$. So defining length as $i_L - i_1$ instead of $j_L - j_1$ will not substantially affect our categorization of which breaks are short or long.

Intuitively, short breaks are smaller and might make an alignment better than the canonical alignment, so we can't rule out alignments containing only short breaks in our analysis. On the other hand, long breaks are sufficiently large such that replacing them with the corresponding part of the canonical alignment should be an improvement with high probability. Lastly, we define two functions that take alignments and make them look more like the canonical alignment A^*.

▶ **Definition 12** (Short and Long Break Replacement). *We define $\mathcal{SBR} : \mathcal{A} \mapsto \mathcal{A}$ as a function from alignments to alignments, such that for any alignment A, $\mathcal{SBR}(A)$ is the alignment arrived at by applying the following modification to all short breaks in A: For a short break from (i_1, j_1) to (i_L, j_L), replace it with the subpath of A^* from (i_1, j_1) to (i_L, j_L). We define $\mathcal{LBR}$ analogously, except $\mathcal{LBR}$ applies the modification to all long breaks instead of short breaks.*

Note that all alignments in the range of $\mathcal{LBR}$ are good by definition. The idea behind these functions and the definitions of good and bad alignments is to use them in the analysis as follows: It is possible to compute the best of the good alignments quickly by only considering a narrow region within the DP table. So it suffices to show any bad alignment is not the best alignment. For a single bad alignment A, it is fairly straightforward to show that A^* is better than A with high probability. However, there are many bad alignments and thus a simple union bound does not suffice to complete the analysis. We instead use $\mathcal{LBR}$ to show that it suffices if all alignments in the range of $\mathcal{SBR}$ are not better than A^* with high probability. There are considerably fewer of these alignments and they can be partitioned in a way that is easy to analyze, and so simple counting and probability techniques let us show this holds.

3 Substitution-Only Case

As a warmup, let's consider the easier case when only substitutions are present in the indel channel. In this case, A^* is just the diagonal $\{(0,0), (1,1) \dots (n,n)\}$. We show the following theorem:

▶ **Theorem 13.** *For $(s_1, s_2, \mathcal{E}) \sim ID(n)$ with $p_i, p_d = 0$, as long as $p_s \leq \rho_s$ where $\rho_s = .028$, there is an $O(n \ln n)$ time algorithm for calculating $ED(s_1, s_2)$ which is correct with probability $1 - n^{-\Omega(1)}$.*

The algorithm is simple - compute entries of the canonical DP table indexed by (i, j) where $|i - j| \leq k \ln n$, ignoring dependencies on entries for which $|i - j| > k \ln n$. The value of k used in the algorithm and the definition of long breaks will be specified by the analysis, which will determine a lower bound for k needed to make the failure probability sufficiently small.

We start by showing that "off-diagonal" alignments, i.e. alignments which do not share any edges with A^*, are not better than A^* with high probability. While there are many bad alignments which are not entirely off-diagonal, this will be useful as later we can show that a bad alignment A in the range of $\mathcal{SBR}$ being better than A^* corresponds to an off-diagonal alignment being better than A^* in a subproblem.

▶ **Lemma 14.** *For $(s_1, s_2, \mathcal{E}) \sim ID(n)$, with probability $1 - e^{-\Omega(n)}$, $A > A^*$ for all alignments A such that A and A^* do not share any edges.*

Proof. The cost of A^* can be upper bounded using a Chernoff bound: The expected number of substitutions is at most $\rho_s n$, so Fact 4 gives

$$Pr\left[A^* \leq \frac{3}{2}\rho_s n\right] \leq 1 - e^{-\frac{\rho_s n}{12}}.$$

Now our goal is to show that with high probability, no alignment A that does not share edges with A^* has cost lower than cn (where $c = \frac{3}{2}\rho_s$). We achieve this using a union bound over alignments, grouping alignments by their number of deletions d (which in the substitution-only case is also the number of insertions). We can ignore alignments with more than $cn/2$ deletions, as they will of course have cost more than cn.

$$\begin{aligned}
Pr[\exists A, A \leq cn] &\leq \sum_{d=1}^{cn/2} \sum_{A \text{ with } d \text{ deletions}} Pr[A \leq cn] \\
&\leq \sum_{d=1}^{cn/2} \binom{n+d}{d, d, n-d} Pr\left[Binom(n-d, \frac{1}{2}) \leq cn - 2d\right] \\
&\leq \frac{cn}{2} \binom{(1+\frac{c}{2})n}{\frac{c}{2}n, \frac{c}{2}n, (1-\frac{c}{2})n} Pr\left[Binom((1-\frac{c}{2})n, \frac{1}{2}) \leq cn\right].
\end{aligned}$$

The second line counts the number of alignments with d deletions, and it expresses the probability of success in terms of the number of substitutions, or edges in A of the form $((i-1, j-1), (i, j))$: The cost of each off-diagonal edge of the form $((i-1, j-1), (i, j))$ is $Bern(\frac{1}{2})$, even if we condition on the cost of all previous edges in A: assuming wlog that $i > j$ knowing the costs of all edges before $((i-1, j-1), (i, j))$ in A gives no information about the bit i of s_1, which is distributed uniformly at random. So the total cost of these edges is given by $Binom((1-\frac{c}{2})n, \frac{1}{2})$. In the third line we upper bound the probability for simplicity. A Chernoff bound now gives:

$$\begin{aligned}
Pr\left[Binom((1-\frac{c}{2})n, \frac{1}{2}) \leq cn\right] &= Pr\left[Binom((1-\frac{c}{2})n, \frac{1}{2}) \leq (1 - \frac{2-5c}{2-c})\frac{1}{2}(1-\frac{c}{2})n\right] \\
&\leq \exp\left(-\frac{(2-5c)^2}{8(2-c)}n\right). \qquad (2)
\end{aligned}$$

Next we upper bound the trinomial using Stirling's approximation:

$$\begin{aligned}
\binom{(1+\frac{c}{2})n}{\frac{c}{2}n, \frac{c}{2}n, (1-\frac{c}{2})n} &\leq \frac{e}{(2\pi)^{3/2}} \frac{((1+\frac{c}{2})n)^{(1+\frac{c}{2})n+\frac{1}{2}}}{(\frac{c}{2}n)^{cn+1}((1-\frac{c}{2})n)^{(1-\frac{c}{2})n+\frac{1}{2}}} \\
&\leq \frac{e}{(2\pi)^{3/2}} \frac{2}{cn} \sqrt{\frac{2+c}{2-c}} \left[\frac{(1+\frac{c}{2})^{(1+\frac{c}{2})}}{(\frac{c}{2})^c(1-\frac{c}{2})^{(1-\frac{c}{2})}}\right]^n. \qquad (3)
\end{aligned}$$

Putting everything together, we have the following upper bound

$$Pr[\exists A, A \leq cn] \leq \frac{e}{(2\pi)^{3/2}} \frac{2}{cn} \sqrt{\frac{2+c}{2-c}} \left[\frac{(1+\frac{c}{2})^{(1+\frac{c}{2})}}{(\frac{c}{2})^c(1-\frac{c}{2})^{(1-\frac{c}{2})}}\right]^n \left[\exp\left(-\frac{(2-5c)^2}{8(2-c)}\right)\right]^n.$$

For the above bound to be exponentially decaying in n, we need that:

$$\frac{(1+\frac{c}{2})^{(1+\frac{c}{2})}}{(\frac{c}{2})^c(1-\frac{c}{2})^{(1-\frac{c}{2})}} \exp\left(-\frac{(2-5c)^2}{8(2-c)}\right) < 1, \qquad (4)$$

which holds as long as $c \leq 0.042$, i.e. $\rho_s \leq .028$. For these values of c, with high probability $A^* < cn$ and $A > cn$ for any A which does not share any edges with A^*. ◀

We now make the following observations which will allow us to apply Lemma 14 to make more powerful statements about the set of all alignments:

▶ **Fact 15.** *Fix any $s_1, s_2, \mathcal{E}$ in the support of $ID(n)$, and let A, A' be any two alignments with the same set of long breaks. Then $\mathcal{LBR}(A) - A = \mathcal{LBR}(A') - A'$.*

This follows because applying $\mathcal{LBR}$ to A, A' results in the same pairs of subpaths being swapped (and thus the same change in cost) as A, A' have the same long breaks.

▶ **Corollary 16.** *Fix any $(s_1, s_2, \mathcal{E})$ in the support of $ID(n)$. If for all alignments A in the range of $\mathcal{SBR}$, $A \geq A^*$, then any lowest-cost good alignment is also a lowest-cost alignment.*

Proof. Applying a composition of $\mathcal{LBR}$ and $\mathcal{SBR}$ to any alignment gives A^*, and for any A, A and $\mathcal{SBR}(A)$ have the same long breaks. This gives that for any alignment A, $\mathcal{LBR}(A)$ (a good alignment) satisfies $\mathcal{LBR}(A) \leq A$:

$$\mathcal{LBR}(A) - A \overset{\text{Fact 15}}{=} \mathcal{LBR}(\mathcal{SBR}(A)) - \mathcal{SBR}(A) = A^* - \mathcal{SBR}(A) \leq 0.$$

Now, letting A' be a lowest-cost good alignment, we get $A \geq \mathcal{LBR}(A) \geq A'$ for all A, i.e. A' is the lowest cost alignment. ◀

We complete the argument by showing that the assumption of Corollary 16 holds with high probability.

▶ **Lemma 17.** *For $(s_1, s_2, \mathcal{E}) \sim ID(n)$, with probability $1 - n^{\Omega(1)}$ for all alignments A in the range of $\mathcal{SBR}$, $A \geq A^*$.*

Proof. As in Lemma 14, we apply a union bound over the range of $\mathcal{SBR}$, grouped by total length of breaks from A^*. Consider the set $\mathcal{A}_i$ contained in the range of $\mathcal{SBR}$, which contains all alignments A for which the sum of the lengths of breaks of A from A^* is in $[ik \ln n, (i+1)k \ln n)$. Then the sets $\{\mathcal{A}_i : 0 \leq i \leq \frac{n}{k \ln n}\}$ forms a disjoint cover of the range $\mathcal{SBR}(\mathcal{A})$. Note that elements of $\mathcal{A}_i$ have at most i breaks from A^*, each of length at least $k \ln n$. Also note that $\mathcal{A}_0$ is a singleton set containing only A^*.

For any alignment A, we call the set of starting and ending indices of all breaks of that alignment the *breakpoint configuration* of A (to simplify future analysis, we index with respect to s_1 [4]). Let $\mathcal{B}_i$ be the set of all possible breakpoint configurations of alignments in $\mathcal{A}_i$. We can view $B \in \mathcal{B}_i$ as a binary assignment of each edge in A^* to either agree or disagree with $A \in \mathcal{A}_i$. For a fixed set of break points $B \in \mathcal{B}_i$, let $\mathcal{A}_B$ be the set of all alignments having the breakpoints corresponding to B (i.e. every alignment in $\mathcal{A}_B$ has the same breaks from A^*). Note that the set $\{\mathcal{A}_B : B \in \mathcal{B}_i\}$ forms a disjoint cover of $\mathcal{A}_i$.

For any fixed set of breaks $B \in \mathcal{B}_i$, let s_1^B, s_2^B denote the restriction of s_1, s_2 to indices contained in the breaks in B, and $(A)_B$ denote the restriction of an alignment A to these indices. s_1^B, s_2^B are distributed according to $ID(b)$ for $b \geq ik \ln n$. Furthermore, for $A \in \mathcal{A}_B$, $A < A^*$ if and only if $(A)_B < (A^*)_B$. Since for all $A \in \mathcal{A}_B$, $(A)_B$ does not share any edges with $(A^*)_B$, by Lemma 14:

$$\Pr[\exists A \in \mathcal{A}_B, A < A^*] = \Pr[\exists A \in \mathcal{A}_B, (A)_B < (A^*)_B] \leq e^{-\Omega(ik \ln n)} = n^{-\Omega(ik)}.$$

[4] In the substitution only case, indexing with respect to s_1 and s_2 is the same, but when indels are present indexing with respect to s_1 will simplify the analysis.

This reduces our problem to that of counting the cardinality of $\mathcal{B}_i$:

$$\begin{aligned}
Pr[\exists A \in \mathcal{SBR}(\mathcal{A}), A < A^*] &= \sum_{i=1}^{\frac{n}{k \ln n}} Pr[\exists A \in \mathcal{A}_i, A < A^*] \\
&= \sum_{i=1}^{\frac{n}{k \ln n}} \sum_{B \in \mathcal{B}_i} Pr[\exists A \in \mathcal{A}_B, A < A^*] \\
&\leq \sum_{i=1}^{\frac{n}{k \ln n}} \sum_{B \in \mathcal{B}_i} n^{-\Omega(ik)} = \sum_{i=1}^{\frac{n}{k \ln n}} |\mathcal{B}_i| \, n^{-\Omega(ik)}.
\end{aligned}$$

Now we must count the cardinality of $\mathcal{B}_i$. We claim that each $B \in \mathcal{B}_i$ can be uniquely mapped to i or less contiguous subsets of $[n]$, each of a size in $[k \ln n, 2k \ln n)$ or size 0. There are at most $nk \ln n + 1$ such subsets (there are n different possible smallest elements for each non-empty subset, and $k \ln n$ different possible sizes for each non-empty subset, and the smallest element and size uniquely determine the non-empty subsets), giving that

$$|\mathcal{B}_i| \leq (nk \ln n + 1)^i.$$

Our mapping is as follows: For a break in $B \in \mathcal{B}_i$ which starts at index j and has length $\ell \in [i' k \ln n, (i'+1)k \ln n)$, we map the break to the subsets $\{j, j+1 \ldots j + k \ln n - 1\}, \{j + k \ln n, j + k \ln n + 1 \ldots j + 2k \ln n - 1\} \ldots \{j + (i'-1)k \ln n, j + (i'-1)k \ln n + 1 \ldots \ell\}$. That is, for a break we take the indices the break spans, and peel off the first $k \ln n$ elements to create a subset, until there are less than $2k \ln n$ indices remaining, which then form their own subset. We map B to the union of the subsets its breaks are mapped to, plus enough empty subsets to make the total number of subsets i. It is straightforward to see that this map from $\mathcal{B}_i$ to a set of subsets is injective as desired, and that the set of subsets has the stated properties.

Using $|\mathcal{B}_i| \leq (nk \ln n + 1)^i$ and assuming k is a sufficiently large constant we get:

$$Pr[\exists A \in \mathcal{SBR}(\mathcal{A}), A < A^*] \leq \sum_{i=1}^{\frac{n}{k \ln n}} (nk \ln n + 1)^i n^{-\Omega(ik)} \leq n^{-\Omega(k)}. \quad \blacktriangleleft$$

Proof of Theorem 13. The algorithm is to use the standard DP algorithm restricted to entries indexed by (i, j) where $|i - j| \leq k \ln n$, ignoring dependencies on entries for which $|i - j| > k \ln n$. Theorem 13 follows immediately from Corollary 16, Lemma 17, and the observation that all good alignments are contained in the set of entries used by the DP algorithm. ◀

4 Finding an Approximate Alignment

We now consider the case where insertions and deletions are present. While in the substitution case it is obvious that the canonical alignment is the diagonal, in the presence of insertions and deletions there is the additional algorithmic challenge of finding something close to the canonical alignment. We now use our previous definition for alignments to define an alignment function, which will be useful in analyzing the approximate alignment algorithm.

▶ **Definition 18.** *Given an alignment A of $(s_1, s_2, \mathcal{E}) \sim ID(n)$, let $f_A : [n] \to \mathbb{Z}$ be the function such that for all $i \in [n]$, $(i, f_A(i))$ is the first vertex in A of the form (i, j).*

Using this definition, $f_{A^*}(j)$ gives the location of the jth bit of s_1 in s_2, or if the jth bit is deleted, where the location would be had it not been deleted. To find the edit distance between s_1, s_2, our algorithm will start by computing an approximate alignment function which does not differ much from f_{A^*}. Before describing our algorithm, it will help to prove some properties about edit distances between pairs of strings sampled from $ID(n)$.

4.1 Properties of the Indel Channel

The term $(\rho_i\rho_i' + (\rho_d + 1/k\ln n)(\rho_d' + 1))$, which is roughly speaking an upper bound on the edit distance (divided by $k\ln n$) between s_1, s_2 sampled from $ID(k\ln n)$ due to indels, appears frequently in the rest of the analysis. To simplify the presentation, we denote $(\rho_i\rho_i' + (\rho_d + 1/k\ln n)(\rho_d' + 1))$ by κ_n for the rest of the paper. Our goal in the following lemmas is to show that by computing the edit distance between the substrings of length $k\ln n$ starting at bit i_1 of s_1 and bit i_2 of s_2, we can identify if $i_2 \approx f_{A^*}(i_1)$.

▶ **Lemma 19.** *For $(s_1, s_2, \mathcal{E}) \sim ID(n)$, let s_1' be the substring formed by bits i to $i+k\ln n-1$ of s_1, and s_2' be the substring formed by bits $f_{A^*}(i)$ to $f_{A^*}(i+k\ln n)-1$ of s_2. Then:*

$$\Pr_{(s_1,s_2,\mathcal{E})\sim ID(n)}\left[ED(s_1', s_2') \geq \frac{3}{2}(\rho_s + \kappa_n)k\ln n\right] \leq n^{-\rho_s k/12} + 2n^{-\rho_i k/60} + 3n^{-\rho_d k/60}.$$

Proof. The edit distance between s_1 and s_2 is upper bounded by the number of substitutions, deletions, and insertions that occur in the channel on bits i to $i+k\ln n-1$ of s_1. So it suffices to show this total is at most $\frac{3}{2}(\rho_s + \rho_i + \rho_d)k\ln n$ with high probability. In turn, it suffices to show the number of substitutions is at most $\frac{3}{2}\rho_s k\ln n$, the number of insertions is at most $\frac{3}{2}\rho_i\rho_i' k\ln n$, and the number of deletions is at most $\frac{3}{2}(\rho_d k\ln n + 1)(\rho_d' + 1)$ with high probability. We do this using a union bound over the three types of mutations.

The number of substitutions is at most $\rho_s k\ln n$ in expectation. A Chernoff bound with $\epsilon = 1/2$ gives that the number of substitutions exceeds $\frac{3}{2}\rho_s k\ln n$ with probability at most $n^{-\rho_s k/12}$. The probability the number of insertions exceeds $\frac{3}{2}\rho_i\rho_i' k\ln n$ is maximized when $p_i = \rho_i, 1/(1-q_i) = \rho_i'$. The number of insertions is then the random variable $NBinom(Binom(k\ln n, \rho_i), 1/\rho_i')$ with expectation $\rho_i\rho_i' k\ln n$, and by Lemma 6 with $k = 3/2$ the probability it exceeds $\frac{3}{2}\rho_i\rho_i' k\ln n$ is at most $2n^{-\rho_i k/60}$.

To bound the number of deletions, we consider the following process for deciding where deletions occur in s_1:

- For each bit of s_1 a "type 1" deletion occurs with probability p_d, except bit 1 of s_1 where the probability is q_d.
- For each bit j where a type 1 deletion occurs, we sample $\delta \sim Geo(\frac{1-q_d}{1-p_d})$. Let Δ be the number of bits between j and the next bit with a type 1 deletion. A type 2 deletion occurs on the $\min\{\delta, \Delta\}$ bits following j.

For bit 1, its probability of seeing a deletion in the indel channel is upper bounded by q_d. Otherwise, if no deletion occurs on bit $j-1$, then for bit $j > i$, the only way bit j sees a deletion is if it has a type 1 deletion, which occurs with probability p_d. If a deletion occurs on bit $j-1$ and bit j does not have a type 1 deletion, it sees a type 2 deletion with probability $(1 - \frac{1-q_d}{1-p_d}) = \frac{q_d-p_d}{1-p_d}$ by the properties of the geometric distribution (this is regardless of the type of deletion on bit $j-1$). So its overall probability of seeing a deletion is $p_d + (1-p_d)\frac{q_d-p_d}{1-p_d} = q_d$. So, the number of deletions in this process stochastically dominates the number of deletions on bits i to $i+k\ln n-1$ of s_1.

Then, the number of deletions is stochastically dominated by the random variable $X+Y$ arrived at by sampling $Y \sim Binom(k\ln n - 1, p_d) + Bern(q_d), X \sim NBinom(Y, \frac{1-q_d}{1-p_d})$, which exceeds $\frac{3}{2}(\rho_d + \rho_d\rho_d')k\ln n$ with maximum probability when $p_d = \rho_d$, $\frac{1-\rho_d}{1-q_d} = \rho_d'$. The probability Y exceeds $\frac{3}{2}(\rho_d k\ln n) + 1$ is at most $n^{-\rho_d k/12}$ by a Chernoff bound. The probability X exceeds $\frac{3}{2}(\rho_d k\ln n + 1)\rho_d'$ is at most $2n^{-\rho_d k/60}$ by Lemma 6 with $k = 3/2$. So by a union bound the probability the number of deletions exceeds $\frac{3}{2}(\rho_d k\ln n + 1)(\rho_d' + 1)$ is at most $3n^{-\rho_d k/60}$. ◀

▶ **Lemma 20.** *Let s_1, s_2 be bitstrings of length $k\ln n$, chosen independently and uniformly at random from all bitstrings of length $k\ln n$. Then $Pr[ED(s_1,s_2) \leq D] \leq \frac{(4e\frac{k\ln n}{D}+5e+\frac{4e}{D})^D}{2^{k\ln n}}$.*

The proof of this lemma is fairly standard (see e.g. [8, Lemma 8]). For completeness, we give a proof in Appendix A.

▶ **Lemma 21.** *For constant $k > 0$, $i \leq n - k\ln n$,*

$$Pr_{(s_1,s_2,\mathcal{E})\sim ID(n)}\left[|f_{A^*}(i+k\ln n) - f_{A^*}(i) - k\ln n| \leq \frac{3}{2}\kappa_n \cdot k\ln n\right] \geq$$
$$1 - 2n^{-\rho_i k/60} - 3n^{-\rho_d k/60}.$$

Proof. $f_{A^*}(i+k\ln n) - f_{A^*}(i) - k\ln n$ is the signed difference between the number of insertions and deletions happening in indices i to $i+k\ln n-1$ of s_1. A simple upper bound for this difference is the sum of the number of insertions and deletions. The same analysis as Lemma 19 gives the lemma. ◀

▶ **Corollary 22.** *Consider the following random process, which we denote $\mathcal{P}$: we choose i_1 such that $i_1 < n - k\ln n$, sample $(s_1,s_2,\mathcal{E}) \sim ID(n)$, and then choose an arbitrary i_2 such that $|i_2 - f_{A^*}(i_1)| \leq \ln n$ and i_2 is at least $k\ln n$ less than the length of s_2. Let s_1' denote the string consisting of bits i_1 to $i_1 + k\ln n - 1$ of s_1 and s_2' the string consisting of bits i_2 to $i_2 + k\ln n - 1$ of s_2. Then for any i_2 we choose satisfying the above conditions,*

$$\Pr_{\mathcal{P}}\left[ED(s_1', s_2') \leq (1 + \frac{3}{2}(\rho_s + 2\kappa_n))k\ln n\right] \geq$$
$$1 - 2n^{-\rho_i k/12} - 4n^{-\rho_i k/60} - 6n^{-\rho_d k/60}.$$

Proof. By Lemma 21 and the assumptions in the corollary statement, with probability at least $1 - 2n^{-\rho_i k/60} - 3n^{-\rho_d k/60}$, the edit distance between s_2' and bits $f_{A^*}(i_1)$ to $f_{A^*}(i_1 + k\ln n) - 1$ of s_2 (call this substring s_2^*) is at most $(1 + \frac{3}{2}\kappa_n)k\ln n$ (the upper bound on the difference between starting indices plus the high-probability upper bound on the difference between ending indices). s_2^* is the result of passing s_1' through the indel channel, so by Lemma 19 with probability at least $1 - n^{-\rho_s k/12} - 2n^{-\rho_i k/60} - 3n^{-\rho_d k/60}$, the edit distance between s_2^* and s_1' is at most $\frac{3}{2}(\rho_s + \kappa_n)k\ln n$, giving the lemma by a union bound and triangle inequality. ◀

▶ **Corollary 23.** *Consider the following random process, which we denote $\mathcal{P}$: we choose i_1 such that $i_1 < n - k\ln n$, sample $(s_1,s_2,\mathcal{E}) \sim ID(n)$, and then choose an arbitrary i_2 such that*

$$|i_2 - f_{A^*}(i_1)| > \left(\frac{3}{2}\kappa_n + 1\right)k\ln n,$$

and i_2 is at least $k\ln n$ less than the length of s_2. Let s_1' denote the string consisting of bits i_1 to $i_1 + k\ln n - 1$ of s_1 and s_2' the string consisting of bits i_2 to $i_2 + k\ln n - 1$ of s_2. Then for $0 < r < 1$,

$$\Pr_{\mathcal{P}}[ED(s_1', s_2') > kr\ln n] \geq 1 - \left[\frac{(\frac{4e}{r}+5e+\frac{4e}{kr\ln n})^r}{2}\right]^{k\ln n} - 2n^{-\rho_i k/60} - 3n^{-\rho_d k/60}.$$

Proof. Either $i_2 < f_{A^*}(i_1) - k\ln n$ or $i_2 > f_{A^*}(i_1) + (\frac{3}{2}\kappa_n + 1)k\ln n$. If $i_2 < f_{A^*}(i_1) - k\ln n$, then none of the bits in s_2' are inherited from bits in s_1'. If $i_2 > f_{A^*}(i_1) + (\frac{3}{2}\kappa_n + 1)k\ln n$, then by Lemma 21 we have with probability $1 - 2n^{-\rho_i k/60} - 3n^{-\rho_d k/60}$:

$$i_2 - f_{A^*}(i_1 + k\ln n) = [i_2 - f_{A^*}(i_1) - k\ln n] + [f_{A^*}(i_1) + k\ln n - f_{A^*}(i_1 + k\ln n)] \geq$$

$$\frac{3}{2}\kappa_n \cdot k\ln n - \frac{3}{2}\kappa_n \cdot k\ln n = 0.$$

Then since $i_2 > f_{A^*}(i_1 + k\ln n)$, none of the bits are in s_2' are inherited from bits in s_1'. In either case, s_1', s_2' are independent and uniformly random bitstrings, and we can apply Lemma 20 with $D = kr\ln n$ to get the lemma by a union bound. ◀

Let n_0 be a sufficiently large constant. If we choose any r which is less than a certain constant (which is approximately .1569), for all $n \geq n_0$, if k is sufficiently large then the term $\frac{(\frac{4e}{r}+5e+\frac{4e}{kr\ln n})^r}{2}$ from Corollary 23 is less than 1 and thus the failure probability in Corollary 23 becomes $n^{-\Omega(k)}$. If for all $n \geq n_0$, $(1 + \frac{3}{2}k(\rho_s + 2\kappa_n)) < kr$, then for all $n \geq n_0$ the lower bound on edit distance given by Corollary 23 exceeds the upper bound given by Corollary 22. In turn, informally we have the desired property that we can use the edit distance between substrings of length $k\ln n$ in s_1 and s_2 to test if these substrings are close in the canonical alignment. So for the rest of this section, we will fix $\rho_s, \rho_i, \rho_i', \rho_d, \rho_d', r$ to be positive values satisfying these conditions for all $n \geq n_0$. Once these values are fixed we can make the failure probabilities in both corollaries n^{-c} with any exponent c of our choice ($c = 2$ will suffice to achieve a final failure probability of $O(1/n)$) by choosing a sufficiently large k depending only on c and n_0. So we also fix k to be said sufficiently large value.

4.2 Algorithm for Quickly Finding an Approximate Alignment

We now describe the algorithm ApproxAlign, given as Algorithm 1, which finds the approximate alignment f'. Informally, ApproxAlign runs as follows: It starts by initializing $f'(1) = 1$, which is of course exactly correct. By Lemma 21, we know that $f_{A^*}(k\ln n + 1)$ will be within $O(\ln n)$ of $1 + k\ln n$. So, to decide what $f'(k\ln n + 1)$ will be, we compute the edit distance between bits $k\ln n + 1$ to $2k\ln n$ of s_1 and bits j to $j + k\ln n - 1$ of s_2 for various values of j close to $1 + k\ln n$. By Corollary 22 we know that when j is near $f_{A^*}(k\ln n + 1)$, the edit distance will be small, and by Corollary 23 we know that when j is far from $f_{A^*}(k\ln n + 1)$ the edit distance will be large. So whichever value of j causes the edit distance to be minimized is not too far from the true value of $f_{A^*}(k\ln n + 1)$. Once we've decided on the value $f'(k\ln n + 1)$, we proceed analogously to choose a value for $f'(2k\ln n + 1)$, using $f'(k\ln n + 1)$ to decide what range of values try, and so on. We now formally prove our guarantee for ApproxAlign (including the runtime guarantee).

▶ **Lemma 24.** *For $(s_1, s_2, \mathcal{E}) \sim ID(n)$, ApproxAlign$(s_1, s_2)$ computes in time $O(n\ln n)$ a function f' such that with probability at least $1 - n^{-\Omega(1)}$, for all i where $f'(i)$ is defined $|f'(i) - f_{A^*}(i)| \leq \lceil(\frac{3}{2}\kappa_n + 1)k\ln n\rceil$.*

Proof. We proceed by induction. Clearly, $|f'(1) - f_{A^*}(1)| = |1 - 1| \leq \lceil(\frac{3}{2}\kappa_n + 1) \cdot k\ln n\rceil$. Suppose $|f'((i-1)k\ln n + 1) - f_{A^*}((i-1)k\ln n + 1)| \leq \lceil(\frac{3}{2}\kappa_n + 1) \cdot k\ln n\rceil$. By Lemma 21 and our choices of constants, with probability $1 - n^{-\Omega(1)}$, $|f_{A^*}((i-1)k\ln n + 1) + k\ln n - f_{A^*}(ik\ln n + 1)| \leq \frac{3}{2}\kappa_n \cdot k\ln n$. This gives:

Algorithm 1 Algorithm for Approximate Alignment.

```
function ApproxAlign($s_1$, $s_2$)
    $f'(1) \leftarrow 1$
    $J \leftarrow 2\lceil(\frac{3}{2}\kappa_n + 1) \cdot k\rceil$
    for $i = 1, 2, \ldots \lfloor\frac{n}{k \ln n}\rfloor - 1$ do
        $minED \leftarrow \infty$
        for $j = -J, -J+1, \ldots J$ do
            $s_1' \leftarrow$ bits $ik \ln n + 1$ to $(i+1)k \ln n$ of $s_1$
            $s_2' \leftarrow$ bits $f'((i-1)k \ln n + 1) + (j+k) \ln n$ to $f'((i-1)k \ln n + 1) + (j+2k) \ln n - 1$ of $s_2$
            if $ED(s_1', s_2') \leq minED$ then
                $minED \leftarrow ED(s_1', s_2')$
                $f'(ik \ln n + 1) \leftarrow f'((i-1)k \ln n) + (j+k) \ln n$
            end if
        end for
    end for
    return $f'$
end function
```

$$
\begin{aligned}
&|[f'((i-1)k \ln n + 1) + k \ln n] - f_{A^*}(ik \ln n + 1)| \leq \\
&|[f'((i-1)k \ln n + 1) + k \ln n] - [f_{A^*}((i-1)k \ln n + 1) + k \ln n]| \\
&\qquad + |[f_{A^*}((i-1)k \ln n + 1) + k \ln n] - f_{A^*}(ik \ln n + 1)| = \\
&|f'((i-1)k \ln n + 1) - f_{A^*}((i-1)k \ln n + 1)| \\
&\qquad + |f_{A^*}((i-1)k \ln n + 1) + k \ln n - f_{A^*}(ik \ln n + 1)| \leq \\
&\left\lceil (\frac{3}{2}\kappa_n + 1) \cdot k \ln n \right\rceil + \frac{3}{2}\kappa_n \cdot k \ln n \leq J.
\end{aligned}
$$

So for some j in the range iterated over by the algorithm, $|f'((i-1)k \ln n + 1) + (j+k) \ln n - f_{A^*}(ik \ln n + 1)| \leq \ln n$ and thus the minimum edit distance $minED$ found by the algorithm in iterating over the j values is at most $(1 + \frac{3}{2}k(\rho_s + 2\kappa_n) \ln n < kr \ln n$ by Corollary 22 with probability at least $1 - n^{-\Omega(1)}$. By Corollary 23, with probability at least $1 - n^{-\Omega(1)}$ the final value of $f'(ik \ln n + 1)$ can't differ from $f_{A^*}(ik \ln n + 1)$ by more than $\lceil(\frac{3}{2}\kappa_n + 1) \cdot k \ln n\rceil$ as desired - otherwise, by the corollary with high probability $minED$ would be larger than $kr \ln n$.

Thus by induction, $|f'(i) - f_{A^*}(i)| \leq \lceil(\frac{3}{2}\kappa_n + 1) \cdot k \ln n\rceil$ for all i if the high probability events of Lemma 21, Corollary 22, and Corollary 23 occur in all inductive steps. Across all inductive steps we require $O(n)$ such events to occur, and each occurs with probability $1 - n^{-\Omega(1)}$ where the negative exponent can be made arbitrarily large, so by a union bound we can conclude that with probability $1 - n^{-\Omega(1)}$, $|f'(i) - f_{A^*}(i)| \leq 2k \ln n$ for all i.

For runtime, note that the for loops iterate over $O(\frac{n}{\ln n})$ values of i and $O(1)$ values of j. For each i, j pair, we perform an edit distance computation between two strings of length $O(\ln n)$ which can be in done in $O(\ln^2 n)$ time using the canonical dynamic programming algorithm. So the overall runtime is $O(n \ln n)$. ◀

5 Error Analysis with Indels

In this section, we extend the results from Section 3 to the case where indels are present.

▶ **Lemma 25.** *For any realization of $(s_1, s_2, \mathcal{E}) \sim ID(n)$, let s_1' be the restriction of s_1 to any fixed subset of indices B of total size $\ell \geq k \ln n$, s_2' be the substring of s_2 that A^* aligns with s_1', and let $(A^*)_B$ denote the restriction of the alignment A^* to indices in s_1', s_2'. Then with probability $1 - e^{-\Omega(\ell)}$ over $(s_1, s_2, \mathcal{E}) \sim ID(n)$, $A > (A^*)_B$ for all alignments A of s_1', s_2' such that A and $(A^*)_B$ do not share any edges.*

The proof is almost identical to that of Lemma 14. We defer the proof to Appendix A. We remark that the resulting constraint on the mutation probailities is given by $\frac{3}{2}\rho_s + \kappa_n < .03485$.

▶ **Lemma 26.** *For $(s_1, s_2, \mathcal{E}) \sim ID(n)$, with probability $1 - n^{-\Omega(1)}$ for all alignments A in the range of $\mathcal{SBR}$, $A \geq A^*$.*

Proof. The proof proceeds similarly to that of Lemma 17. Recall that the starting/ending indices and the lengths of breaks are defined with respect to the indices in s_1. Since s_1's length is n always, we can define sets of break points independently of the realization of $ID(n)$, and so we define $\mathcal{A}_i$, $\mathcal{B}_i$, $\mathcal{A}_B$ as in Lemma 17. The restriction of s_1 to a fixed subset of indices in the statement 25 can be applied to the subsets of indices contained in breaks, so the same analysis as in Lemma 17 gives:

$$Pr[\exists A \in \mathcal{SBR}(\mathcal{A}), A < A^*] = \sum_{i=1}^{\frac{n}{k \ln n}} |\mathcal{B}_i| \, n^{-\Omega(ik)}.$$

Since breakpoints are defined with respect to the fixed-length string s_1, as before we have $|\mathcal{B}_i| \leq (nk \ln n + 1)^i$ and thus $Pr[\exists A \in \mathcal{SBR}(\mathcal{A}), A < A^*] \leq n^{-\Omega(k)}$ as desired. ◀

Proof of Theorem 2. We estimate f_{A^*} using ApproxAlign to obtain f'. Then, we use the standard DP algorithm restricted to entries that are within distance $k_2 \ln n$ from $(i, f'(i))$ for some i. By Theorem 24, for any fixed k, if k_2 is sufficiently large, this range of entries computed contains all entries within distance $k \ln n$ of A^*, i.e. contains the range of $\mathcal{LBR}$. Fact 15 and Corollary 16 also hold when indels are present, so by Lemma 26, the optimality of the DP algorithm gives that the algorithm is correct.

For runtime, note that ApproxAlign runs in $O(n \ln n)$ time per Theorem 24 and the set of entries considered by the DP algorithm is size at most $O(n \ln n)$ (each of the $n/\ln n$ indices where f' is defined contribute $O(\ln^2 n)$ entries to be computed), and each entry can be computed in constant time. So the overall runtime is $O(n \ln n)$ as desired. ◀

References

1 S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215(3):403–410, October 1990. doi:10.1016/s0022-2836(05)80360-2.

2 Alexandr Andoni, Mark Braverman, and Avinatan Hassidim. Phylogenetic reconstruction with insertions and deletions. *Preprint*, 2010.

3 Alexandr Andoni, Constantinos Daskalakis, Avinatan Hassidim, and Sebastien Roch. Global alignment of molecular sequences via ancestral state reconstruction. *Stochastic Processes and their Applications*, 122(12):3852–3874, 2012. doi:10.1016/j.spa.2012.08.004.

4 Alexandr Andoni and Robert Krauthgamer. The smoothed complexity of edit distance. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming*, pages 357–369, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. doi:10.1145/2344422.2344434.

5 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. *Polylogarithmic Approximation for Edit Distance and the Asymmetric Query Complexity*, pages 244–252. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. doi:10.1007/978-3-642-16367-8_16.

6 Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. *SIAM Journal on Computing*, 41(6):1635–1648, 2012. doi:10.1137/090767182.

7 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, page 51–58, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2746539.2746612.

8 Tugkan Batu, Funda Ergün, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami. A sublinear algorithm for weakly approximating edit distance. In *STOC*, 2003. doi:10.1145/780542.780590.

9 Tugkan Batu, Funda Ergün, and Süleyman Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 792–801, 2006. doi:10.5555/1109557.1109644.

10 K. Bringmann and M. Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 79–97, 2015. doi:10.1109/FOCS.2015.15.

11 D. Chakraborty, D. Das, E. Goldenberg, M. Koucky, and M. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 979–990, 2018. doi:10.1109/FOCS.2018.00096.

12 Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh Vazirani. *Algorithms*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 2008. doi:10.1016/j.cosrev.2008.03.001.

13 Constantinos Daskalakis and Sebastien Roch. Alignment-free phylogenetic reconstruction. In *Annual International Conference on Research in Computational Molecular Biology*, pages 123–137. Springer, 2010. doi:10.1007/978-3-642-12683-3_9.

14 Martin C. Frith. Large-scale sequence comparison: Spaced seeds and suffix arrays, 2008. URL: http://last.cbrc.jp/mcf-kyoto08.pdf.

15 Martin C Frith. How sequence alignment scores correspond to probability models. *Bioinformatics*, 36(2):408–415, July 2019. doi:10.1093/bioinformatics/btz576.

16 Arun Ganesh and Qiuyi (Richard) Zhang. Optimal sequence length requirements for phylogenetic tree reconstruction with indels. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, page 721–732, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316345.

17 Paweł Gawrychowski. Faster algorithm for computing the edit distance between slp-compressed strings. In Liliana Calderón-Benavides, Cristina González-Caro, Edgar Chávez, and Nivio Ziviani, editors, *String Processing and Information Retrieval*, pages 229–236, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-34109-0_24.

18 Nina Holden, Robin Pemantle, and Yuval Peres. Subpolynomial trace reconstruction for random strings and arbitrary deletion probability. In *COLT*, 2018. URL: http://proceedings.mlr.press/v75/holden18a.html.

19 Thomas Holenstein, Michael Mitzenmacher, Rina Panigrahy, and Udi Wieder. Trace reconstruction with constant deletion probability and related results. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 389–398, January 2008. doi:10.1145/1347082.1347125.

20 Kazutaka Katoh, Kazuharu Misawa, Keiichi Kuma, and Takashi Miyata. MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Research*, 30(14):3059–3066, July 2002. doi:10.1093/nar/gkf436.

21 William Kuszmaul. Efficiently approximating edit distance between pseudorandom strings. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, page 1165–1180, USA, 2019. Society for Industrial and Applied Mathematics.

22 G. Landau, E. Myers, and J. Schmidt. Incremental string comparison. *SIAM Journal on Computing*, 27(2):557–582, 1998. doi:10.1137/S0097539794264810.

23 William J. Masek and Michael S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20:18–31, February 1980. doi:10.1016/0022-0000(80)90002-1.

24 Fedor Nazarov and Yuval Peres. Trace reconstruction with exp(o(n1/3)) samples. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 1042–1046, New York, NY, USA, 2017. ACM. doi:10.1145/3055399.3055494.

25 W R Pearson and D J Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448, 1988. doi:10.1073/pnas.85.8.2444.

A Deferred Proofs

A.1 Proof of Lemma 6

Proof. We consider two cases for the realization of m and apply tail bounds to each case:

$$Pr\left[X \geq k \cdot \frac{\mu}{q}\right] = Pr\left[X \geq k \cdot \frac{\mu}{q} \wedge m \geq \sqrt{k}\mu\right] + Pr\left[X \geq k \cdot \frac{\mu}{q} \wedge m < \sqrt{k}\mu\right]$$

$$\leq Pr\left[m \geq \sqrt{k}\mu\right] + Pr\left[X \geq k \cdot \frac{\mu}{q} | m \leq \sqrt{k}\mu\right].$$

A Chernoff bound gives $Pr[m \geq \sqrt{k}\mu] \leq e^{-\frac{(\sqrt{k}-1)^2\mu}{3}}$, the negative binomial tail bound (and noticing that $Pr[X \geq k \cdot \frac{\mu}{q} | m \leq \sqrt{k}\mu]$ is maximized when $m = \sqrt{k}\mu$) gives $Pr[X \geq k \cdot \frac{\mu}{q} | m < \sqrt{k}\mu] \leq e^{-\frac{\sqrt{k}\mu(1-1/\sqrt{k})^2}{2}}$, giving the lemma. ◀

A.2 Proof of Lemma 20

Proof. We first bound the number of strings within edit distance D of s_1. Fix any set of up to D edits that can be applied to a bitstring initially of length $k \ln n$, that does not contain redundant edits (such as substituting the same bit more than once, deleting an inserted bit). This set can be mapped to a set of D tuples as follows:

- For a substitution (or deletion) applied to the bit in the ith position (using the indexing prior to insertions and deletions), it is encoded as the tuple (i, S) (or (i, D) for a deletion). Note that by the assumption that there are no redundant edits, all substitution and deletion edits in the set of edits map to distinct tuples.
- For insertions, we handle indexing differently to still ensure no two insertions are mapped to the same tuple. Suppose the set of D edits inserts the bitstring $b_1 b_2 \ldots b_k$ to the right of index of i (using the original indexing - we treat bits are being inserted to the left of the entire bitstring as being inserted to the right of bit 0). Let i' be i plus the number of insertions in the set of edits occurring before bit i. Then we map these k insertions to the tuples $(i', I_{b_1}), (i'+1, I_{b_2}) \ldots (i'+k-1, I_{b_k})$. This ensures that the insertions in the set of edits also get mapped to different tuples, since the first index will be distinct for all tuples that insertions are mapped to.
- If the number of edits is $D-k$, we include $(1, N), (2, N) \ldots (k, N)$ in the final set of tuples so the final set of tuples still has size D.

Every tuple that can be mapped to in this encoding scheme is of the form (i, E) for $0 \leq i \leq k \ln n + D, E \in \{S, D, I_0, I_1\}$ or (i, N) for $1 \leq i \leq D$. So, there are at most $\binom{4k \ln n+5D+4}{D}$ sets of D tuples that any set of up to D edits can be mapped to. Furthermore, note that the mapping is injective, i.e. given a set of D tuples, using the reverse of the above process it can be uniquely mapped to set of edits. So, there are also at most $\binom{4k \ln n+5D+4}{D}$ possible ways to apply at most D edits to a bitstring which is initially length $k \ln n$. Stirling's approximation gives that this is at most $(4e\frac{k \ln n}{D} + 5e + \frac{4e}{D})^D$. So there are at most $(4e\frac{k \ln n}{D} + 5e + \frac{4e}{D})^D$ strings s' such that $ED(s_1, s') \leq D$. The number of bitstrings of length $k \ln n$ is $2^{k \ln n}$. So the probability $ED(s_1, s_2) \leq D$ is at most $\frac{(4e\frac{k \ln n}{D}+5e+\frac{4e}{D})^D}{2^{k \ln n}}$. ◀

A.3 Proof of Lemma 25

Proof. We proceed similarly to Lemma 14, but for the case with indels. The same analysis as in Lemma 19 gives that that for a fixed s_1',

$$\Pr_{(s_1,s_2,\mathcal{E})\sim ID(n)}[(A^*)_B \geq \frac{3}{2}\left(\rho_s + \kappa_n\right)\ell] \leq e^{-\rho_s \ell/12} + 2e^{-\rho_i \ell/60} + 3e^{-\rho_d \ell/60}.$$

Our goal now is to show any alignment A of s_1', s_2' that shares no edges with $(A^*)_B$ has $A > c\ell$ with high probability, where $c = \frac{3}{2}\rho_s + \kappa_n$.

Fix any realization ζ of the positions of indels generated by $(s_1, s_2, \mathcal{E}) \sim ID(n)$, without fixing the values of s_1, the inserted bits, or the positions of substitutions. Let $\ell_1 = \ell$ and ℓ_2 be the lengths of s_1' and s_2'. Let $r = |\ell_1 - \ell_2|$. A similar analysis to Lemma 21 gives that $r \leq \kappa_n \ell$ with probability $1 - e^{-\Omega(\ell)}$, so it suffices to prove the lemma statement holds with high probability conditioned on any ζ such that $r < \kappa_n \ell$, so we condition on ζ for the rest of the proof. Assume without loss of generality that $\ell_2 - \ell_1 = r$, i.e. that the r excess indels are insertions. The counting argument is similar when $\ell_1 - \ell_2 = r$. As before, we sum over the number of deletions, d, which corresponds to $d + r$ insertions and $\ell - d$ substitutions.

$$\begin{aligned} Pr[\exists A, A \leq c\ell] &\leq \sum_{d=0}^{c\ell/2} \sum_{A \in \mathcal{A} \text{ with } d \text{ deletions}} Pr[A \leq c\ell] \\ &\leq \sum_{d=0}^{c\ell/2} \binom{\ell + d + r}{d, d + r, \ell - d} Pr[Binom(\ell - d, \frac{1}{2}) \leq c\ell - 2d - r] \\ &\leq \frac{c\ell}{2} \binom{(1 + \frac{c}{2})\ell + r}{\frac{c}{2}\ell, \frac{c}{2}\ell + r, (1 - \frac{c}{2})\ell} Pr[Binom((1 - \frac{c}{2})\ell, \frac{1}{2}) \leq c\ell]. \end{aligned}$$

Where the probability is taken over the events we haven't conditioned on, i.e. the realization of s_1, the inserted bits, and the positions of substitutions. Since we assume $r < \kappa_n \ell$, then $\binom{(1+\frac{c}{2})\ell+r}{\frac{c}{2}\ell, \frac{c}{2}\ell+r, (1-\frac{c}{2})\ell} \leq \binom{(1+\frac{c}{2}+\kappa_n)\ell}{\frac{c}{2}\ell, (\frac{c}{2}+\kappa_n)\ell, (1-\frac{c}{2})\ell}$ with high probability. Note also that $\kappa_n \leq \frac{3}{2}\rho_s + \kappa_n < c$. Hence, similar to Equation (3) from Lemma 14, Stirling's approximation gives an upperbound on the trinomial

$$\binom{(1 + \frac{c}{2} + \kappa_n)\ell}{\frac{c}{2}\ell, (\frac{c}{2} + \kappa_n)\ell, (1 - \frac{c}{2})\ell} \leq \frac{e}{(2\pi)^{3/2}} \frac{2}{c\ell} \sqrt{\frac{2 + 3c}{2 - c}} \left[\frac{(1 + \frac{3}{2}c)^{(1+\frac{3}{2}c)}}{(\frac{c}{2})^c (1 - \frac{c}{2})^{(1-\frac{c}{2})}} \right]^{\ell}.$$

We combine this with Equation (2) from Lemma 14 for the term $Pr[Binom((1 - \frac{c}{2})\ell, \frac{1}{2}) \leq c\ell]$, to get that when $c < 0.03485$, the probability decays exponentially in ℓ. Hence requiring that $\frac{3}{2}\rho_s + \kappa_n < .03485$ ensures that $A > (A^*)_B$ with high probability. ◀

Fold Family-Regularized Bayesian Optimization for Directed Protein Evolution

Trevor S. Frisby
Computational Biology Department, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
tfrisby@andrew.cmu.edu

Christopher J. Langmead[1]
Computational Biology Department, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
http://www.cs.cmu.edu/~cjl
cjl@cs.cmu.edu

Abstract

Directed Evolution (DE) is a technique for protein engineering that involves iterative rounds of mutagenesis and screening to search for sequences that optimize a given property (ex. binding affinity to a specified target). Unfortunately, the underlying optimization problem is under-determined, and so mutations introduced to improve the specified property may come at the expense of unmeasured, but nevertheless important properties (ex. subcellular localization). We seek to address this issue by incorporating a fold-specific regularization factor into the optimization problem. The regularization factor biases the search towards designs that resemble sequences from the fold family to which the protein belongs. We applied our method to a large library of protein GB1 mutants with binding affinity measurements to IgG-Fc. Our results demonstrate that the regularized optimization problem produces more native-like GB1 sequences with only a minor decrease in binding affinity. Specifically, the log-odds of our designs under a generative model of the GB1 fold family are between 41 − 45% higher than those obtained without regularization, with only a 7% drop in binding affinity. Thus, our method is capable of making a trade-off between competing traits. Moreover, we demonstrate that our active-learning driven approach reduces the wet-lab burden to identify optimal GB1 designs by 67%, relative to recent results from the Arnold lab on the same data.

2012 ACM Subject Classification Computing methodologies → Active learning settings; Computing methodologies → Machine learning; Applied computing → Computational proteomics; Mathematics of computing → Discrete optimization

Keywords and phrases Protein design, Bayesian Optimization, Gaussian Process Regression, Regularization

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.18

Funding This research is supported by an NIH T32 training grant T32 EB009403 as part of the HHMI-NIBIB Interfaces Initiative.

1 Introduction

The field of protein engineering seeks to design molecules with novel or improved properties [15]. The primary techniques used in protein engineering fall into two categories: *rational design* [22] and *directed evolution* (DE) [1]. Rational design uses model-driven *in silico* combinatorial searches to identify promising candidate designs, which are then synthesized and tested experimentally. Directed evolution, in contrast, involves iterative rounds of saturation mutagenesis at select residue positions, followed by *in vitro* or *in vivo* screening

[1] Corresponding author: cjl@cs.cmu.edu

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti; Article No. 18; pp. 18:1–18:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

for desirable traits. The most promising sequences are then isolated and used to seed the next round of mutagenesis. Traditionally, directed evolution is a model-free approach. That is, computational models are not used to guide or simulate mutagenesis.

Recently, a technique for incorporating Machine Learning (ML) into the DE workflow was introduced [32]. Briefly, this ML-assisted form of DE uses the screening data from each round to update a model that predicts the effects of mutations on property being optimized, $f(s_k) \rightarrow y$. Here, y is the measured trait, s_k is the choice of residues at $k \ll n$ positions, and n is the length of the protein's primary sequence. The mutagenesis step in the current round of DE is then biased towards generating sequences with high predicted fitness under the model, as opposed to generating a uniformly random sample. ML-assisted DE has been shown to reduce the number of rounds needed to find optimal sequences, relative to traditional (i.e., model-free) DE [32].

Significantly, the models learned in ML-assisted DE are *myopic* in the sense that they only consider the relationship between s_k and the screened trait, y (ex binding affinity). Therefore, the underlying optimization problem is under-determined, and so the technique may improve the measured trait at the expense of those that are unmeasured, but nevertheless important (ex. thermostability, solubility, subcellular localization, etc). The primary goal of this paper is to introduce an enhanced version of ML-assisted DE that is biased towards native-like designs, while optimizing the desired trait. By "native" we mean that the optimized design still has high probability under a generative model of the fold family to which the protein belongs. The intuition behind this approach is that any high-probability sequence is likely to respect factors that are not directly accounted for by the fitness model, f, such as epistatic interactions between the mutated residues and the rest of the protein [27], among others.

Our method performs Bayesian optimization [16] and incorporates a regularization factor derived from a generative model of the fold family. Here, we evaluate two choices of generative models: Markov Random Fields (MRF) generated by the GREMLIN algorithm [2], and profile HMMs [11]. We demonstrate our method by re-designing the B1 domain of streptococcal protein G (GB1) to maximize binding affinity to the IgG Fc receptor. Our results demonstrate that the regularization term leads to more native-like GB1 sequences, with minimal impact on binding affinity. Like previous studies, our results also show that ML-assisted DE outperforms the traditional, model-free approach to DE. Additionally, we demonstrate that our approach reduces the wet-lab burden to identify optimal GB1 designs by 67%, relative to recent results from previous results on the same data [32].

2 Background

The method introduced in this paper combines several technologies: *directed protein evolution*, *Bayesian optimization*, and *generative modeling of protein fold families*. The following subsections provide brief summaries of these techniques.

2.1 Directed Protein Evolution

Directed evolution (DE) is an iterative technique for designing molecules. It has been used to create proteins with increased stability (ex. [6]), improved binding affinity (ex. [9]), to design new protein folds (ex. [7]), to change an enzyme's substrate specificity (ex. [26]) or ability to selectively synthesize enantiomeric products (ex. [32]), and to study fitness landscapes ([24]), among others. Given an initial sequence, the primary steps in directed evolution are: (i) *mutagenesis*, to create a library of variants; (ii) *screening*, to identify variants with the desired traits; and (iii) *amplification* of the best variants, to seed the next round. Each step

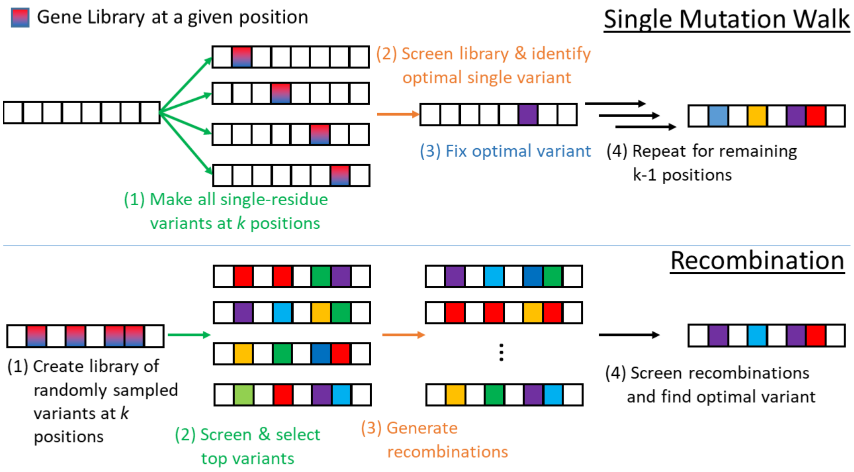

Gene Library at a given position
Single Mutation Walk
(2) Screen library & identify optimal single variant
(3) Fix optimal variant
(4) Repeat for remaining k-1 positions
(1) Make all single-residue variants at k positions
Recombination
(1) Create library of randomly sampled variants at k positions
(2) Screen & select top variants
(3) Generate recombinations
(4) Screen recombinations and find optimal variant

Figure 1 Traditional, model-free approaches to directed evolution: (*Top*) The "single mutation walk" approach to directed evolution. The library of variants is the union of k libraries created by performing saturation mutagenesis at a single location. The resulting library, therefore, has $20k$ variants. The library is screened to find the single variant that optimizes the measured trait. That variant is fixed and the procedure is repeated for the remaining $k-1$ positions. (*Bottom*) The library of variants is created by performing saturation mutagenesis at k positions. The top variants are identified through screening. Those variants are randomly recombined to generate a second library, which is then screened to find the top design.

can be performed in a variety of ways, giving rise to multiple options for performing DE. For example, the mutagenesis step can be performed one residue at a time, called a single mutation walk (Fig. 1-top), or simultaneously at multiple positions, followed by genetic recombination (Fig. 1-bottom).

Rational design and directed evolution have complementary strengths and weaknesses, and in practice it is not unusual for protein engineers to use both. The computational models used in rational design are typically physics- or knowledge based [4], and will therefore filter designs that are predicted to be energetically or statistically unfavorable. This filtering, in turn, may reduce the total number of designs that need to be synthesized/cloned and tested in the lab. Directed evolution, in contrast, performs what is in effect a parallel *in vitro* or *in vivo* search over designs. Most of the designs will lack the desired trait, which is inefficient in the sense that consumable resources are wasted. On the other hand, DE's approach results in the screening of a larger number of designs than rational design, because it generates large libraries of variants, as opposed to individual designs. That is, DE may ultimately be the faster route to finding optimal designs due to its parallel and high throughput nature, which may also increase the odds of serendipitous discoveries. Moreover, the traditional approach to DE is model-free, and therefore not subject to the limitations of computational models which are, at best, only approximations to the underlying physics and are not intended to reflect phenomena at higher scales (i.e., chemistry and biology).

Machine Learning-assisted directed protein evolution

While effective, the mutagenesis, screening, and amplification steps in DE are expensive and time-consuming, relative to rational design's *in silico* screens. In an effort to reduce these experimental demands, a Machine Learning-assisted approach to DE was introduced recently [32]. This ML-assisted form of DE is summarized in Fig. 2. The key difference between

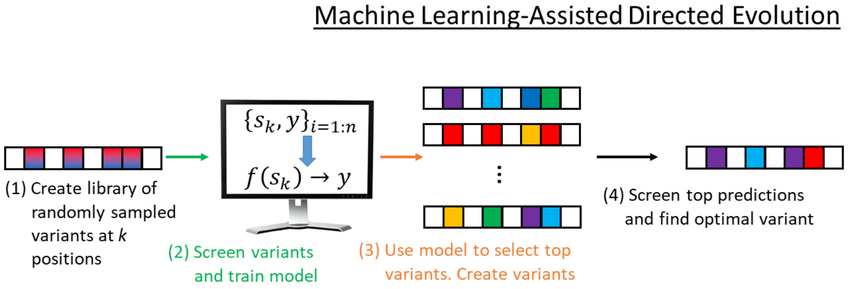

Machine Learning-Assisted Directed Evolution
$\{s_k, y\}_{i=1:n}$
$f(s_k) \rightarrow y$
(1) Create library of randomly sampled variants at k positions
(2) Screen variants and train model
(3) Use model to select top variants. Create variants
(4) Screen top predictions and find optimal variant

Figure 2 Machine Learning-assisted directed evolution: The first step in ML-assisted DE is the same as for traditional DE (see Fig 1). A library of variants is created via mutagenesis. Existing data, $\mathcal{S} = \{s_k, y\}_{i=1:n}$ are used to train a classifier or regression model, $f(s_k) \rightarrow y$, which is then used to rank variants via an *in silico* screen. The top variants are then synthesized/cloned and screened using *in vitro* or *in vivo* assays. The data from the ith round is added to $\mathcal{S}$ and used in subsequent DE rounds.

traditional and ML-Assisted DE is that the data generated during screening, $\mathcal{S} = \{s_k, y\}_{i=1:n}$ are used to train a model, $f(s_k) \rightarrow y$, that maps the set of mutations to the trait. The model, f, can be a classifier or regression model, and is used to perform an *in silico* screen over designs. Promising designs are then synthesized/cloned and screened in the lab. The key assumption made by ML-assisted DE is that the cost of performing an *in silico* screen is much lower than running wet-lab experiments.

Like rational design, ML-assisted DE uses computational models, but the nature of those models is rather different. For one, the models used in ML-assisted DE make predictions corresponding to the quantity measured in the screening step, whereas the models used in rational design tend to be based on physical or statistical energy functions, and are therefore making predictions about the energetic favorability of the design. Second, the models used in ML-assisted DE are updated after each DE round to incorporate the new screening data, and thus adapt to protein-specific experimental observations. The models used in rational design, in contrast, are typically fixed. Finally, the models used in ML-assisted DE are myopic, in the sense that they only consider the relationship between a small subset of sequence positions and the measured quantity. The models used in rational design, in contrast, generally consider the *entire* sequence, and are thus better suited to filtering energetically unfavorable designs. The technique introduced in this paper seeks to combine the strengths of both methods; our method uses a fitness model that adapts to the experimental data, but also considers the favorability of the mutations across the entire sequence.

2.2 Bayesian Optimization

Bayesian optimization [16] is a technique for optimizing black-box objective functions, f. It has been used in a variety of contexts, including robotics (ex. [14]), particle physics (ex. [10]), and hyper-parameter optimization in deep learning (ex [3]). The first published form of ML-assisted DE [32] did not employ Bayesian optimization, but the same lab subsequently introduced a version that does [33].

In the Bayesian optimization framework, our goal is to find x^*, the point that maximizes (or minimizes) $f(x)$. This is challenging because f is both unknown and expensive to evaluate. Therefore, we want to minimize the number of times $f(x)$ is evaluated. Because it is unknown, f is treated as a random *function* with a suitably defined prior. Given experimental observations, a posterior distribution over f is computed, which becomes the prior for the next round. Typical choices for priors/posteriors include Gaussian Processes [21] and Tree-structured Parzen estimators [3]. In the context of this paper, f is the function that maps designs to fitness values. The evaluation of f is expensive, because it requires the previously described DE mutagenesis and screening steps.

The posterior over f becomes the input to an *acquisition function*, which is used to select the next point(s) to evaluate [30]. A variety of acquisition functions have been proposed, including: expected improvement (EI), upper (or lower) confidence bounds, Thompson sampling [28], and probability of improvement. In general, an acquisition function defines some trade-off between *exploring* the design space, and simply selecting the point that has the best expected value under the posterior (aka *exploitation*). In this paper, we seek to maximize f, and use expected improvement criterion as the acquisition function.

2.3 Generative modeling of protein fold families

The proposed approach uses a modified acquisition function that considers not only the expected improvement in fitness for a given set of mutations, s_k, but also whether the overall design, s_n, resembles proteins within the same fold family. The latter criterion is implemented using a regularization term, as described in Sec. 3. Our assumption is that the statistical properties of the proteins within a given fold family have been optimized through natural evolution to ensure that they have a full range of physical, chemical, and biological properties to function properly in a complex cellular environment. Therefore, during the process of protein engineering, we should seek designs that are as native-like as possible [8, 12, 17].

To accomplish this task, we propose to use fold family-specific generative models of sequences. We evaluate two options for such models – profile HMMs and Markov Random Fields (MRF), as generated by the GREMLIN algorithm [2]. HMM and MRF models can be learned from known sequences from a given fold family. The primary difference between these models is that whereas HMMs only encode dependencies between adjacent residues, the GREMLIN algorithm can detect and encode both sequential and long-range dependencies. Either way, the models encode a joint distribution over residue types at each position in the primary sequence, $P(S_1, ..., S_n)$, which can be used to compute the probability (or related quantities, like log-odds) of given design. We assume that any design with a high probability or log-odds under the generative model is native-like.

3 Methods

3.1 Generative models of Protein G IgG Fc binding domain (GB1)

Our method incorporates generative models of protein sequences for the fold family to which the target protein belongs. We evaluated two options for such models, (i) a Markov Random Field (MRF) model learned using the GREMLIN algorithm [2], and (ii) a profile Hidden Markov Model (HMM). We downloaded the profile HMM [11] for the fold family to which GB1 belongs (Pfam id: PF01378) from the Pfam database [5]. We also downloaded the multiple sequence alignment that was used to train the HMM from Pfam, and then used the alignment to train the GREMLIN model. Thus, the GREMLIN and HMM models were trained from the same sequence data. We used these models to compute the log-odds of each design. These log-odds are used as a regularization factor in the Bayesian optimization (see Sec 3.2). The two models make different assumptions about the conditional independencies among the residues in the distribution over GB1 sequences, and thus will output different log-odds scores for the same design, in general.

3.2 Fold family-regularized Bayesian Optimization for Directed Protein Evolution

The Bayesian optimization is performed using Gaussian Process (GP) regression as the prior over the unknown fitness function, f. A GP requires a kernel, K, which describes the covariance between sequences s_i and s_j. In our models, we use the squared exponential kernel given by:

$$K_{s_i,s_j} = \exp\left(- \frac{d(s_i, s_j)^2}{2\ell^2} \right)$$

where $d(\cdot,\cdot)$ is the Euclidean distance and ℓ the length scale. A one-hot encoding of the variants at the selected positions was used to compute distances. Hyperparameter θ is optimized while fitting the GP to data by maximizing the log marginal likelihood:

$$\log p(y|\theta) = -\frac{N}{2}\log 2\pi - \frac{1}{2}\log\det|K + \sigma^2 I| - \frac{1}{2}y^T(K + \sigma^2 I)^{-1}y$$

The term y is a vector of the given property (e.g. fitness) of N sequences, σ^2 the variance of observations, and I is the $N \times N$ identity matrix. Once fitted, the GP encodes a distribution, $\mathcal{P}$, which is used to obtain a posterior mean function $\mu_{\mathcal{P}}(s_k)$ and variance over the unknown fitness function f:

$$\begin{aligned} \mu_{\mathcal{P}}(s_k) = \mathbb{E}[f(s_k)] &= K_{s_k,s}(K + \sigma^2 I)^{-1}y \\ \mathrm{Var}[f(s_k)] &= K_{s_k,s_k} - K_{s_k,s}(K + \sigma^2 I)^{-1}K_{s,s_k} \end{aligned}$$

$K_{s_k,s}$ refers to the row vector of kernel function values between sequence s_k and all other sequences, denoted by subscript s. Additionally, $K_{s,s_k} = K^T_{s_k,s}$.

The GP becomes the argument to an acquisition function, which is used to select sequences for wet-lab screening. The data produced via the screening step are used to update the GP for the next round. In our experiments, we used the expected improvement (EI) criterion as our acquisition function. Two versions of EI were considered: (i) the standard version, which is often used in Bayesian optimization, and (ii) a regularized form of EI. The standard form of EI is given by:

$$\mathrm{EI}(s_k;\mathcal{P}) = \mathbb{E}_{\mathcal{P}}\big[\max(0, f(s_k) - \mu_{\mathcal{P}}(x^+)\big] \tag{1}$$

where x^+ is the location of the (estimated) optimal posterior mean.

Regularized Expected Improvement

We also evaluated a *regularized* form of EI by scaling the standard EI by a design-specific scaling factor, $\mathcal{F}(s_k;\mathcal{P})$. In our experiments, $\mathcal{F}$ refers to the log-odds score obtained by either an MRF or profile HMM, as described in Section 3.1. Our regularized EI is defined as:

$$\mathrm{EI}_{\mathcal{F}}(s_k;\mathcal{P}) = \mathrm{EI}(s_k;\mathcal{P})\mathcal{F}(s_k) \tag{2}$$

We will demonstrate in Section 4 that this small modification to the acquisition function results in a substantial shift in the designs discovered via ML-assisted DE. In particular, our method finds designs that are substantially more native-like, with only a small decrease in expected fitness.

3.3 Directed evolution with machine learning and *in silico* traditional approaches

Our experiments contrast the performance of "standard" ML-assisted DE (i.e., non-regularized) to the regularized version. We also compare the results to simulated forms of "traditional" DE (i.e., without ML), as was also done in [32]. Specifically, we simulated both the single mutation walk and recombination versions of DE (see Fig. 1). We note that the single mutation walk approach is deterministic, given the starting sequence. With the single step, we start each trial with a randomly chosen sequence from the GB1 variant library. At each of positions 39, 40, 41, and 54, we observe the experimentally determined fitness values for all possible single-residue mutations. Having observed these mutations, we then fix in place the single-residue mutant which has the highest fitness. With this residue fixed, we then repeat this procedure for the remaining unfixed residue positions. Continuing in this manner, the trial ends when all residues have been fixed. All observed fitness values within a trial thus represent a DE determined fitness function approximation.

For the recombination method, we mimic saturation mutatgenesis experiments by starting with n randomly chosen sequences from the GB1 variant library. From these, we identify the top three sequences that have highest fitness (as was done in [32]), and use these sequences to perform recombination. A recombinant library is simulated *in silico* by computing the Cartesian product $S_{39} \times S_{40} \times S_{41} \times S_{54}$, where the set S_m refers to the variant residues found at position m among the three highest fitness sequences in the initial random library. The resulting list of 4-tuples defines the recombinant library. Here, the DE fitness function approximation is given by observing fitness values for the n starting sequences as well as the recombined sequences.

4 Results

In this section, we report the results of five approaches to performing DE: (i) single mutation walk (see Fig. 1-top); (ii) recombination (see Fig. 1-bottom); (iii) Bayesian optimization using standard expected improvement (EI), denoted by "GP+EI"; (iv) Bayesian optimization using regularized EI with MRF-derived log-odds, denoted by "GP+EI+GREMLIN"; and (v) Bayesian optimization using regularized EI with HMM-derived log-odds, denoted by "GP+EI+HMM". Gaussian Process regression models are used for (iii)-(v). The standard form of expected improvement (Eq. 1) is used for (iii), and the regularized version of EI (Eq. 2) is used for (iv) and (v).

Each method was allowed to screen (i.e., obtain fitness values for) a total of 191 variants. This number was chosen to be similar to the number of sequences screened by the deterministic single mutant walk so that each method had similar experimental burden. Each model was initially trained on 20 randomly selected sequences. The small number of initial sequences simulates the scenario where the available fitness data is limited, prior to DE. The Bayesian optimization methods selected the top 19 sequences during each acquisition round. Each model is then updated with the experimentally measured fitness values for the chosen batch of 19 sequences, and this process is repeated for 9 batches (ie. 20 initial sequences plus 9 batches of 19 designs per batch, giving $20 + 19 \times 9 = 191$ variants selected). We refer to a complete set of variant selection batches as a *trial.* We performed 100 total trials with each selection strategy with different random initial starting sequences. 20% of the data were held out for testing purposes.

Table 1 Descriptive statistics for fitness, GREMLIN log odds score, and HMM log odds score. The final three columns show correlations between each respective score.

Metric	Mean	Median	Variance	(1)	(2)	(3)
(1) Fitness	0.08	0.003	0.16	1		
(2) GREMLIN	0.54	0	1.06	0.17	1	
(3) HMM	2.71	2.56	1.12	0.12	0.67	1

4.1 Data

Protein G is an antibody-binding protein expressed in *Streptococcus*. The B1 domain of protein G (GB1) interacts with the Fc domain of immunoglobulins. We performed our experiments on an existing dataset generated by Wu *et al.* [31], who performed saturation mutagenesis at four carefully chosen sites in GB1 in order to investigate the protein's evolutionary landscape. The four chosen residues (V39, D40, G41, and V54) are collectively present in 12 of the protein's top 20 pairwise epistatic interactions, meaning these sites are expected to contain evolutionarily favorable variants [20].

The fitness criterion for their study was binding affinity to IgG-Fc. Experimental measurements were obtained for 149,361 out of 160,000 (i.e. 20^4) possible variants at these four loci using mRNA display [23], followed by high-throughput Illumina sequencing. Briefly, this approach to measuring binding affinity works by first creating an input mRNA-protein fusion library from GB1 variants. This input library is then exposed to the GB1 binding target IgG-Fc. Any variant that binds to the target is subsequently sequenced for identification. By measuring the counts of each variant contained in the input library, c^{in}, and output "selected" library, c^{out}, the relative fitness w of the ith variant is calculated as follows:

$$w_i = \gamma \frac{c_i^{\text{in}}}{c_i^{\text{out}}} \tag{3}$$

Here, γ is a normalizing factor that ensures the wildtype sequence has fitness 1, and sequences with improved fitness are greater than 1. The range of fitness scores is from 0 to 8.76, with mean 0.08. Only 3,643 sequences in the dataset ($\approx 2.4\%$) have fitness greater than 1.

The MRF and HMM models described in Sec. 3 were used to compute the log-odds of each of the 149,361 variants in the GB1 library. The log-odds were scaled to match the range of the fitness scores $(0 - 8.76)$. As described in Sec. 3.2, the (scaled) log-odds were used as regularization terms. Table 1 displays the descriptive statistics of the fitness and the scaled log-odds values used in our experiment. While most fitness and GREMLIN scores lie close to 0, the HMM scores have mean and median values of 2.71 and 2.56, respectively. The difference between the GREMLIN and HMM log-odds is not unexpected, because the HMM makes strong assumptions about the conditional independencies between residues, and therefore does not penalize as many interaction pairs. We note that because GREMLIN and HMM scores describe statistics related to position specific and pairwise patterns, wildtype sequence {V39,D40,G41,V54} takes the maximum value under these metrics. Further, GREMLIN and HMM scores have weak positive Pearson's correlation with fitness (0.17 and 0.12, respectively). Conversely, GREMLIN and HMM have relatively high correlation (0.67) to each other.

4.2 Protein fold family-regularization biases variant selection

Traditionally, DE techniques aim to identify sequences that score highly in one property. In Figure 3 we demonstrate that ML-assisted DE outperform simulated traditional approaches (i.e., single-mutant walk and recombination). Over each cumulative fractions of trials, each

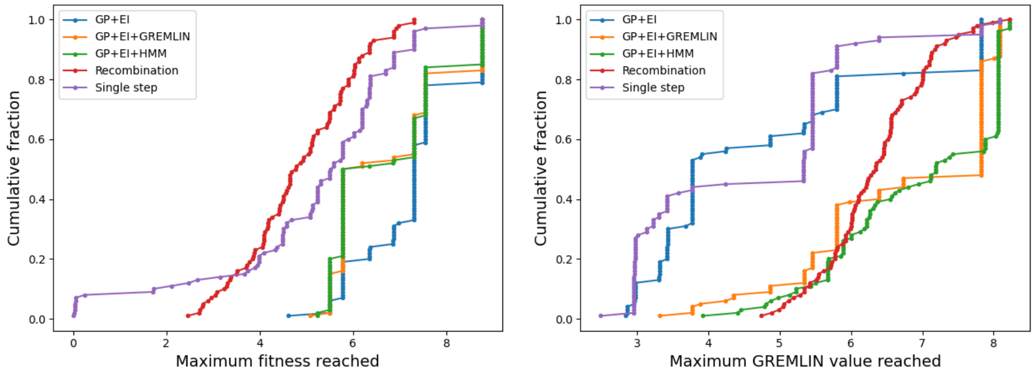


Figure 3 ML-assisted Directed Evolution techniques identify high fitness variants in fewer experiments. These curves show the fraction of trials (y-axis) that reach less than or equal to a specified fitness or log-odds value (x-axis). **(Left)** On average, GP+EI selected a variant with maximum fitness 7.28, followed by GP+EI+GREMLIN at 6.76, then GP+EI+HMM at 6.74. The simulated traditional single step and recombination methods identified variants with maximum fitness 5.13 and 4.86 on average, respectively. **(Right)** With respect to GREMLIN log-odds score, on average, the maximum variant scores identified by each model are 4.79 for GP+EI, 6.75 for GP+EI+GREMLIN, 6.95 for GP+EI+HMM, 6.34 for recombination, and 4.60 for single step.

of the three ML-assisted methods identify higher fitness variants than simulated traditional approaches. Of the two traditional approaches, the single mutant walk method fares better than recombination, though both consistently identify variants that have higher fitness than wildtype. On average, the single mutant walk procedure finds a variant with maximum fitness 5.13, whereas recombination yields a variant with maximum fitness 4.85.

GP+EI on average identifies variants with 7.28 fitness, followed by GP+EI+GREMLIN and GP+EI+HMM at 6.76 and 6.74, respectively. This suggests that protein fold family-regularization via GREMLIN or HMM induces a small reduction ($\approx 7\%$) in overall fitness. Since GP+EI+GREMLIN and GP+EI+HMM are regularized with the intent to select variants with higher GREMLIN or HMM scores, we expect that this fitness cost is actually a tradeoff with these metrics. Figure 4 supports this notion. In Figure 4 top left, GP+EI achieves highest per-batch average maximum fitness, and is able to do so in fewer batches. This is consistent with the result in Figure 3. However, GP+EI+GREMLIN and GP+EI+HMM are able to find variants of equal fitness levels to GP+EI if given more batches. We note that 20 initial sequences plus 9 batches of 19 corresponds to 191 mutagensis and screening experiments, a relatively modest burden.

The per-batch average GREMLIN and HMM scores of selected variants reveal a different pattern. In the top right and bottom of Figure 4, it is clear that variants selected by GP+EI+GREM and GP+EI+HMM have highest per-batch average maximum log-odds, and this holds true for every batch. Thus, the 7% decrease in binding affinity is more than compensated for by a 41 to 45% increase in the the log-odds of our designs under a generative model of the GB1 fold family.

That GP+EI+GREMLIN and GP+EI+HMM have high overlap is not unexpected, given the high correlation between these two metrics. GP+EI at no point selects variants with log-odds scores on par with GP+EI+GREMLIN and GP+EI+HMM. Since GP+EI gives no consideration to these scores when making variant selection decisions, it should not be expected that this performance difference will ameliorate itself given more batches. Thus, protein fold family-regularization can bias variant selections towards those with favorable fitness and fold family conservation statistics.

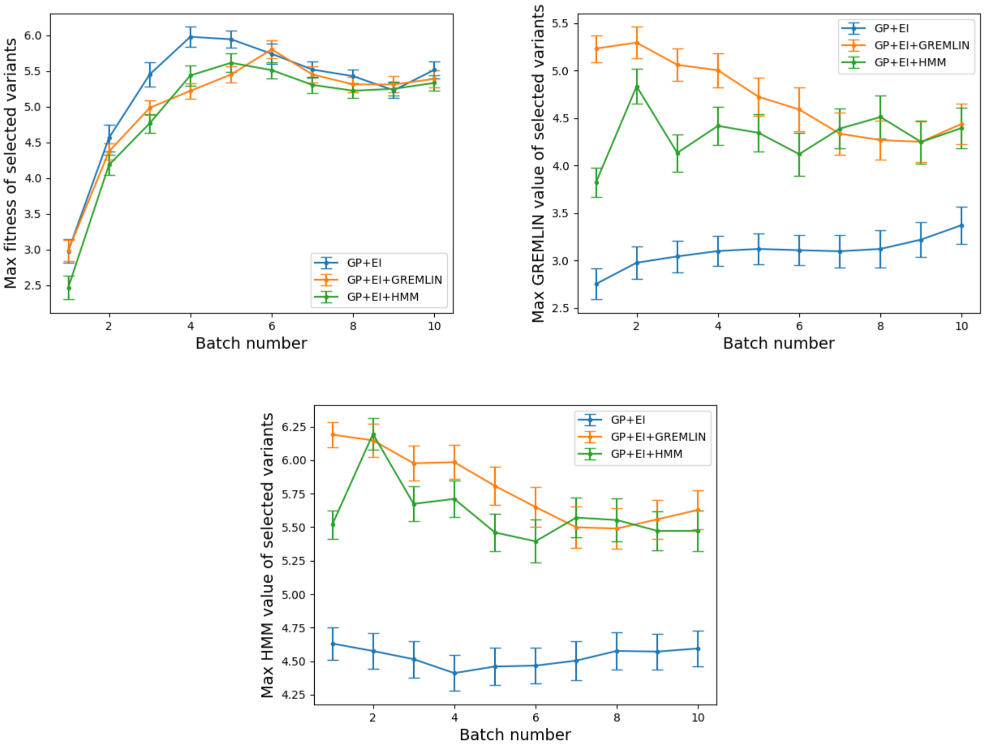


Figure 4 Protein fold family-regularization biases variant selections towards those with native-like position-specific and pairwise statistics with little cost to fitness. These plots show per batch averages over 100 trials. Models were initialized with 20 randomly chosen sequences, and each batch consists of 19 selected variants. **(Top Left)** GP+EI chooses variants with higher max fitness in fewer batches than other methods. GP+EI+GREMLIN and GP+EI+HMM catch up to GP+EI given enough batches. The curves are not monotonic because the values being plotted are fitness values for a particular batch, not the best fitness seen thus far. **(Top right)** GP+EI+GREMLIN and GP+EI+HMM are biased towards selecting variants with higher GREMLIN scores, and do so consistently better than GP+EI in each batch. **(Bottom)** The same pattern is observed for the HMM scores.

To further demonstrate the utility of protein fold family-regularization, Figure 5 compares the maximum fitness and maximum GREMLIN scores of variants selected in each of 100 trials. For brevity, we omit similar results with HMM scores from this discussion. To show how this process evolves as selections are made, the left figure shows selections from the first batches of each trial, and the right figure shows the final selections made in each trial. First batch selections use models that have only observed 20 random sequences, while the last batches have also observed all choices made in previous batches. There is clear separation between max GREMLIN scores of GP+EI and GP+EI+GREMLIN in the first batch. GP+EI+GREMLIN identifies variants with maximum GREMLIN score of at least 3 in most trials, whereas most GP+EI selected variants have GREMLIN scores of 4 or less. On the other hand, the max fitness of selected variants have very similar means (GP+EI = 2.98, GP+EI+GREMLIN =2.99) and variance (GP+EI = 2.57, GP+EI+GREMLIN = 2.21). Even as the models are able to select new variants and update themselves, we observe that this general trend persists (Figure 5, right hand side). Thus, selections made by GP+EI+GREMLIN are most strongly biased toward variants with high GREMLIN scores, while still being able to identify variants of fitness on par with GP+EI.

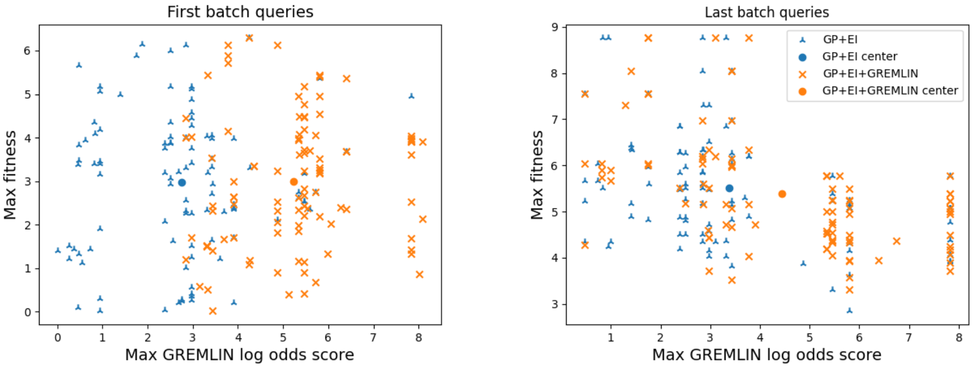


Figure 5 Biased variant selection persists over many batches when using protein fold family-regularization. Shown is the average maximum fitness and GREMLIN scores of selected variants with the first **(Left)** and last **(Right)** batch of variant selection using GP+EI and GP+EI+GREMLIN. The dots denoted "center" corresponds to the mean of its respective metric.

4.3 Sequence characteristics of selected variants

Thus far, we have characterized each ML-assisted DE technique in terms of the average maximum fitness and the log-odds of selected variants. We now consider the residue-specific behavior of choices made under each model. Figure 6 shows residue-specific entropy of variant selections. A high entropy grid (dark blue) indicates that the model selects many different residue types at that position within a given batch. Low entropy (light blue) indicates that the model selects only few residue types. The relative entropy of selections thus provides a sense of the confidence the model has that a particular variant residue is informative. Each model has relatively low entropy at residues 41 and 54, where GP+EI has lowest entropy at residue 54. The models attain low entropy statuses at these residues during the early batches, and maintain low entropy for the duration of the trial. Conversely, residues 39 and 40 have high entropy throughout, even increasing in later batches. This suggests that all the models attain certainty at residues 41 and 54, but are uncertain at residue positions 39 and 40.

In Figure 7, we show sequence logos obtained from each model after the final batch selections. Again, positions 41 and 54 have highest information content among all positions. Interestingly, all models select similar residue types at each position. In positions 40, 41, and 54, the top two selections are the same residues, just in different orders. Most notably, the top choice at each position yields a variant sequence that is most optimal in terms of the applied regularization (or lack thereof). GP+EI has consensus sequence, {W39,W40,C41,A54} which has a very high fitness (7.28), but comparably low GREMLIN (0.47) and HMM (1.71) scores. Meanwhile, GP+EI+GREMLIN with consensus sequence {I39,W40,G41,A54} (fitness=2.75,GREMLIN=3.77,HMM=4.34) and GP+EI+HMM with consensus sequence {I39,W40,G41,A54} (fitness=2.34,GREMLIN=3.42,HMM=4.67) strike a balance between finding variants with increased fitness yet high GREMLIN and HMM score.

4.4 Predictions on unseen data

In the previous sections, we characterized the batched variant sequence selections made by ML-assisted DE techniques with and without protein fold family-regularization. These selections allowed us to iteratively update models for variant fitness using sequences that each model expected to be informative. Since each model selected different sequences according

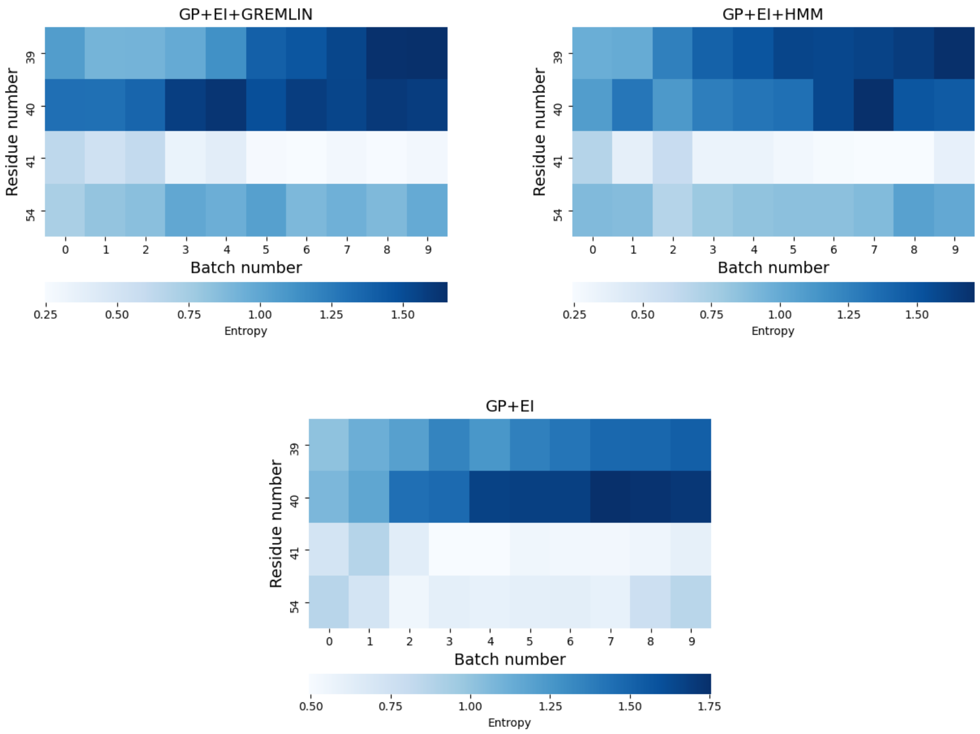


Figure 6 Bayesian selection techniques quickly identify informative sequence patterns. These plots show the per-batch average position-specific entropy of selections under each model. Lighter squares denote low entropy decisions, meaning the model selects among fewer residue types at that position in that batch. Clockwise from top left, models shown include GP+EI+GREMLIN, GP+EI+HMM, and GP+EI.

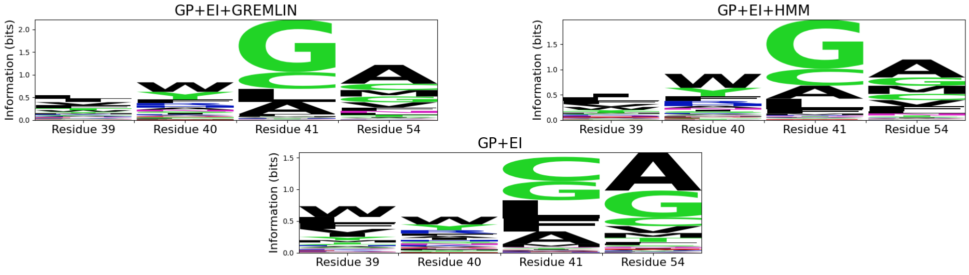


Figure 7 Protein fold family-regularization biases sequence logos towards sequences with desired properties. While each have similar logos, the GP+EI consensus sequence ({W39,W40,C41,A54}) has high fitness but low log-odds scores, whereas the consensus GP+EI+GREMLIN ({L39,W40,G41,A54}) and GP+EI+HMM ({I39,W40,G41,A54}) scores strike a balance between elevated fitness and high log-odds scores.

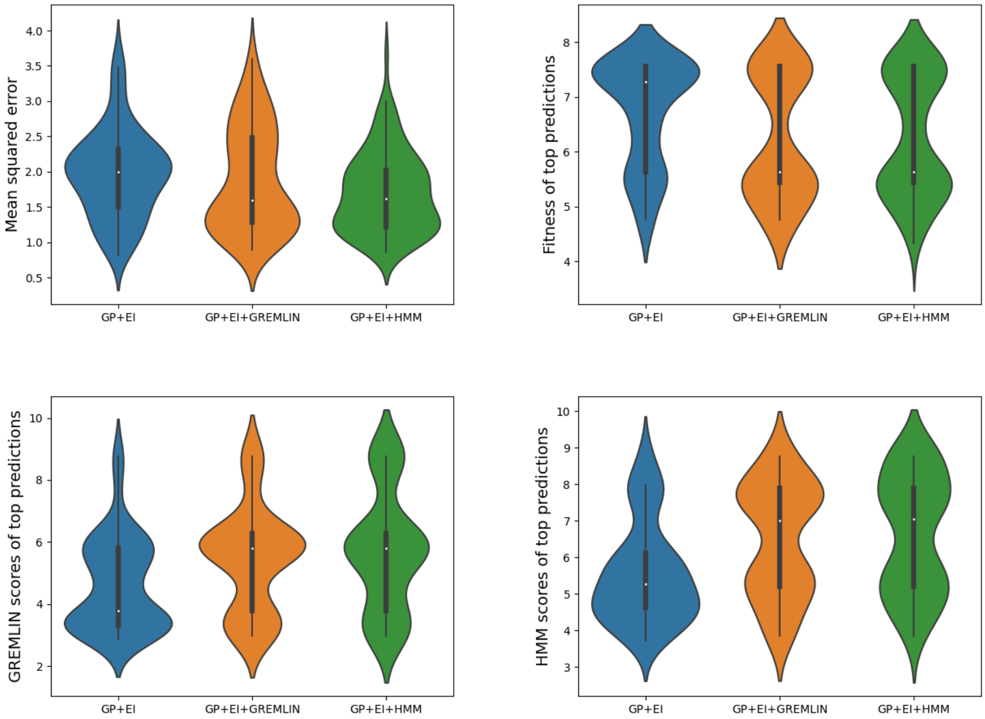


Figure 8 The sequential active learning strategies of ML-assisted DE approaches yields low-accuracy models, but are able to maintain any biases induced by protein fold family-regularization. Predictions made on 30,000 held out test sequences from the final models in each of 100 separate DE trials. Moving counterclockwise from the top left, included are mean squared error, true fitness values of the top 100 predictions from each model, true HMM scores of the top 100 predictions, and true GREMLIN scores of the top 100 predictions.

to their own unique selection objective, each model should also have unique predictive capabilities on unseen sequences. To demonstrate this, we used models obtained from each trial to predict the fitness of a held out test set of approximately 30,000 variants (Figure 8). We emphasize that these sequences were never seen by the models during the iterative variant selection stage of each trial, and that they constitute a random subsample of the GB1 data set. While the models generally yield low-accuracy in terms of predicting actual fitness values, we find that each are still able to discern between high fitness and low fitness variants, and that regularization introduced during selection and training influences model predictions.

Each plot in Figure 8 shows distributions over 100 trials of each model type. The thick black bar shows the interquartile range, and the white dot the median value. The width of each plot corresponds to the mass of the distribution at a given value. The top left figure shows the mean squared error of predictions. All models tend to have error greater than 1, meaning each model has relatively low accuracy in terms of predicting fitness of unseen sequences. The remaining plots show the true fitness, GREMLIN, and HMM scores of the top 100 sequences ranked by predicted fitness for each model. With respect to fitness (top right Figure 8), top predictions made by GP+EI have higher mass than those of GP+EI+GREMLIN or GP+EI+HMM. Conversely, with respect to GREMLIN and HMM scores (bottom Figure 8), the opposite trend exists, where the mass of the distributions are higher in GP+EI+GREMLIN and GP+EI+HMM than in GP+EI.

5 Discussion

Our work extends recent successes of ML-assisted DE. In particular, we show that GP regression techniques with and without protein fold family-regularization are able to identify high fitness variants with greater efficiency than traditional laboratory-based methods. These results echo those found by [32], though our ML-based strategies differ from theirs in key ways. First, they used an ensemble of different regressor types followed by extensive model selection, whereas we use a single Bayesian framework. Additionally, while they use a more traditional supervised learning setup, we adopt an active learning strategy where models iteratively select variants and are able to update themselves based on these selections. This more naturally mimics the workflow of real experimental labs. The batch sizes of an active learning setup can be easily tuned to match the throughput of a given lab, which is useful if incorporating ML-assisted DE into a laboratory-based experimental design. Finally, our models are able to identify high fitness variants while observing true fitness values of fewer sequences compared to the approach in [32]. Specifically, our sequential method only required 191 fitness value acquisitions (20 initial observations plus 9 batches of 19) to identify designs with high fitness values. In contrast, the experiment in [32], which used the same GB1 data, required 470 initial observations plus a single batch of 100 to find similarly fit designs. This is a 67% reduction in the number of sequences tested, which demonstrates the merits of active learning and Bayesian optimization, as the models are able to correct for mistakes made early in the learning process.

A common weakness with traditional DE approaches is that they only seek to optimize sequences with respect to a single property, such as fitness. However, there are any number of other characteristics that describe a protein, such as solubility, thermostability, or subcellular location, and in general, the relationships between these variables may be nonlinear. It is often favorable to identify sequences that score favorably according to multiple such metrics. In principle, one might consider using a set of regression models, each specialized to make predictions about a particular trait. Unfortunately, obtaining sufficient data to train such models is challenging, and we may not know all the relevant properties to consider. More importantly, such models would likely be trained on data from a range of fold families, and may therefore not include factors that are unique to a specific fold family. This motivated our use of fold family-regularization. Our results demonstrate that such regularization produces more native-like designs, with only a slight reduction in fitness.

We note that there are other ways that one might introduce fold family-regularization into this problem. A separate approach that we tried includes creating a new label for each sequence that is the product of fold family score $\mathcal{F}$ and variant fitness. Since the highest products should correspond to jointly high fitness and log odds scores, we expected that identifying variants that score highly according to this product should identify mutually favorable variants with respect to each individual score. However, we found that this approach does not obtain a desirable balanced tradeoff between fitness and log odds scores (data not shown, average maximum fitness of selected variants, GREMLIN×fitness = 4.62, HMM×fitness = 4.52, average maximum log odds score: GREMLIN×fitness = 7.37, HMM×fitness = 7.04). One possible explanation is that the generative fold family model is conditioned on the entire protein sequence, whereas our prediction task was constrained over only a few specific residues. Another is that the small number of training samples (between 20 and 190, in our experiments) may not be sufficient to learn such a complex function.

Finally, we noted a trend where selections by each model had low entropy at two residue positions. This coincided with each model being able to identify high fitness variants early within each active learning trial. This may suggest that the models quickly identified informative sequence patterns. When we looked at the types of residues selected by each model, we saw that each model had similar but different selection patterns. With respect to residue positions 41 and 54, the top two residues identified by each model were 41G vs 41C and 54A vs 54G. Interestingly, these selections correspond to some known biology. Olson *et al.* [20] used this same dataset to study epistasis in GB1. They observed that the mutation V54A is by itself neutral, but often beneficial (ie. increase fitness) in the presence of other mutants, and that this effect was most pronounced at residue 41. While 41G represents the wildtype sequence, each model seems to also favor the mutation G41C. Olson *et al.* find that, with a V54A background, the G41C mutation provides the highest fitness. Even at residue positions where the models had high entropy, it seems they identify informative sequence features. Each model consistently chooses variant D40W, and both in the background of mutant V54A and by itself as single mutant, this mutation increases fitness. Thus, our Bayesian ML-assisted DE methods are capable of identifying informative sequence patterns.

6 Conclusions and future work

We have introduced protein fold family-regularization for Bayesian ML-assisted directed evolution. Using log-odds scores from GREMLIN and Pfam HMM's, we calculate an adjusted expected improvement score that is biased towards selecting variants with native-like sequences. Using an active learning approach, we show that we can efficiently trade off small losses in variant fitness for larger gains in GREMLIN and HMM scores, while also outperforming traditional laboratory methods.

In our experiments, we investigated variants at four residue positions, totaling nearly 150,000 sequences. While we only needed small training sets to identify favorable variants, it could be the case that the amount of necessary training data increases as the number of residue positions under investigation increases. A downside with using GP as the underlying model is that they do not scale well with large data, as inference with a GP has complexity $\mathcal{O}(n^3)$. A natural extension to our procedure then is to incorporate recent advances in learning sparse GPs, using either pseudoinputs or variational techniques [29, 13]. We could also imagine improving our model by using kernels that are specifically designed with sequential data in mind [18, 19].

Finally, we have so far observed that using fold family-regularization induces a fitness tradeoff. However, our model does not attempt to harness this tradeoff in any way. We can imagine a utility where a user specifies a tradeoff parameter which dictates by how much the regularization should favor the generative model for a protein fold family compared to fitness. It could be informative to use such a utility to identify a Pareto optimal frontier over variant sequences (ex. [25]), where sequences on the boundary correspond to those that jointly optimize fitness and probability under a given fold family model.

References

1 Frances H. Arnold. Directed evolution: Bringing new chemistry to life. *Angewandte Chemie International Edition*, 57(16):4143–4148, 2018. doi:10.1002/anie.201708408.

2 S. Balakrishnan, H. Kamisetty, J.C. Carbonell, S.I. Lee, and Langmead C.J. Learning Generative Models for Protein Fold Families. *Proteins: Structure, Function, and Bioinformatics*, 79(6):1061–1078, 2011.

3 James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc., 2011.

4 F Edward Boas and Pehr B Harbury. Potential energy functions for protein design. *Current Opinion in Structural Biology*, 17(2):199–204, April 2007. doi:10.1016/j.sbi.2007.03.006.

5 Sara El-Gebali, Jaina Mistry, Alex Bateman, Sean R Eddy, Aurélien Luciani, Simon C Potter, Matloob Qureshi, Lorna J Richardson, Gustavo A Salazar, Alfredo Smart, Erik L L Sonnhammer, Layla Hirsh, Lisanna Paladin, Damiano Piovesan, Silvio C E Tosatto, and Robert D Finn. The Pfam protein families database in 2019. *Nucleic Acids Research*, 47(D1):D427–D432, October 2018. doi:10.1093/nar/gky995.

6 Pietro Gatti-Lafranconi, Antonino Natalello, Sascha Rehm, Silvia Maria Doglia, Jürgen Pleiss, and Marina Lotti. Evolution of Stability in a Cold-Active Enzyme Elicits Specificity Relaxation and Highlights Substrate-Related Effects on Temperature Adaptation. *Journal of Molecular Biology*, 395(1):155–166, January 2010. doi:10.1016/j.jmb.2009.10.026.

7 Lars Giger, Sami Caner, Richard Obexer, Peter Kast, David Baker, Nenad Ban, and Donald Hilvert. Evolution of a designed retro-aldolase leads to complete active site remodeling. *Nature Chemical Biology*, 9(8):494–498, August 2013. doi:10.1038/nchembio.1276.

8 Adi Goldenzweig and Sarel J. Fleishman. Principles of protein stability and their application in computational design. *Annual Review of Biochemistry*, 87(1):105–129, 2018. doi:10.1146/annurev-biochem-062917-012102.

9 Robert E. Hawkins, Stephen J. Russell, and Greg Winter. Selection of phage antibodies by binding affinity. *Journal of Molecular Biology*, 226(3):889–896, August 1992. doi:10.1016/0022-2836(92)90639-2.

10 P. Ilten, M. Williams, and Y. Yang. Event generator tuning using Bayesian optimization. *Journal of Instrumentation*, 12(04):P04028–P04028, April 2017. doi:10.1088/1748-0221/12/04/P04028.

11 Anders Krogh, Michael Brown, I.Saira Mian, Kimmen Sjölander, and David Haussler. Hidden Markov Models in Computational Biology. *Journal of Molecular Biology*, 235(5):1501–1531, February 1994. doi:10.1006/jmbi.1994.1104.

12 B. Kuhlman and D. Baker. Native protein sequences are close to optimal for their structures. *Proceedings of the National Academy of Sciences of the United States of America*, 97(19):10383–10388, September 2000. doi:10.1073/pnas.97.19.10383.

13 Haitao Liu, Yew-Soon Ong, Xiaobo Shen, and Jianfei Cai. When gaussian process meets big data: A review of scalable gps. *IEEE Transactions on Neural Networks and Learning Systems*, page 1–19, 2020.

14 Daniel J. Lizotte, Tao Wang, Michael H. Bowling, and Dale Schuurmans. Automatic gait optimization with gaussian process regression. In Manuela M. Veloso, editor, *IJCAI*, pages 944–949, 2007. URL: http://ijcai.org/Proceedings/07/Papers/152.pdf.

15 Stefan Lutz and Uwe Theo Bornscheuer. *Protein Engineering Handbook*. Wiley-VCH, Weinheim, 2012. OCLC: 890049290.

16 Jonas Mockus. *Bayesian Approach to Global Optimization: Theory and Applications*, volume 37 of *Mathematics and Its Applications*. Springer Netherlands, Dordrecht, 1989. doi:10.1007/978-94-009-0909-0.

17 Marziyeh Movahedi, Fatemeh Zare-Mirakabad, and Seyed Shahriar Arab. Evaluating the accuracy of protein design using native secondary sub-structures. *BMC Bioinformatics*, 17(1):353, September 2016. doi:10.1186/s12859-016-1199-y.

18 Saghi Nojoomi and Patrice Koehl. String kernels for protein sequence comparisons: improved fold recognition. *BMC Bioinformatics*, 18(1):137, February 2017. doi:10.1186/s12859-017-1560-9.

19 Saghi Nojoomi and Patrice Koehl. A weighted string kernel for protein fold recognition. *BMC Bioinformatics*, 18(1):378, August 2017. doi:10.1186/s12859-017-1795-5.

20 C. Anders Olson, Nicholas C. Wu, and Ren Sun. A comprehensive biophysical description of pairwise epistasis throughout an entire protein domain. *Current Biology*, 24(22):2643–2651, November 2014. doi:10.1016/j.cub.2014.09.072.

21 Carl Edward Rasmussen and Christopher K. I Williams. *Gaussian processes for machine learning*. MIT Press, Cambridge, Mass.; London, 2006. OCLC: 898708515.

22 Janes S. Richardson and David C. Richardson. The de novo design of protein structures. *Trends in Biochemical Sciences*, 14(7):304–309, July 1989. doi:10.1016/0968-0004(89)90070-4.

23 Richard W. Roberts and Jack W. Szostak. Rna-peptide fusions for the in vitro selection of peptides and proteins. *Proceedings of the National Academy of Sciences*, 94(23):12297–12302, 1997. doi:10.1073/pnas.94.23.12297.

24 Philip A. Romero and Frances H. Arnold. Exploring protein fitness landscapes by directed evolution. *Nature Reviews Molecular Cell Biology*, 10(12):866–876, December 2009. doi:10.1038/nrm2805.

25 Regina S. Salvat, Andrew S. Parker, Yoonjoo Choi, Chris Bailey-Kellogg, and Karl E. Griswold. Mapping the Pareto Optimal Design Space for a Functionally Deimmunized Biotherapeutic Candidate. *PLoS Computational Biology*, 11(1):e1003988, January 2015. doi:10.1371/journal.pcbi.1003988.

26 Fathima Aidha Shaikh and Stephen G. Withers. Teaching old enzymes new tricks: engineering and evolution of glycosidases and glycosyl transferases for improved glycoside synthesisThis paper is one of a selection of papers published in this Special Issue, entitled CSBMCB — Systems and Chemical Biology, and has undergone the Journal's usual peer review process. *Biochemistry and Cell Biology*, 86(2):169–177, April 2008. doi:10.1139/O07-149.

27 Tyler N. Starr and Joseph W. Thornton. Epistasis in protein evolution. *Protein science : a publication of the Protein Society*, 25(7):1204–1218, July 2016. doi:10.1002/pro.2897.

28 W. R Thompson. ON THE LIKELIHOOD THAT ONE UNKNOWN PROBABILITY EXCEEDS ANOTHER IN VIEW OF THE EVIDENCE OF TWO SAMPLES. *Biometrika*, 25(3-4):285–294, December 1933. doi:10.1093/biomet/25.3-4.285.

29 Michalis Titsias. Variational learning of inducing variables in sparse gaussian processes. In David van Dyk and Max Welling, editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 567–574, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16 2009.

30 James Wilson, Frank Hutter, and Marc Deisenroth. Maximizing acquisition functions for bayesian optimization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 9884–9895. Curran Associates, Inc., 2018.

31 Nicholas C Wu, Lei Dai, C Anders Olson, James O Lloyd-Smith, and Ren Sun. Adaptation in protein fitness landscapes is facilitated by indirect paths. *eLife*, 5:e16965, July 2016. doi:10.7554/eLife.16965.

32 Zachary Wu, S. B. Jennifer Kan, Russell D. Lewis, Bruce J. Wittmann, and Frances H. Arnold. Machine learning-assisted directed protein evolution with combinatorial libraries. *Proceedings of the National Academy of Sciences*, 116(18):8852–8858, 2019.

33 Kevin K. Yang, Zachary Wu, and Frances H. Arnold. Machine-learning-guided directed evolution for protein engineering. *Nature Methods*, 16(8):687–694, August 2019. doi:10.1038/s41592-019-0496-6.

Sequence Searching Allowing for Non-Overlapping Adjacent Unbalanced Translocations

Domenico Cantone
University of Catania, Italy
domenico.cantone@unict.it

Simone Faro
University of Catania, Italy
simone.faro@unict.it

Arianna Pavone
University of Messina, Italy
apavone@unime.it

Abstract

Unbalanced translocations are among the most frequent chromosomal alterations, accounted for 30% of all losses of heterozygosity, a major genetic event causing inactivation of tumor suppressor genes. Despite of their central role in genomic sequence analysis, little attention has been devoted to the problem of matching sequences allowing for this kind of chromosomal alteration.

In this paper we investigate the *approximate string matching* problem when the edit operations are non-overlapping unbalanced translocations of adjacent factors. In particular, we first present a $\mathcal{O}(nm^3)$-time and $\mathcal{O}(m^2)$-space algorithm based on the dynamic-programming approach. Then we improve our first result by designing a second solution which makes use of the Directed Acyclic Word Graph of the pattern. In particular, we show that under the assumptions of equiprobability and independence of characters, our algorithm has a $\mathcal{O}(n \log_\sigma^2 m)$ average time complexity, for an alphabet of size σ, still maintaining the $\mathcal{O}(nm^3)$-time and the $\mathcal{O}(m^2)$-space complexity in the worst case. To the best of our knowledge this is the first solution in literature for the approximate string matching problem allowing for unbalanced translocations of factors.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases Text processing, approximate matching, inversions, sequence matching

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.19

Funding This work was supported by the University of Catania “programma ricerca di ateneo UNICT 2020-22 linea 2”.

1 Introduction

Retrieving information and teasing out the meaning of biological sequences are central problems in modern biology. Generally, basic biological information is stored in strings of nucleic acids (DNA, RNA) or amino acids (proteins).

In recent years, much work has been devoted to the development of efficient methods for aligning strings and, despite sequence alignment seems to be a well-understood problem (especially in the edit-distance model), the same cannot be said for the approximate string matching problem on biological sequences.

String alignment and *approximate string matching* are two fundamental problems in text processing. Given two input sequences x, of length m, and y, of length n, the *string alignment* problem consists in finding a set of edit operations able to transform x in y, while the *approximate string matching* problem consists in finding all approximate matches of x in y. The closeness of a match is measured in terms of the sum of the costs of the elementary edit operations necessary to convert the string into an exact match.

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti; Article No. 19; pp. 19:1–19:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Most biological string matching methods are based on the *Levenshtein distance* [15], commonly referred to just as *edit distance*, or on the *Damerau distance* [11]. The edit operations in the case of the Levenshtein distance are *insertions*, *deletions*, and *substitutions* of characters, whereas, in the case of the Damerau distance, *swaps* of characters, i.e., transpositions of two adjacent characters, are also allowed (for an in-depth survey on approximate string matching, see [18]). Both distances assume that changes between strings occur locally, i.e., only a small portion of the string is involved in the mutation event. However, evidence shows that in some cases large scale changes are possible [8, 10, 21] and that such mutations are crucial in DNA since they often cause genetic diseases [16, 17]. For example, large pieces of DNA can be moved from one location to another (*translocations*) [8, 19, 22, 23], or replaced by their reversed complements (*inversions*) [3].

Translocations can be *balanced* (when equal length pieces are swapped) or *unbalanced* (when pieces with different lengths are moved). Interestingly, unbalanced translocations are a relatively common type of mutation and a major contributor to neurodevelopmental disorders [23]. In addition, cytogenetic studies have also indicated that unbalanced translocations can be found in human genome with a de novo frequency of 1 in 2000 [22] and that it is a frequent chromosome alteration in a variety of human cancers [19]. Hence the need for practical and efficient methods for detecting and locating such kind of large scale mutations in biological sequences arises.

1.1 Related Results

In the last three decades much work has been made for the alignment and matching problem allowing for chromosomal alteration, especially for non-overlapping inversions. Table 1 shows the list of all solutions proposed over the years, together with their worst-case, average-case and space complexities.

Concerning the alignment problem with inversions, a first solution based on dynamic programming, was proposed by Schöniger and Waterman [20], which runs in $\mathcal{O}(n^2m^2)$-time and $\mathcal{O}(n^2m^2)$-space on input sequences of length n and m. Several other papers have been devoted to the alignment problem with inversions. The best solution is due to Vellozo *et al.* [21], who proposed a $\mathcal{O}(nm^2)$-time and $\mathcal{O}(nm)$-space algorithm, within the more general framework of an edit graph.

Regarding the alignment problem with translocations, Cho *et al.* [8] presented a first solution for the case of inversions and translocations of equal length factors (i.e., balanced translocations), working in $\mathcal{O}(n^3)$-time and $\mathcal{O}(m^2)$-space. However their solution generalizes the problem to the case where edit operations can occur on both strings and assume that the input sequences have the same length, namely $|x| = |y| = n$.

Regarding the approximate string matching problem, a first solution was presented by Cantone *et al.* [2], where the authors presented an algorithm running in $\mathcal{O}(nm)$ worst-case time and $\mathcal{O}(m^2)$-space for the approximate string matching problem allowing for non-overlapping inversions. Additionally, they also provided a variant [3] of the algorithm which has the same complexity in the worst case, but achieves $\mathcal{O}(n)$-time complexity on average. Cantone *et al.* also proposed in [4] an efficient solution running in $\mathcal{O}(nm^2)$-time and $\mathcal{O}(m^2)$-space for a slightly more general problem, allowing for balanced translocations of adjacent factors besides non-overlapping inversions. The authors improved their previous result in [5] obtaining an algorithm having $\mathcal{O}(n)$-time complexity on average. We mention also the result by Grabowski *et al.* [14], which solves the same string matching problem in $\mathcal{O}(nm^2)$-time and $\mathcal{O}(m)$-space, reaching in practical cases $\mathcal{O}(n)$-time complexity.

Table 1 Results related to alignment and matching of strings allowing for inversions and translocations of factors. All the edit operations allowed by all the listed solutions are intended to involve only non-overlapping factors of the pattern.

Authors	Year	W.C. Time	AVG Time	Space
Alignment with inversions				
Schoniger and Waterman [20]	(1992)	$\mathcal{O}(n^2m^2)$	-	$\mathcal{O}(m^2)$
Gao *et al.* [13]	(2003)	$\mathcal{O}(n^2m^2)$	-	$\mathcal{O}(nm)$
Chen *et al.* [7]	(2004)	$\mathcal{O}(n^2m^2)$	-	$\mathcal{O}(nm)$
Alves *et al.* [1]	(2005)	$\mathcal{O}(n^3 \log n)$	-	$\mathcal{O}(n^2)$
Vellozo *et al.* [21]	(2006)	$\mathcal{O}(nm^2)$	-	$\mathcal{O}(nm)$
Alignment with inversions and balanced translocations on both strings				
Cho *et al.* [8]	(2015)	$\mathcal{O}(m^3)$	-	$\mathcal{O}(m^2)$
Pattern matching with inversions				
Cantone *et al.* [2]	(2011)	$\mathcal{O}(nm)$	-	$\mathcal{O}(m^2)$
Cantone *et al.* [3]	(2013)	$\mathcal{O}(nm)$	$\mathcal{O}(n)$	$\mathcal{O}(m^2)$
Pattern matching with inversions and balanced translocations				
Cantone *et al.* [4]	(2010)	$\mathcal{O}(nm^2)$	$\mathcal{O}(n \log m)$	$\mathcal{O}(m^2)$
Grabowski *et al.* [14]	(2011)	$\mathcal{O}(nm^2)$	$\mathcal{O}(n)$	$\mathcal{O}(m)$
Cantone *et al.* [5]	(2014)	$\mathcal{O}(nm^2)$	$\mathcal{O}(n)$	$\mathcal{O}(m)$
Pattern matching with unbalanced translocations				
This paper	(2020)	$\mathcal{O}(nm^3)$	$\mathcal{O}(n \log^2 m)$	$\mathcal{O}(m^2)$

1.2 Our Results

While in the previous results mentioned above it is intended that a translocation may take place only between balanced factors of the pattern, in this paper we investigate the approximate string matching problem under a string distance whose edit operations are non-overlapping unbalanced translocations of adjacent factors. To the best of our knowledge, this slightly more general problem has never been addressed in the context of approximate pattern matching on biological sequences. First, we propose a solution to the problem, based on the general dynamic programming approach, which needs $\mathcal{O}(nm^3)$-time and $\mathcal{O}(m^2)$-space. Subsequently, we propose a second solution to the problem that makes use of the Directed Acyclic Word Graph of the pattern and achieves a $\mathcal{O}(n \log_\sigma^2 m)$-time complexity on average, for an alphabet of size $\sigma \geq 4$, still maintaining the same complexity, in the worst case, as for in the first solution.

The rest of the paper is organized as follows. In Section 2 we introduce some preliminary notions and definitions. Subsequently, in Section 3 we present our first solution running in $\mathcal{O}(nm^3)$-time based on the dynamic programming approach. In Section 4 we present our second algorithm and analyze it both in the worst and in the average case. Finally draw our conclusions in Section 5.

2 Basic notions and definitions

A string x of length $m \geq 0$, over an alphabet Σ, is represented as a finite array $x[1\,..\,m]$ of elements of Σ. We write $|x| = m$ to indicate its length. In particular, when $m = 0$ we have the empty string ε. We denote by $x[i]$ the i-th character of x, for $1 \leq i \leq m$. Likewise, the factor of x, contained between the i-th and the j-th characters of x is indicated with $x[i\,..\,j]$, for $1 \leq i \leq j \leq m$. The set of factors of x is denoted by $\mathit{Fact}(x)$ and its size is $\mathcal{O}(m^2)$.

A string $w \in \Sigma^*$ is a suffix of x (in symbols, $w \sqsupseteq x$) if $w = x[i\,..\,m]$, for some $1 \leq i \leq m$. We denote by $\mathit{Suff}(x)$ the set of the suffixes of x. Similarly, we say that w is a prefix of x if $w = x[1\,..\,i]$, for some $1 \leq i \leq m$. Additionally, we use the symbol x_i to denote the prefix of x of length i (i.e.,$x_i = x[1\,..\,i]$), for $1 \leq i \leq m$, and make the convention that x_0 denotes the empty string ε. In addition, we write xw for the concatenation of the strings x and w.

For $w \in \mathit{Fact}(p)$, we denote with $\mathit{end\text{-}pos}(w)$ the set of all positions in x at which an occurrence of w ends; formally, we put $\mathit{end\text{-}pos}(w) := \{i \mid w \sqsupseteq x_i\}$. For any given pattern x, we define an equivalence relation $\mathcal{R}_x$ by putting, for all $w, z \in \Sigma^*$,

$$w \; \mathcal{R}_x \; z \iff \mathit{end\text{-}pos}(w) = \mathit{end\text{-}pos}(z).$$

We also denote with $\mathcal{R}_x(w)$ the equivalence class of the string w. For each equivalence class q of $\mathcal{R}_x$, we put $\mathit{len}(q) = |\mathit{val}(q)|$, where $\mathit{val}(q)$ is the longest string w in the equivalence class q.

► **Example 1.** Let $x = agcagccag$ be a string over $\Sigma = \{a, g, c, t\}$ of length $m = 9$. Then we have $\mathit{end\text{-}pos}(ag) = \{2, 5, 9\}$, since the substring ag occurs three times in x, ending at positions 2, 5 and 9, respectively, in that order. Similarly we have $\mathit{end\text{-}pos}(gcc) = \{7\}$. Observe that $\mathcal{R}_x(ag) = \{ag, g\}$. Similarly we have $\mathcal{R}_x(gc) = \{agc, gc\}$. Thus, we have $\mathit{val}(\mathcal{R}_x(ag)) = ag$, $\mathit{len}(\mathcal{R}_x(ag)) = 2$, $\mathit{val}(\mathcal{R}_x(gc)) = agc$ and $\mathit{len}(\mathcal{R}_x(gc)) = 3$.

The Directed Acyclic Word Graph [9] of a pattern x (Dawg, for short) is the deterministic automaton $\mathcal{A}(x) = (Q, \Sigma, \delta, \mathit{root}, F)$ whose language is $\mathit{Fact}(x)$, where $Q = \{\mathcal{R}_x(w) : w \in \mathit{Fact}(x)\}$ is the set of states, Σ is the alphabet of the characters in x, $\mathit{root} = \mathcal{R}_x(\varepsilon)$ is the initial state, $F = Q$ is the set of final states, and $\delta : Q \times \Sigma \to Q$ is the transition function defined by $\delta(\mathcal{R}_x(y), c) = \mathcal{R}_x(yc)$, for all $c \in \Sigma$ and $yc \in \mathit{Fact}(x)$.

We define a failure function, $s\ell : \mathit{Fact}(x) \setminus \{\varepsilon\} \to \mathit{Fact}(x)$, called *suffix link*, by putting, for any $w \in \mathit{Fact}(x) \setminus \{\varepsilon\}$,

$$s\ell(w) = \text{“ longest } y \in \mathit{Suff}(w) \text{ such that } y \; \mathcal{R}_x \, w\text{"}.$$

The function $s\ell$ enjoys the following property

$$w \; \mathcal{R}_x \; y \Longrightarrow s\ell(w) = s\ell(y).$$

We extend the functions $s\ell$ and $\mathit{end\text{-}pos}$ to Q by putting $s\ell(q) := \mathcal{R}_x(s\ell(\mathit{val}(q)))$ and $\mathit{end\text{-}pos}(q) = \mathit{end\text{-}pos}(\mathit{val}(q))$, for each $q \in Q$. Figure 2 shows the Dawg of the pattern $x = aggga$, where the edges of the automaton are depicted in black while the suffix links are depicted in red.

A *distance* $d : \Sigma^* \times \Sigma^* \to \mathbb{R}$ is a function which associates to any pair of strings x and y the minimal cost of any finite sequence of edit operations which transforms x into y, if such a sequence exists, ∞ otherwise.

In this paper we consider the *unbalanced translocation distance*, $\mathit{utd}(x, y)$, whose unique edit operation is the *translocation* of two adjacent factors of the string, with possibly different lengths. Specifically, in an *unbalanced translocation* a factor of the form zw is transformed into wz, provided that both $|z|, |w| > 0$ (it is not necessary that $|z| = |w|$). We assign a unit cost to each translocation.

► **Example 2.** Let $x = g\bar{t}\underline{ga}c\overline{cgt}\underline{ccag}$ and $y = g\underline{ga}\bar{t}c\underline{ccag}\overline{cgt}$ be given two strings of length 12. Then $\mathit{utd}(x, y) = 2$ since x can be transformed into y by translocating the substrings $x[3..4] = ga$ and $x[2..2] = t$, and translocating the substrings $x[6..8] = cgt$ and $x[9..12] = ccag$.

When $\mathit{utd}(x, y) < \infty$, we say that x and y have utd-match. If x has utd-match with a suffix of y, we write $x \overset{\mathit{trd}}{\sqsupseteq} y$.

3 A Dynamic Programming Solution

In this section we present a general dynamic programming algorithm for the pattern matching problem with adjacent unbalanced translocations.

Let x be a pattern of length m and y a text of length n, with $n \geq m$, both over the same alphabet Σ of size σ. Our algorithm is designed to iteratively compute, for $j = m, m{+}1, \ldots, n$, all the prefixes of x which have a *utd*-match with the suffix of y_j, by exploiting information gathered at previous iterations. For this purpose, it is convenient to introduce the set $\mathtt{P}_j$, for $1 \leq j \leq n$, defined by

$$\mathtt{P}_j := \{1 \leq i \leq m \mid x_i \overset{trd}{\sqsupseteq} y_j\}.$$

Thus, the pattern x has an *utd*-match ending at position j of the text y if and only if $m \in \mathtt{P}_j$.

Since the allowed edit operations involve substrings of the pattern x, it is useful to consider also the set $\mathtt{F}_j^k$ of all the positions in x at which an occurrence of the suffix of y_j of length k ends. More precisely, for $1 \leq k \leq m$ and $k-1 \leq j \leq n$, we put

$$\mathtt{F}_j^k := \{k-1 \leq i \leq m \mid y[j-k+1 \mathinner{..} j] \sqsupseteq x_i\}\,.$$

Observe that

$$\mathtt{F}_j^k \subseteq \mathtt{F}_j^h, \text{ for } 1 \leq h \leq k \leq m. \tag{1}$$

▶ **Example 3.** Let $x = cattcatgatcat$ $y = atcatgacttactgactta$ be a pattern and respectively a text. Then $\mathtt{F}_5^3$ is the set of all positions in x at which an occurrence of the suffix of y_5 of length 3 ends, namely, *cat*. Thus $\mathtt{F}_5^3 = \{3, 7, 13\}$. Similarly, we have that $\mathtt{F}_5^2 = \{3, 7, 10, 13\}$. Observe that $\mathtt{F}_5^3 \subseteq \mathtt{F}_5^2$.

The sets $\mathtt{P}_j$ and $\mathtt{F}_j^k$ can then be computed by way of the recursive relations contained in the following elementary lemmas.

▶ **Lemma 4.** *Let y and x be a text of length n and a pattern of length m, respectively. Then $i \in P_j$, for $1 \leq i \leq m$ and $i \leq j \leq n$, if and only if one of the following two facts holds:*

(a) *$x[i] = y[j]$ and $(i-1) \in P_{j-1} \cup \{0\}$;*

(b) *$(i-k) \in F_j^h$, $i \in F_{j-h}^k$, and $(i-k-h) \in P_{j-k-h} \cup \{0\}$, for some $1 \leq k, h < i$ such that $h + k \leq i$;* ◀

Notice that condition (b) in Lemma 4 refers to a translocation of adjacent factors of length k and h, respectively.

Likewise, the sets $\mathtt{F}_j^k$ can be computed according to the following lemma.

▶ **Lemma 5.** *Let y and x be a text of length n and a pattern of length m, respectively. Then $i \in F_j^k$ if and only if one of the following condition holds:*

(a) *$x[i] = y[j]$ and $k = 1$;*

(b) *$x[i] = y[j]$, $1 < k < i$ and $(i-1) \in F_{j-1}^{k-1}$,*

for $1 \leq k < i < m$ and $k-1 \leq j \leq n$. ◀

Based on the recurrence relations in Lemmas 4 and 5, a general dynamic programming algorithm can be readily constructed, characterized by an overall $\mathcal{O}(nm^3)$-time and $\mathcal{O}(m^3)$-space complexity. However, the overhead due to the computation and the maintenance of the sets $\mathtt{F}_j^k$ turns out to be quite relevant.

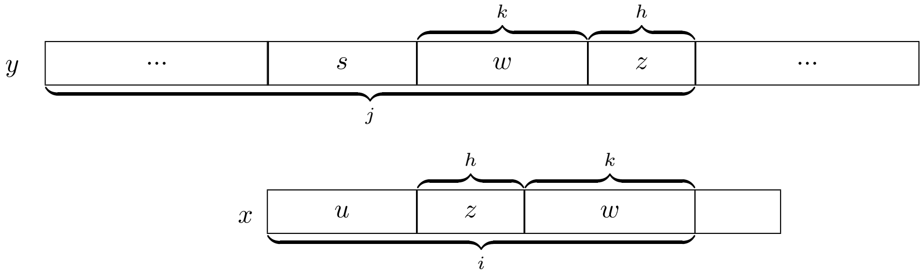


Figure 1 Case (b) of Lemma 4. The prefix u of the pattern, of length $i-h-k$, has a *utd*-match ending at position $j-h-k$ of the text, i.e. $(i-h-k) \in \mathtt{P}_{j-k-h} \cup \{0\}$. In addition the substring of the pattern $z = x[i-h-k+1..i-k]$, of length h has an exact match with the substring of the text $y[j-h+1..j]$, i.e. $(i-k) \in \mathtt{F}^h_j$. Finally the substring of the pattern $w = x[i-k+1..i]$, of length k has an exact match with the substring of the text $y[j-h-k+1..j-h]$, i.e. $i \in \mathtt{F}^k_{j-h}$.

Algorithm 1 A dynamic programming algorithm for the approximate string matching problem allowing for unbalanced translocations of adjacent factors. The algorithm is characterised by a $\mathcal{O}(nm^3)$-time and a $\mathcal{O}(m^2)$-space complexity.

```
ALGORITHM1
 1.  $P[0,0] \leftarrow$ true
 2.  for $j \leftarrow 1$ to $n$ do
 3.    $P[0,j] \leftarrow$ true
 4.    for $i \leftarrow 1$ to $m$ do
 5.      if $(x[i] = y[j])$ then
 6.        if $(P[i-1,j-1])$ then
 7.          $P[i,j] \leftarrow$ true
 8.        $F[i,j] \leftarrow F[i-1,j-1]+1$
 9.      for $k \leftarrow 1$ to $i-1$ do
10.        for $h \leftarrow 1$ to $i-k$ do
11.          if $(F[i-h,j] \geq k$ and $F[i,j-k] \geq h)$ then
12.            if $(P[i-h-k,j-h-k])$ then
13.              $P[i,j] \leftarrow$ true
```

Based on equation (1), we can represent the sets $\mathtt{F}^k_j$ by means of a single matrix $\mathtt{F}$ of size $m \times n$. Specifically, for $1 \leq i \leq m$ and $1 \leq j \leq n$, we set $\mathtt{F}[i,j]$ as the length of the longest suffix of x_i which is also a suffix of y_j. More formally we have:

$$i \in \mathtt{F}^k_j \iff \mathtt{F}[i,j] \geq k.$$

However, since we need only the last m columns of the matrix $\mathtt{F}$ for computing the next column, we can maintain it by means of a matrix $\mathtt{F}$ of size m^2. The pseudocode of the resulting dynamic programming algorithm, called ALGORITHM1, is shown in Figure 1. Due to the four for-loops at lines 2, 4, 9, and 10, respectively, it can straightforwardly be proved that ALGORITHM1 has a $\mathcal{O}(nm^3)$-time and $\mathcal{O}(m^2)$-space complexity. Indeed, the matrices $\mathtt{F}$ and $\mathtt{P}$ are filled by columns and therefore we need to store only the last m columns at each iteration of the for-loop at line 2.

4 An Automaton-Based Algorithm

In this section we improve the algorithm described in Section 3 by means of an efficient method for computing the sets $\mathtt{F}_j^k$, for $1 \leq j \leq n$ and $1 \leq k < j$. Such method makes use of the DAWG of the pattern x and the function *end-pos*.

Let $\mathcal{A}(x) = (Q, \Sigma, \delta, \mathit{root}, F)$ be the DAWG of x. For each position j in y, let w be the longest factor of x which is a suffix of y_j too; also, let q_j be the state of $\mathcal{A}(x)$ such that $\mathcal{R}_x(w) = q_j$, and let l_j be the length of w. We call the pair (q_j, l_j) a *y-configuration* of $\mathcal{A}(x)$.

The idea is to compute the y-configuration (q_j, l_j) of $\mathcal{A}(x)$, for each position j of the text, while scanning the text y. The set $\mathtt{F}_j^k$ computed at previous iterations do not need to be maintained explicitly; rather, it is enough to maintain only y-configurations. These are then used to compute efficiently the set $\mathtt{F}_j^k$ only when needed.

▶ **Example 6.** Let $x = \mathit{aggga}$ be a pattern and $y = \mathit{aggagcatgggactaga}$ a text respectively. Let $\mathcal{A}(x) = (Q, \Sigma, \delta, \mathit{root}, F)$ be the DAWG of x as depicted in Figure 2, where $\mathit{root} = q_0$ is the initial state and $F = \{q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$ is the set of final states. Edges of the DAWG are depicted in black while suffix links are depicted in red. Observe that, after scanning the suffix y_5 starting from state q_0 of $\mathcal{A}(x)$, we reach state q_2 of the automaton. Thus, the corresponding y-configuration is $(q_2, 2)$. Similarly, after scanning the suffix y_{11}, we get the y-configuration $(q_4, 3)$.

The longest factor of x ending at position j of y is computed in the same way as in the Forward-Dawg-Matching algorithm for the exact pattern matching problem (the interested readers are referred to [9] for further details).

Specifically, let (q_{j-1}, l_{j-1}) be the y-configuration of $\mathcal{A}(x)$ at step $(j-1)$. The new y-configuration (q_j, l_j) is set to $(\delta(q, y[j]), \mathit{length}(q) + 1)$, where q is the first node in the suffix path $\langle q_{j-1}, s\ell(q_{j-1}), s\ell^{(2)}(q_{j-1}), \ldots\rangle$ of q_{j-1}, including q_{j-1}, having a transition on $y[j]$, if such a node exists; otherwise (q_j, l_j) is set to $(\mathit{root}, 0)$.[1]

Before explaining how to compute the sets $\mathtt{F}_j^k$, it is convenient to introduce the partial function $\phi : Q \times \mathbb{N} \rightarrow Q$, which, given a node $q \in Q$ and a length $k \leq \mathit{length}(q)$, computes the state $\phi(q, k)$ whose corresponding set of factors contains the suffix of $\mathit{val}(q)$ of length k. Roughly speaking, $\phi(q, k)$ is the first node p in the suffix path of q such that $\mathit{length}(s\ell(p)) < k$.

In the preprocessing phase, the DAWG $\mathcal{A}(x) = (Q, \Sigma, \delta, \mathit{root}, F)$, together with the associated *end-pos* function, is computed. Since for a pattern x of length m we have $|Q| \leq 2m + 1$ and $|\mathit{end\text{-}pos}(q)| \leq m$, for each $q \in Q$, we need only $\mathcal{O}(m^2)$ extra space (see [9]).

To compute the set $\mathtt{F}_j^k$, for $1 \leq k \leq l_j$, we take advantage of the relation

$$\mathtt{F}_j^k = \mathit{end\text{-}pos}(\phi(q_j, k)). \tag{2}$$

Notice that, in particular, we have $\mathtt{F}_j^{l_j} = \mathit{end\text{-}pos}(q_j)$.

The time complexity of the computation of $\phi(q, k)$ can be bounded by the length of the suffix path of node q. Specifically, since the sequence

$$\langle \mathit{len}(s\ell^{(0)}(q)), \mathit{len}(s\ell^{(1)}(q)), \ldots, 0\rangle$$

of the lengths of the nodes in the suffix path from q is strictly decreasing, we can do at most $\mathit{len}(q)$ iterations over the suffix link, obtaining a $\mathcal{O}(m)$-time complexity.

[1] We recall that $s\ell^{(0)}(q) = q$ and, recursively, $s\ell^{(h+1)}(q) = s\ell(s\ell^{(h)}(q))$, for $h \geq 0$.

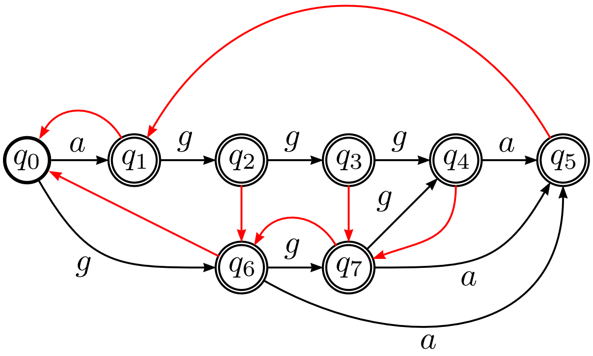


Figure 2 The Directed Acyclic Word Graph (DAWG) of the pattern $x = aggga$, where the edges of the automaton are depicted in black while the suffix links are depicted in red.

According to Lemma 4, a translocation of two adjacent factors of length k and h, respectively, at position j of the text y is possible only if two factors of x of lengths at least k and h, respectively, have been recognized at both positions j and $j-h$, namely if $l_j \geq h$ and $l_{j-h} \geq k$ hold (see Figure 1).

Let $\langle h_1, h_2, \ldots, h_r \rangle$ be the increasing sequence of all the values h such that $1 \leq h \leq \min(l_j, l_{j-h})$. For each $1 \leq i \leq r$, condition (b) of Lemma 4 requires member queries on the sets $\mathtt{F}_j^{h_i}$.

Observe that if we proceed for decreasing values of h, the sets $\mathtt{F}_j^h$, for $1 \leq h \leq l_j$, can be computed in constant time. Specifically, for $h = 1, \ldots, l_j - 1$ $\mathtt{F}_j^h$ can be computed in constant time from $\mathtt{F}_j^{h+1}$ with at most one iteration over the suffix link of the state $\phi(q_j, h+1)$.

Subsequently, for each member query on the set $\mathtt{F}_j^{h_i}$, condition (b) of Lemma 4 requires also member queries on the sets $\mathtt{F}_{j-h_i}^k$, for $1 \leq k < h_i$. Let $\langle k_1, k_2, \ldots, k_s \rangle$ be the increasing sequence of all the values k such that $1 \leq k \leq \min(l_{j-h_i}, l_{j-h_i-k})$. Also in this case we can proceed for decreasing values of k, in order to compute the sets $\mathtt{F}_{j-h_i}^k$ in constant time, for $1 \leq k \leq l_{j-h_i}$.

The resulting algorithm for the approximate string matching problem allowing for unbalanced translocations of adjacent factors is shown in Figure 2 and is named ALGORITHM2. In the next sections, we analyze its worst-case and average-case complexity.

4.1 Worst-case time and space analysis

In this section we determine the worst-case time and space analysis of ALGORITHM2 presented in the previous section. In particular, we will refer to the implementation of the ALGORITHM2 reported in Figure 2.

First of all, observe that the main for-loop at line 4 is always executed n times. For each of its iterations, the cost of the execution DAWG-DELTA (line 5) for computing the new y-configuration requires at most $\mathcal{O}(m)$-time. Since we have $|\mathtt{P}_j| \leq m+1$, for all $1 \leq j \leq n$, the for-loop at line 7 is also executed $\mathcal{O}(m)$-times. In addition, since we have $l_j \leq m$, for all $1 \leq j \leq n$, the two nested for-loops at lines 11 and 14 are executed m times. Observe also that the transitions of suffix links performed at lines 12 and 15 need only constant time. Thus, at each iteration of the main for-loop, the internal for-loop at line 14 takes at most $\mathcal{O}(m)$-time, while the for-loop at line 11 takes at most $\mathcal{O}(m^2)$-time. In addition the for-loop at line 16 takes at most $\mathcal{O}(m)$-time, since $|\mathtt{P}_{j-h-k}| \leq m+1$ and the tests at line 17 can be performed in constant time. Summing up, ALGORITHM2 has a $\mathcal{O}(nm^3)$ worst-case time complexity.

Algorithm 2 The code of ALGORITHM2 and the DAWG state update algorithm DAWG-DELTA. Notice that the function $s\ell^*$ in procedure DAWG-DELTA denotes the improved suffix link [9].

```
ALGORITHM2
    $m \leftarrow |x|$,    $n \leftarrow |y|$
    $(q_0, l_0) \leftarrow$ DAWG-DELTA$(root_{\mathcal{A}}, 0, y[0], \mathcal{A})$
    $\texttt{P}_0 \leftarrow \{0\}$
    for $j \leftarrow 1$ to $n$ do
        $(q_j, l_j) \leftarrow$ DAWG-DELTA$(q_{j-1}, l_{j-1}, y[j], \mathcal{A})$
        $\texttt{P}_j \leftarrow \{0\}$
        for $i \in \texttt{P}_{j-1}$ do
            if $i < m$ and $x[i+1] = y[j]$ then
                $\texttt{P}_j \leftarrow \texttt{P}_j \cup \{i+1\}$
        $u \leftarrow q_j$
        for $h \leftarrow l_j$ downto $1$ do
            if $h = len(s\ell_{\mathcal{A}}(u))$ then $u \leftarrow s\ell_{\mathcal{A}}(u)$
            $p \leftarrow q_{j-h}$
            for $k \leftarrow l_{j-h}$ downto $1$ do
                if $k = len(s\ell_{\mathcal{A}}(p))$ then $p \leftarrow s\ell_{\mathcal{A}}(p)$
                for $i \in \texttt{P}_{j-h-k}$ do
                    if $((i+h) \in end\text{-}pos(u)$ and $(i+h+k) \in end\text{-}pos(p))$ then
                        $\texttt{P}_j \leftarrow \texttt{P}_j \cup \{i+h+k\}$
        if $(m \in \texttt{P}_j)$ then Output$(j)$

DAWG-DELTA$(q, l, c, \mathcal{A})$
    DAWG-DELTA$(q, l, c, \mathcal{A})$
    if $\delta_{\mathcal{B}}(q, c) =$ NIL then
        do
            $q \leftarrow s\ell^*_{\mathcal{A}}(q)$
        while $q \neq$ NIL and $\delta_{\mathcal{A}}(q, c) =$ NIL
        if $q =$ NIL then
            $l \leftarrow 0, q \leftarrow root_{\mathcal{A}}$
        else $l \leftarrow len(q) + 1$
            $q \leftarrow \delta_{\mathcal{A}}(q, c)$
    else $l \leftarrow l + 1$
        $q \leftarrow \delta_{\mathcal{A}}(q, c)$
    return $(q, l)$
```

In order to evaluate the space complexity of ALGORITHM2, we observe that in the worst case, during the j-th iteration of its main for-loop, the sets $\texttt{F}^k_{j-k}$ and $\texttt{P}_{j-k}$, for $1 \leq k \leq m$, must be kept in memory to handle translocations. However, as explained before, we do not keep the values of $\texttt{F}^k_{j-k}$ explicitly but rather we maintain only their corresponding y-configurations of the automaton $\mathcal{A}(x)$. Thus, we need $\mathcal{O}(m)$-space for the last m configurations of the automaton and $\mathcal{O}(m^2)$-space to keep the last $m+1$ values of the sets $\texttt{P}_{j-k}$, since the maximum cardinality of each such set is $m+1$. Observe also that, although the size of the DAWG is linear in m, the $end\text{–}pos(\cdot)$ function can require $\mathcal{O}(m^2)$-space. Therefore, the total space complexity of the ALGORITHM2 is $\mathcal{O}(m^2)$.

4.2 Average-case time analysis

In this section we evaluate the average time complexity of our new automaton-based algorithm, assuming the uniform distribution and independence of characters in an alphabet Σ with $\sigma \geq 4$ characters.

In our analysis we do not include the time required for the computation of the DAWG and the *end–pos*$(\cdot)$ function, since they require $\mathcal{O}(m)$ and $\mathcal{O}(m^2)$ worst-case time, respectively, which turn out to be negligible if we assume that m is much smaller than n. Hence we evaluate only the searching phase of the algorithm.

Given an alphabet Σ of size $\sigma \geq 4$, for $j = 1, \ldots, n$, we consider the following nonnegative random variables over the sample space of the pairs of strings $x, y \in \Sigma^*$ of length m and n, respectively:

- $X(j)$ = the length $l_j \leq m$ of the longest factor of x which is a suffix of y_j;
- $Z(j) = |\mathtt{P}_j|$, where we recall that $\mathtt{P}_j = \{1 \leq i \leq m \mid x_i \overset{trd}{\sqsupseteq} y_j\}$.

Then the run-time of a call to ALGORITHM2 with parameters (x, y) is proportional to

$$\sum_{j=1}^{n} \left(Z(j-1) + X(j) + \sum_{h=1}^{X(j)} \left(X(j-h) + \sum_{k=1}^{X(j-h)} Z(j-h-k) \right) \right), \tag{3}$$

where the external summation refers to the main for-loop (at line 4), the second summation within it takes care of the internal for-loop at line 11, and the third summation refers to the inner for-loop at line 14.

Hence the average-case complexity of ALGORITHM2 is the expectation of (3), which, by linearity, is equal to

$$\sum_{j=1}^{n} \left(E(Z(j-1)) + E(X(j)) + E\left(\sum_{h=1}^{X(j)} X(j-h) \right) + E\left(\sum_{h=1}^{X(j)} \sum_{k=1}^{X(j-h)} Z(j-h-k) \right) \right). \tag{4}$$

where $E(\cdot)$ be the expectation function. Since $E(X(j)) \leq E(X(n-1))$ and $E(Z(j)) \leq E(Z(n-1))$, for $1 \leq j \leq n$,[2] by putting $X = X(n-1)$ and $Z = Z(n-1)$, the expression (4) gets bounded from above by

$$\sum_{j=1}^{n} \left(E(Z) + E(X) + E\left(\sum_{h=1}^{X} X \right) + E\left(\sum_{h=1}^{X} \sum_{k=1}^{X} Z \right) \right). \tag{5}$$

Let Z_i and X_h be the indicator variables defined for $i = 1, \ldots, m$ and $h = 1, \ldots, m$, respectively as

$$Z_i = \begin{cases} 1 & \text{if } i \in \mathtt{P}_n \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad X_h = \begin{cases} 1 & \text{if } X \geq h \\ 0 & \text{otherwise}, \end{cases}$$

Hence

$$Z = \sum_{i=1}^{m} Z_i, \;\; E(Z_i^2) = E(Z_i) = Pr\{x_i \overset{trd}{\sqsupseteq} y\},$$

$$X = \sum_{h=1}^{m} X_h, \quad \text{and} \quad E(X_h^2) = E(X_h) = Pr\{X \geq h\}.$$

[2] In fact, for $j = m+1, \ldots, n$ all the inequalities hold as equalities.

So that we have

$$\sum_{h=1}^{X} X = XX = \left(\sum_{h=1}^{m} X_h\right) \cdot \left(\sum_{k=1}^{m} X_k\right) = \sum_{h=1}^{m}\sum_{k=1}^{m} X_h X_k \,.$$

Therefore

$$\sum_{h=1}^{X}\sum_{k=1}^{X} Z = XXZ = \left(\sum_{h=1}^{m}\sum_{k=1}^{m} X_h X_k\right) \cdot Z,$$

which yields the following upper bound for (5):

$$\sum_{j=1}^{n}\left(E(Z) + E(X) + (E(Z)+1) \cdot \sum_{h=1}^{m}\sum_{k=1}^{m} E(X_h X_k)\right). \tag{6}$$

To estimate each of the terms $E(X_h X_k)$ in (6), we use the well-known Cauchy-Schwarz inequality which in the context of expectations assumes the form $|E(UV)| \leq \sqrt{E(U^2)E(V^2)}$, for any two random variables U and V such that $E(U^2)$, $E(V^2)$ and $E(UV)$ are all finite.

Then, for $1 \leq h \leq m$ and $1 \leq k \leq m$, we have

$$E(X_h X_k) \leq \sqrt{E(X_h^2)E(X_k^2)} = \sqrt{E(X_h)E(X_k)}\,. \tag{7}$$

From (7), it then follows that (6) is bounded from above by

$$\begin{aligned} &\sum_{j=1}^{n}\left(E(Z) + E(X) + (E(Z)+1) \cdot \sum_{h=1}^{m}\sum_{k=1}^{m} \sqrt{E(X_h)E(X_k)}\right) \\ &= \sum_{j=1}^{n}\left(E(Z) + E(X) + (E(Z)+1) \cdot \left(\sum_{h=1}^{m}\sqrt{E(X_h)}\right) \cdot \left(\sum_{k=1}^{m}\sqrt{E(X_k)}\right)\right) \\ &= \sum_{j=1}^{n}\left(E(Z) + E(X) + (E(Z)+1) \cdot \left(\sum_{h=1}^{m}\sqrt{E(X_h)}\right)^2\right). \end{aligned} \tag{8}$$

To better understand (8), we evaluate the expectations $E(X)$ and $E(Z)$ and the sum $\sum_{h=1}^{m}\sqrt{E(X_h)}$. To this purpose, it will be useful to estimate also the expectations $E(X_h) = Pr\{X \geq h\}$, for $1 \leq h \leq m$, and $E(X_k) = Pr\{x_i \overset{trd}{\sqsupseteq} y\}$, for $1 \leq k \leq m$.

Concerning $E(X_h) = Pr\{X \geq h\}$, we reason as follows. Since $y[n-h+1\,..\,n]$ ranges uniformly over a collection of σ^h strings and there can be at most $\min(\sigma^h, m-h+1)$ distinct factors of length h in x, the probability $Pr\{X \geq h\}$ that one of them matches $y[n-h+1\,..\,n]$ is at most $\min\left(1, \frac{m-h+1}{\sigma^h}\right)$. Hence, for $h = 1, \ldots, m$, we have

$$E(X_h) \leq \min\left(1, \frac{m-h+1}{\sigma^h}\right). \tag{9}$$

In view of (9), we have:

$$E(X) = \sum_{i=0}^{m} i \cdot Pr\{X = i\} = \sum_{i=1}^{m} Pr\{X \geq i\} \leq \sum_{i=1}^{m} \min\left(1, \frac{m-i+1}{\sigma^i}\right). \tag{10}$$

Let $\overline{h}$ be the smallest integer $1 \leq h < m$ such that $\frac{m-h+1}{\sigma^h} < 1$. Then, from (10), we have

$$\begin{aligned} E(X) &\leq \sum_{i=1}^{\overline{h}-1} 1 + \sum_{i=\overline{h}}^{m} \frac{m-i+1}{\sigma^i} \leq \overline{h} - 1 + (m - \overline{h} + 1)\sum_{i=\overline{h}}^{m}\frac{1}{\sigma^i} \\ &< \overline{h} - 1 + \frac{\sigma}{\sigma - 1} \cdot \frac{m - \overline{h} + 1}{\sigma^{\overline{h}}} < \overline{h} - 1 + \frac{\sigma}{\sigma-1} < \overline{h} + 1\,. \end{aligned} \tag{11}$$

Since $\frac{m-(\overline{h}+1)+1}{\sigma^{\overline{h}+1}} \geq 1$, then $\sigma^{\overline{h}+1} \leq m-(\overline{h}+1)+1 \leq m-1$, so that $\overline{h}+1 < \log_\sigma m$. Hence from (11) and $\overline{h}+1 < \log_\sigma m$, we obtain

$$E(X) < \log_\sigma m \,. \tag{12}$$

Likewise, from (9) and $\overline{h}+1 < \log_\sigma m$, we have

$$\begin{aligned}\sum_{h=1}^{m} \sqrt{E(X_h)} &\leq \sum_{h=1}^{m} \sqrt{\min\left(1, \frac{m-h+1}{\sigma^k}\right)} = \sum_{h=1}^{\overline{h}-1} 1 + \sum_{h=\overline{h}}^{m} \sqrt{\frac{m-h+1}{\sigma^h}} \\ &\leq \overline{h} - 1 + \sqrt{m-\overline{h}+1} \cdot \sum_{h=\overline{h}}^{m} \frac{1}{\sqrt{\sigma^h}} < \overline{h} - 1 + \frac{\sqrt{\sigma}}{\sqrt{\sigma}-1} \cdot \sqrt{\frac{m-\overline{h}+1}{\sigma^{\overline{h}}}} \\ &< \overline{h} - 1 + \frac{\sqrt{\sigma}}{\sqrt{\sigma}-1} \leq \overline{h} + 1 < \log_\sigma m \,,\end{aligned} \tag{13}$$

where $\overline{h}$ is defined as above.

Next we estimate $E(Z_i) = Pr\{x_i \overset{trd}{\sqsupseteq} y\}$, for $1 \leq i \leq m$. Let us denote by $\mu(i)$ the number of distinct strings which have a *utd*-match with a given string of length i and whose characters are pairwise distinct. Then $Pr\{x_i \overset{trd}{\sqsupseteq} y\} \leq \mu(i+1)/\sigma^{i+1}$. From the recursion

$$\begin{cases} \mu(0) &= \quad 1 \\ \mu(k+1) &= \quad \displaystyle\sum_{h=0}^{k} \mu(h) + \sum_{h=1}^{\lfloor \frac{k-1}{2} \rfloor} \mu(k-2h-1) \qquad (\text{for } k \geq 0)\,, \end{cases}$$

it is not hard to see that $\mu(i+1) \leq 3^i$, for $i = 1, 2, \ldots, m$, so that we have

$$E(Z_i) = Pr\{x_i \overset{trd}{\sqsupseteq} y\} \leq \frac{3^i}{\sigma^{i+1}} \,. \tag{14}$$

Then, concerning $E(Z)$, from (14) we have

$$E(Z) = E\left(\sum_{i=1}^{m} Z_i\right) = \sum_{i=1}^{m} E(Z_i) \leq \sum_{i=1}^{m} \frac{3^i}{\sigma^{i+1}} < \frac{1}{\sigma} \cdot \frac{1}{1-\frac{3}{\sigma}} = \frac{1}{\sigma-3} \leq 1 \tag{15}$$

(we recall that we have assumed $\sigma \geq 4$).

From (15), (12), and (13), it then follows that (4) is bounded from above by $n \cdot (1 + \log_\sigma m + 3\log_\sigma^2 m)$, yielding a $\mathcal{O}(n \log_\sigma^2 m)$ average-time complexity for our automaton based algorithm.

5 Conclusions

In this paper we proposed a first solution for the approximate string matching problem allowing for non-overlapping translocations when factors are restricted to be adjacent. Our algorithm has a $\mathcal{O}(nm^3)$-time and $\mathcal{O}(m^2)$-space complexity in the worst case, and $\mathcal{O}(n \log_\sigma^2)$-time complexity in the average-case. In our future research, we plan to improve the present result, by reducing the overall worst-case time complexity of the algorithm, and to generalize our algorithm also to the case of unrestricted non-overlapping translocations. We also plan to investigate whether this kind of approximate matching problem can be solved by using a skip-search filtering approach, recently used [6, 12] in the context of other kind of non standard matching problems.

References

1 Carlos Eduardo Rodrigues Alves, Alair Pereira do Lago, and Augusto F. Vellozo. Alignment with non-overlapping inversions in o(n^{3}logn)-time. *Electron. Notes Discret. Math.*, 19:365–371, 2005. doi:10.1016/j.endm.2005.05.049.

2 Domenico Cantone, Salvatore Cristofaro, and Simone Faro. Efficient matching of biological sequences allowing for non-overlapping inversions. In Raffaele Giancarlo and Giovanni Manzini, editors, *Combinatorial Pattern Matching - 22nd Annual Symposium, CPM 2011, Palermo, Italy, June 27-29, 2011. Proceedings*, volume 6661 of *Lecture Notes in Computer Science*, pages 364–375. Springer, 2011. doi:10.1007/978-3-642-21458-5_31.

3 Domenico Cantone, Salvatore Cristofaro, and Simone Faro. Efficient string-matching allowing for non-overlapping inversions. *Theor. Comput. Sci.*, 483:85–95, 2013. doi:10.1016/j.tcs.2012.06.009.

4 Domenico Cantone, Simone Faro, and Emanuele Giaquinta. Approximate string matching allowing for inversions and translocations. In Jan Holub and Jan Zdárek, editors, *Proceedings of the Prague Stringology Conference 2010, Prague, Czech Republic, August 30 - September 1, 2010*, pages 37–51. Prague Stringology Club, Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, 2010. URL: http://www.stringology.org/event/2010/p04.html.

5 Domenico Cantone, Simone Faro, and Emanuele Giaquinta. Text searching allowing for inversions and translocations of factors. *Discret. Appl. Math.*, 163:247–257, 2014. doi:10.1016/j.dam.2013.05.016.

6 Domenico Cantone, Simone Faro, and M. Oguzhan Külekci. An efficient skip-search approach to the order-preserving pattern matching problem. In Jan Holub and Jan Zdárek, editors, *Proceedings of the Prague Stringology Conference 2015, Prague, Czech Republic, August 24-26, 2015*, pages 22–35. Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, 2015. URL: http://www.stringology.org/event/2015/p04.html.

7 Zhi-Zhong Chen, Yong Gao, Guohui Lin, Robert Niewiadomski, Yang Wang, and Junfeng Wu. A space-efficient algorithm for sequence alignment with inversions and reversals. *Theor. Comput. Sci.*, 325(3):361–372, 2004. doi:10.1016/j.tcs.2004.02.040.

8 Da-Jung Cho, Yo-Sub Han, and Hwee Kim. Alignment with non-overlapping inversions and translocations on two strings. *Theor. Comput. Sci.*, 575:90–101, 2015. doi:10.1016/j.tcs.2014.10.036.

9 Maxime Crochemore and Wojciech Rytter. *Text Algorithms.* Oxford University Press, 1994. URL: http://www-igm.univ-mlv.fr/%7Emac/REC/B1.html.

10 Paul Cull and Tai Hsu. Recent advances in the walking tree method for biological sequence alignment. In Roberto Moreno-Díaz and Franz Pichler, editors, *Computer Aided Systems Theory - EUROCAST 2003, 9th International Workshop on Computer Aided Systems Theory, Las Palmas de Gran Canaria, Spain, February 24-28, 2003, Revised Selected Papers*, volume 2809 of *Lecture Notes in Computer Science*, pages 349–359. Springer, 2003. doi:10.1007/978-3-540-45210-2_32.

11 Fred Damerau. A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7(3):171–176, 1964. doi:10.1145/363958.363994.

12 Simone Faro and Arianna Pavone. An efficient skip-search approach to swap matching. *Comput. J.*, 61(9):1351–1360, 2018. doi:10.1093/comjnl/bxx123.

13 Yong Gao, Junfeng Wu, Robert Niewiadomski, Yang Wang, Zhi-Zhong Chen, and Guohui Lin. A space efficient algorithm for sequence alignment with inversions. In Tandy J. Warnow and Binhai Zhu, editors, *Computing and Combinatorics, 9th Annual International Conference, COCOON 2003, Big Sky, MT, USA, July 25-28, 2003, Proceedings*, volume 2697 of *Lecture Notes in Computer Science*, pages 57–67. Springer, 2003. doi:10.1007/3-540-45071-8_8.

14 Szymon Grabowski, Simone Faro, and Emanuele Giaquinta. String matching with inversions and translocations in linear average time (most of the time). *Inf. Process. Lett.*, 111(11):516–520, 2011. doi:10.1016/j.ipl.2011.02.015.

15 V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707–710, 1966.

16 J.R. Lupski. Genomic disorders: structural features of the genome can lead to dna rearrangements and human disease traits. *Trends Genet.*, 14:417–422, 1998.

17 M. Carrera andJ. Egozcue M. Oliver-Bonet andJ. Navarro and J. Benet. Aneuploid and unbalanced sperm in two translocation carriers: evaluation of the genetic risk. *Mol. Hum. Reprod.*, 8:958–963, 2002.

18 Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001. doi:10.1145/375360.375365.

19 H. Ogiwara, T. Kohno andH. Nakanishi, K. Nagayama, M. Sato, and J. Yokota. Unbalanced translocation, a major chromosome alteration causing loss of heterozygosity in human lung cancer. *Oncogene*, 27:4788–4797, 2008.

20 M. Schöniger and M. Waterman. A local algorithm for dna sequence alignment with inversions. *Bulletin of Mathematical Biology*, 54:521–536, 1992.

21 Augusto F. Vellozo, Carlos E. R. Alves, and Alair Pereira do Lago. Alignment with non-overlapping inversions in $O(n^3)$-time. In Philipp Bucher and Bernard M. E. Moret, editors, *Algorithms in Bioinformatics, 6th International Workshop, WABI 2006, Zurich, Switzerland, September 11-13, 2006, Proceedings*, volume 4175 of *Lecture Notes in Computer Science*, pages 186–196. Springer, 2006. doi:10.1007/11851561_18.

22 D. Warburton. De novo balanced chromosome rearrangements and extra marker chromosomes identified at prenatal diagnosis: clinical significance and distribution of breakpoints. *Am J. Hum Genet*, 49:995–1013, 1991.

23 B. Weckselblatt, K. E. Hermetz, and M.K. Rudd. Unbalanced translocations arise from diverse mutational mechanisms including chromothripsis. *Genome Research*, 25:937–947, 2015.