# Circular Sequence Comparison with $q$-grams

Roberto Grossi[1], Costas S. Iliopoulos[2], Robert Mercaş[2,3⋆], Nadia Pisanti[1], Solon P. Pissis[2⋆⋆], Ahmad Retha[2], and Fatima Vayani[2]

[1] Department of Computer Science, University of Pisa, Italy & Erable Team, INRIA
[2] Department of Informatics, King's College London, UK
[3] Department of Computer Science, Kiel University, Germany

**Abstract.** Sequence comparison is a fundamental step in many important tasks in bioinformatics. Traditional algorithms for measuring approximation in sequence comparison are based on the notions of distance or similarity, and are generally computed through sequence alignment techniques. As *circular* genome structure is a common phenomenon in nature, a caveat of specialized alignment techniques for circular sequence comparison is that they are computationally expensive, requiring from super-quadratic to cubic time in the length of the sequences. In this paper, we introduce a new distance measure based on $q$-grams, and show how it can be computed efficiently for circular sequence comparison. Experimental results, using real and synthetic data, demonstrate orders-of-magnitude superiority of our approach in terms of efficiency, while maintaining an accuracy very competitive to the state of the art.

## 1 Introduction

Circular molecular structures are present, in abundance, in all domains of life: bacteria, archaea, and eukaryotes; and in viruses. They can be composed of both amino and nucleic acids. The following is a superficial description of such occurrences. Exhaustive reviews can be found in [10] (proteins) and [19] (DNA).

Circular genomes and plasmids are found in bacteria and archaea. Whole-genome comparison is a very useful tool in identifying bacterial strains, as well as inferring phylogenies. The extended benefit of aligning plasmids is the ability to identify important genes, such as antibiotic resistance genes, thereby enabling their study and enhancement by genetic engineering techniques [12].

The most familiar examples of such structures in eukaryotes are mitochondrial (MtDNA) and plastid DNA. However, there exist other structures, called extrachromosomal circular DNA, which are described as one of the characteristics of genomic plasticity in eukaryotes [9]. MtDNA is generally conserved from parent to offspring, and so it can be used as an indicator of evolutionary relationships among species. The absence of recombination in these sequences allows

⋆ Supported by the P.R.I.M.E. programme of DAAD co-funded by BMBF and EU's 7th Framework Programme (grant 605728).
⋆⋆ Supported by a Research Grant (#RG130720) awarded by the Royal Society.

them to be used as simple tests of phylogenetic evolution, and their high mutation rate is a powerful discriminative feature [17, 33].

It is common knowledge that many viral genomes are circular. Multiple sequence alignment of viral genomes can be useful in the elucidation of novel sites of interest [4], as well as the inference of evolutionary relationships [3]. Viroids are plant pathogens that comprise very small single-stranded circular RNA. Their multiple sequence alignment could prove useful in the analysis of their secondary structures and pathogenicity [24].

Ribosomally synthesized circular proteins occur in both prokaryotes and eukaryotes [10]. An interesting phenomenon known to occur naturally in genes encoding linear protein structures is circular permutation [34]. This can be exemplified by swaposins: highly-similar proteins resulting from circularly permuted linear peptide sequences [29]. The ability to align linear sequences from circular proteins can significantly speed-up and enhance their analyses, and could also lead to the discovery of novel pairs of circularly permuted proteins.

Conventional tools to align circular sequences could yield an incorrectly high genetic distance between closely-related species. Indeed, when sequencing molecules, the position where a circular sequence starts can be totally arbitrary. Due to this arbitrariness, a suitable rotation of one sequence would give much better results for a pairwise alignment. A practical example of the benefit this can bring to sequence analysis is the following. Linearized human (NC_001807) and chimpanzee (NC_001643) MtDNA sequences do not start in the same region. Their pairwise sequence alignment using EMBOSS Needle [31] gives a similarity of 85.1% and consists of 1,195 gaps. However, taking different rotations of these sequences into account yields a much more significant alignment with a similarity of 91% and only 77 gaps. This example motivates the design of efficient algorithms that are devoted to the specific comparison of circular sequences, as they can be relevant in the analysis of organisms containing these structures.

*Our Problem.* We consider the pairwise circular sequence comparison problem. Under the edit distance model, it consists in finding an optimal linear alignment of two circular strings. This problem for two strings $x$ and $y$ of length $m$ and $n \geq m$, respectively, can be solved under the edit distance model in time $\mathcal{O}(nm \log m)$ [21]. Several other super-quadratic [23] and *approximate* quadratic-time [5] algorithms exist. Trivially, for molecular biology applications, the same problem can be solved in time $\mathcal{O}(nm^2)$ with scoring matrices and affine gap penalty scores. A *direct* application of pairwise circular sequence comparison is for multiple circular sequence alignment [1, 14, 24]. The latter has also been considered in [20] under the Hamming distance model.

To the best of our knowledge, there is no fast (that is, with sub-quadratic time complexity) and exact (or at least very accurate) algorithm for circular sequence comparison under some realistic model (that is, allowing *indels*). Taking into account edit distance rather than Hamming distance is computationally challenging as the search space for seeking similarity is wider. Moreover, filters that work for Hamming distance do not work in general for edit distance [28] as well. An exception to this are the $q$-gram filtering techniques [32] that have

successfully been used for string matching under the edit distance model (e.g. [7, 30, 26]), as well as for multiple local alignments both under the Hamming [27] and edit [26] distance model.

*Our Contribution.* We present new efficient $q$-gram-based methods for pairwise circular sequence comparison. Specifically, our contribution is threefold.

1. We introduce the *$\beta$-blockwise $q$-gram distance* between two strings $x$ and $y$, that is, a more powerful generalization of the $q$-gram distance introduced as a string distance measure in [32]. Intuitively, and similarly to [7, 30, 26], this generalization comprises partitioning $x$ and $y$ in $\beta$ *blocks* each, as evenly as possible, computing the $q$-gram distance between the corresponding block pairs, and then summing up the distances computed blockwise.
2. We present an algorithm based on the suffix array [22] that finds the rotation of $x$ such that the $\beta$-blockwise $q$-gram distance between the rotated $x$ and $y$ is minimal, in time and space $\mathcal{O}(\beta m + n)$, where $m = |x|$ and $n = |y|$, thereby solving *exactly* the circular sequence comparison problem under the $\beta$-blockwise $q$-gram distance measure. We also present a simple heuristic algorithm to solve an *approximate* version of the problem.
3. We present an experimental study, using real and synthetic data, which demonstrates orders-of-magnitude superiority of our approach, in terms of efficiency, while maintaining an accuracy very competitive to the *optimal* obtained after considering all rotations of $x$ against $y$ using EMBOSS Needle.

The paper is organized as follows. Section 2 gives some preliminary definitions, notation, and properties. Section 3 describes two algorithms, one is a heuristic approach and the other is an exact algorithm for circular sequence comparison. Section 4 shows the experimental results of performance and accuracy of our algorithms. Section 5 gives some concluding remarks and future proposals.

## 2 Definitions and Properties

We begin with a few definitions, following [11]. We think of a *string* $x$ of *length* $m$ as an array $x[0\,..\,m-1]$, where every $x[i]$, $0 \leq i < m$, is a *letter* drawn from some fixed *alphabet* $\Sigma$ of size $|\Sigma| = \mathcal{O}(1)$. By a *$q$-gram* we refer to any string $x \in \Sigma^q$. The *empty string* of length 0 is denoted by $\varepsilon$. A string $x$ is a *factor* of a string $y$ if there exist two strings $u$ and $v$, such that $y = uxv$. Let $x$ be a non-empty string of length $m$ and $y$ be a string. We say that there is an *occurrence* of $x$ in $y$, or, simply, that $x$ *occurs in* $y$, when $x$ is a factor of $y$. The *Parikh vector* associated with a string $w \in \Sigma^*$ is denoted by $\mathcal{P}(w)$ and represents a vector of size $|\Sigma|$, where each component denotes the number of occurrences in $w$ of the corresponding letter from $\Sigma$.

Consider the strings $x, y, u$, and $v$, such that $y = uxv$. If $u = \varepsilon$, then $x$ is a *prefix* of $y$. If $v = \varepsilon$, then $x$ is a *suffix* of $y$. We denote by $\mathsf{SA}$ the *suffix array* of $y$ of length $n$, that is, an integer array of size $n$ storing the starting positions of all (lexicographically) sorted suffixes of $y$, i.e. for all $1 \leq r < n$, we have

$y[\mathsf{SA}[r-1]\,..\,n-1] < y[\mathsf{SA}[r]\,..\,n-1]$ [22]. Let $\mathsf{lcp}(r,s)$ denote the length of the longest common prefix between $y[\mathsf{SA}[r]\,..\,n-1]$ and $y[\mathsf{SA}[s]\,..\,n-1]$, for all positions $r$, $s$ on $y$, and 0 otherwise. We denote by $\mathsf{LCP}$ the *longest common prefix* array of $y$ defined by $\mathsf{LCP}[r] = \mathsf{lcp}(r-1,r)$, for all $1 \le r < n$, and $\mathsf{LCP}[0] = 0$. The inverse $\mathsf{iSA}$ of the array $\mathsf{SA}$ is defined by $\mathsf{iSA}[\mathsf{SA}[r]] = r$, for all $0 \le r < n$. $\mathsf{SA}$, $\mathsf{iSA}$, and $\mathsf{LCP}$ of $y$ can be computed in time and space $\mathcal{O}(n)$ [15].

A circular string of length $m$ can be viewed as a traditional linear string which has the left- and right-most letters wrapped around and glued together in some way. Under this notion, the same circular string can be seen as $m$ different linear strings, which would all be considered equivalent. Given a string $x$ of length $m$, we denote by $x^i = x[i\,..\,n-1]x[0\,..\,i-1]$, $0 < i < m$, the $i$th *rotation* of $x$ and $x^0 = x$. Consider, for instance, the string $x = x^0 = \texttt{abababbc}$; this string has the following rotations: $x^1 = \texttt{bababbca}$, $x^2 = \texttt{ababbcab}$, and so on.

We give some further definitions following [32]. The *q-gram profile* of a string $x$ of length $m$ is the vector $G_q(x)$, where $q > 0$ and $G_q(x)[v]$ denotes the total number of occurrences of $v \in \Sigma^q$ in $x$. The $q$-gram distance between two strings $x$ and $y$ is defined as

$$D_q(x,y) = \sum_{v \in \Sigma^q} |G_q(x)[v] - G_q(y)[v]|\,. \tag{1}$$

Note that $D_q$ is a *pseudo-metric* as $D_q(x,y)$ can be 0 even if $x \neq y$. $D_q$ has the following properties [32] for all $x, y, z \in \Sigma^*$ of length at least $q$.

1. Positivity: $D_q(x,y) \ge 0$
2. Symmetry: $D_q(x,y) = D_q(y,x)$
3. Triangular inequality: $D_q(x,y) \le D_q(x,z) + D_q(z,y)$
4. $|(|x| - |y|)| \le D_q(x,y) \le |x| + |y| - 2q - 2$
5. $D_q(x_1x_2, y_1y_2) \le D_q(x_1,y_1) + D_q(x_2,y_2) + 2(q-1)$
6. $D_q(h(x),h(y)) \le D_q(x,y)$, for a non-length-increasing morphism $h$ on $\Sigma^*$.

For a given integer parameter $\beta \ge 1$, we define a generalization of the $q$-gram distance in (1) by partitioning $x$ and $y$ in $\beta$ *blocks* as evenly as possible, and using the $q$-gram distance within each pair of blocks, one from $x$ and one from $y$. The rationale is to enforce *locality* in the resulting distance. For the sake of presentation in the rest of the paper, we assume that the lengths $|x| = m$ and $|y| = n$ are both multiples of $\beta$, so that $x$ and $y$ are conceptually partitioned into $\beta$ blocks, each of size $m/\beta$ for $x$ and $n/\beta$ for $y$.

**Definition 1.** *Given strings $x$ of length $m$ and $y$ of length $n \ge m$ and integers $\beta \ge 1$ and $q > 0$, the* $\beta$-blockwise $q$-gram distance $D_{\beta,q}(x,y)$ *is defined as*

$$D_{\beta,q}(x,y) = \sum_{j=0}^{\beta-1} D_q\left(x\left[\frac{jm}{\beta}\,..\,\frac{(j+1)m}{\beta} - 1\right], y\left[\frac{jn}{\beta}\,..\,\frac{(j+1)n}{\beta} - 1\right]\right). \tag{2}$$

In this paper, we consider the following problem, where we search for the $i$th rotation of $x$ that minimizes its blockwise distance from $y$ as defined in (2). Ties are broken arbitrarily.

| Circular Sequence Comparison (CSC) |
|---|
| **Input:** strings $x$ and $y$ of lengths $m$ and $n \geq m$, respectively, and integers $\beta \geq 1$ and $q < m$ |
| **Output:** $i$ such that $D_{\beta,q}(x^i, y)$ is minimal |

## 3 Algorithms

We use the following result to first give a naïve solution to the CSC problem.

**Lemma 2 ([32]).** *If we have space $\mathcal{O}(|\Sigma|^q)$ available, then the $q$-gram distance $D_q(x, y)$ can be computed in time $\mathcal{O}(m + n)$ and extra space $\mathcal{O}(m + n)$, where $m = |x|$ and $n = |y|$.*

We then apply Lemma 2 to each pair of blocks of $x$ and $y$ separately.

**Lemma 3.** *If we have space $\mathcal{O}(|\Sigma|^q)$ available, then the $\beta$-blockwise $q$-gram distance $D_{\beta,q}(x, y)$ can be computed in time $\mathcal{O}(m + n)$ and extra space $\mathcal{O}(\frac{m+n}{\beta})$, where $m = |x|$ and $n = |y|$.*

The naïve algorithm, denoted by nCSC, computes for $x' = xx$ the values

$$\delta_i = D_{\beta,q}(x'[i \,..\, i + m - 1], y),$$

for all $0 \leq i < m$; we report position $i$ such that $\delta_i$ is minimal. This requires the application of Lemma 3, $m$ times. Therefore, we obtain the following.

**Lemma 4.** *If we have space $\mathcal{O}(|\Sigma|^q)$ available, then algorithm nCSC solves the CSC problem in time $\mathcal{O}(m(m + n))$ and extra space $\mathcal{O}(\frac{m+n}{\beta})$.*

### 3.1 Algorithm hCSC: a Heuristic Algorithm

Here we give a simple heuristic algorithm, denoted by hCSC, to solve the CSC problem faster than nCSC, and return an approximation of the best rotation.

**Step 1**: We split $x' = xx$ in $2\beta$ non-overlapping string *blocks* of length $m/\beta$. We obtain strings $x_0, x_1, \ldots, x_{2\beta-1}$, such that $x_i = x'[\frac{im}{\beta} \,..\, \frac{(i+1)m}{\beta} - 1]$, for all $0 \leq i < 2\beta$. We split $y$ in $\beta$ non-overlapping string blocks of length $n/\beta$. We obtain strings $y_0, y_1, \ldots, y_{\beta-1}$, such that $y_i = y[\frac{in}{\beta} \,..\, \frac{(i+1)n}{\beta} - 1]$, for all $0 \leq i < \beta$.

**Step 2**: For a given sequence $x_j, \ldots, x_{j+\beta-1}$ of strings and $y$, we compute the $\beta$-blockwise $q$-gram distance as follows

$$\delta_j = D_{\beta,q}(x'[\frac{jm}{\beta} \,..\, \frac{jm}{\beta} + m - 1], y) = \sum_{i=0}^{\beta-1} D_q(x_{j+i}, y_i).$$

We compute $\delta_j$, for all $0 \leq j \leq \beta$. We choose $j_{best} = j$ such that $\delta_j$ is minimal, for all $0 \leq j \leq \beta$. In other words, we have found a *window* of length $m$ starting at position $j_{best}$, such that $(j_{best} + 1) \bmod (m/\beta) = 0$, consisting of $\beta$ blocks of length $m/\beta$ each, that minimizes its $\beta$-blockwise $q$-gram distance from $y$.

**Step 3**: To perform a refinement on the position of the window, we consider all starting positions included in the two blocks starting at positions $j_{best}$ and $j_{best}-m/\beta$. This includes $2m/\beta-1$ starting positions in total—we do not need to consider position $j_{best}-m/\beta$ as this was already considered by another window in Step 2. Similarly to Step 2, we obtain the $\beta$-blockwise $q$-gram distance $\delta_i$ between the window starting at position $i$ and $y$, for all $j_{best}-m/\beta < i \leq j_{best}+m/\beta-1$. We report position $i_{best}=i$ such that $\delta_i$ is minimal, for all $j_{best}-m/\beta < i \leq j_{best}+m/\beta-1$.

*Analysis.* Step 1 can be done trivially in time $\mathcal{O}(m+n)$. If we have space $\mathcal{O}(|\Sigma|^q)$ available, then, by Lemma 2, $D_q(x_{j+i}, y_i)$ can be computed in time $\mathcal{O}(\frac{m+n}{\beta})$. By Lemma 3, $\delta_j$ can be computed in time $\mathcal{O}(\beta(\frac{m+n}{\beta})) = \mathcal{O}(m+n)$. Hence, Step 2 can be done in time $\mathcal{O}(\beta(m+n))$. In Step 3, the blockwise $q$-gram distance $\delta_i$ between a single window and $y$ can be computed in time $\mathcal{O}(\beta(\frac{m+n}{\beta})) = \mathcal{O}(m+n)$. There exist $2m/\beta-1$ such windows. Hence, Step 3 can be done in time $\mathcal{O}(\frac{m(m+n)}{\beta})$. Overall, the algorithm requires time $\mathcal{O}(\beta(m+n)+\frac{m(m+n)}{\beta})$ and space $\mathcal{O}(|\Sigma|^q+m+n)$.

For practical purposes, setting $\beta=\mathcal{O}(\sqrt{m})$ and $q=\mathcal{O}(\log_{|\Sigma|} m)$ gives an algorithm with time complexity $\mathcal{O}(\sqrt{m}(m+n))$ and space complexity $\mathcal{O}(m+n)$.

### 3.2 Algorithm saCSC: an Exact Suffix-Array-based Algorithm

The above heuristics hCSC does not guarantee to find the exact value $i$, for which $\delta_i = D_{\beta,q}(x^i, y)$ is minimal. In particular, when we identify in Step 2 $j_{best}$, that is, the $j$ for which $\delta_j$ is minimal, we take into account only the values of $j$ such that $(j+1) \bmod (m/\beta) = 0$. Thus, Step 3 cannot guarantee that $i_{best}$, the local minimum obtained by shifting the window $m/\beta$ positions to the right and left of $j_{best}$, is minimal for all $0 \leq i < m$. In this section, we give a fast and exact algorithm, denoted by saCSC, to find $i$ such that $\delta_i = D_{\beta,q}(x^i, y)$ is minimal, based on the suffix array (see Section 2).

We partially follow the idea from [13]. This work investigates the string matching problem in the setting of $k$-abelian equivalences: two strings are considered $k$-abelian equivalent for some positive integer $k$, if they have the same length and share the same factors of length at most $k$, including multiplicities. Note that if $k$ is greater than or equal to the string's length, then the strings must be equal. A version of this result, called extended $k$-abelian equivalence, focuses only on the factors of length $k$. By setting $k=q$, it is quite straightforward to notice the equivalence with $q$-grams. Therefore, in order to avoid confusion we will refer to the former notion from now on as *$q$-abelian equivalence*.

In [13], the authors propose a linear-time algorithm to solve the string matching problem when looking at $q$-abelian equivalent strings: given a string $x$ of length $m$, a string $y$ of length $n \geq m$, and a positive integer $q < m$, all factors of $y$ that are $q$-abelian equivalent to $x$ can be found in time and space $\mathcal{O}(m+n)$. The idea of the algorithm in [13] consists in constructing the suffix array of the string $xy$, and ranking sets of identical $q$-length prefixes of suffixes in the suffix

array in the order of their appearance. Then it constructs new strings based on this ranking, and solves the problem as in the *jumbled matching* case [6], i.e. identifying all factors of $y$ that have the same Parikh vector as $x$.

*Basic Algorithm for $\beta = 1$.* We construct the suffix array of the string $xxy$ and assign a *rank* to the prefix with length $q$ of each suffix with length at least $q$, based on its order in the suffix array. That is, the first $i_0$ suffixes in the suffix array, all sharing the same prefix of length at least $q$, will get rank 0; the next $i_1$ suffixes sharing the same prefix of length at least $q$, different from the previous one, will get rank 1, and so on. Next, based on this ranking, we construct two new strings $x'$ of length $2m-q+1$ and $y'$ of length $n-q+1$, such that $x'[i] = j$, if $j$ is the rank of the $q$-length prefix of the $(i+1)$th suffix of $xx$ in the suffix array of $xxy$ (the same goes for $y$). It is not difficult to see that the ranks go up at most to value $m+n-q+1$. However, we can reduce this value to $m+2$ by introducing two new ranks $a_x$ and $a_y$: we can conceptually replace by $a_x$ every letter of $x'$ that does not occur in $y'$, and by $a_y$ every letter of $y'$ that does not occur in $x'$. Hence we can consider that the new strings $x'$ and $y'$ are defined over an integer alphabet of size *at most* $\min(n-q+1, m)+2 \leq m+2$.

*Example 5.* Let $x = \texttt{GAGTCTA}$, $y = \texttt{TCTAGCG}$, and $q = 3$. We denote $xxy$ by $z$.

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $z[i]$ | G | A | G | T | C | T | A | G | A | G | T | C | T | A | T | C | T | A | G | C | G |
| SA$[i]$ | 6 | 17 | 1 | 8 | 13 | 19 | 4 | 15 | 11 | 20 | 0 | 7 | 18 | 2 | 9 | 5 | 16 | 12 | 3 | 14 | 10 |
| LCP$[i]$ | 0 | 2 | 2 | 6 | 1 | 0 | 1 | 4 | 3 | 0 | 1 | 7 | 1 | 1 | 5 | 0 | 3 | 2 | 1 | 5 | 4 |
| $x'[i]$ | $a_x$ | $a_x$ | $a_x$ | 2 | $a_x$ | 1 | $a_x$ | $a_x$ | $a_x$ | $a_x$ | $a_x$ | 0 | | | | | | | | | |
| $y'[i]$ | 2 | 0 | 1 | $a_y$ | $a_y$ | | | | | | | | | | | | | | | | |

$x'[3] = y'[0] = 2$ denotes that $x[3\,..\,5] = y[0\,..\,2] = \texttt{TCT}$. $x'[0] = a_x$ denotes that $x[0\,..\,2] = \texttt{GAG}$ does not occur in $y$. □

We observe that when identifying the $q$-gram distance between two blocks, we can apply the idea in [13], with the only difference that we should also maintain a Parikh vector that stores the *differences* between the number of occurrences of $q$-grams in the current block of $xx$ and $y$ (in fact the new letters given by the ranks). Moreover, at the time of the construction of $y'$, we also construct a Parikh vector $\mathcal{P}(y')$, storing, for each letter of $y'$, the number of its occurrences in $y'$. Notice that $|\mathcal{P}(y')| \leq m+2$. Later on, when computing the $q$-gram distances, we can construct another vector diff to store the letter differences between $\mathcal{P}(y')$ and the Parikh vector covering the $m-q+1$ letters of $x'$ associated with a window of length $m$ on the string $xx$. This gives us the current Parikh difference and, in fact, represents the $q$-gram distance between the two analyzed blocks, where $|\mathsf{diff}| \leq m+2$. Apart from these, we only need another vector $\delta$ of size $m$, which stores at each position $i$ the actual $q$-gram distance $\delta_i$ between $y$ and the window starting at position $i$ in $xx$, which is the $i$th rotation $x^i$ of $x$.

We use a sliding window of length $m$ to maintain the above information. When the window is shifted one position to the right, we have to add to the

difference-vector $\mathsf{diff}$ the previous first element of the window, and deduct from it the current last element of it. The distance $\delta_i$ between $y'$ and the factor of $x'$ starting at position $i$ is thus updated using, in addition, the value of the $q$-gram distance $\delta_{i-1}$ as follows. If, after adding the previous first element to the vector, we have a non-positive value at this position, we update the distance by decreasing the previous value by 1; otherwise, we increase it by 1. If, after deducting the current last element to the vector, we have a non-negative value at this position, we update the distance by decreasing the previous value by 1; otherwise, we increase it by 1. The distance will never be less than the number of occurrences of $a_y$. Furthermore, if the previous first element was $a_x$, the new distance decreases by 1, and for every newly added $a_x$, it increases by 1. As these operations require constant time, after going once through $x'$ with $y'$, we obtain the list of distances $\delta_i$ from $y$ to each rotation $x^i$ in linear time.

We are now able to give a more formal description of the steps to solve the CSC problem for $\beta = 1$, which follow a dynamic programming scheme.

**Step 1**: Construct the $\mathsf{SA}$, $\mathsf{iSA}$, and $\mathsf{LCP}$ of $xxy$. Rank the $q$-length prefixes of suffixes using $\mathsf{LCP}$-array queries. Construct $x'$ and $y'$, as well as $\mathcal{P}(y')$, the Parikh vector storing, for each letter of $y'$, the number of its occurrences in $y'$; make proper use of letters $a_x$ and $a_y$, the ranks that do not occur in either $y'$ or $x'$, respectively. Further, create $\mathsf{diff} = \mathcal{P}(y')$ and $\delta_0 = \sum_{i=0}^{|\mathcal{P}(y')|-1} \mathcal{P}(y')[i]$.

**Step 2**: Read the first $m - q + 1$ letters of $x'$, which constitute our sliding window of length $m$ on the string $xx$. When reading letter $x'[i]$, update $\mathsf{diff}$ by decreasing by 1 the value of the newly read letter, and update $\delta_0$, by either increasing the current value of the distance when there were read too many of the current letters, or decreasing it, when more of these letters still occur in $y'$

$$\mathsf{diff}[x'[i]] = \mathsf{diff}[x'[i]] - 1 \qquad \text{and} \qquad \delta_0 = \begin{cases} \delta_0 - 1, \text{ if } \mathsf{diff}[x'[i]] \geq 0 \\ \delta_0 + 1, \text{ if } \mathsf{diff}[x'[i]] < 0. \end{cases}$$

**Step 3**: Let $i$ be the current position in $x'$ and repeat this step, one position at a time. Shift the window to the right, update the information for $\mathsf{diff}$

$$\mathsf{diff}[x'[i]] = \mathsf{diff}[x'[i]] + 1 \qquad \text{and} \qquad \mathsf{diff}[x'[i+m]] = \mathsf{diff}[x'[i+m]] - 1,$$

and calculate $\delta_{i+1}$, based on this information, sequentially applying the two following rules

$$\delta_{i+1} = \begin{cases} \delta_i - 1, \text{ if } \mathsf{diff}[x'[i]] \leq 0 \\ \delta_i + 1, \text{ if } \mathsf{diff}[x'[i]] > 0 \end{cases}$$

$$\delta_{i+1} = \begin{cases} \delta_{i+1} - 1, \text{ if } \mathsf{diff}[x'[i+m]] \geq 0 \\ \delta_{i+1} + 1, \text{ if } \mathsf{diff}[x'[i+m]] < 0. \end{cases}$$

*Correctness.* Steps 1 and 2 are trivially correct as at the end of them we have that $\mathsf{diff}$ is the difference between $\mathcal{P}(y')$ and the vector corresponding to the window. These operations follow directly from the definitions of $\mathsf{SA}$ and $\mathsf{LCP}$, and are followed by a simple traversal of the suffix array in order to obtain the ranks and create the $\mathcal{P}(y')$ and $\mathsf{diff}$ vectors. Also, $\delta_0$, which was initially

the number of letters in $y'$, is decreasing as long as the difference between the vectors for a specific letter is non-negative (thus, we still have more occurrences of that letter in $y'$ compared to the window), and increasing otherwise. In Step 3, we update the difference vector by increasing the value at position $x'[i]$ and decreasing that of the new letter $x'[i+m]$ added to the difference. The $q$-gram distance at that position is based on the values of the newly obtained difference vector, as well as the $q$-gram distance at the previous position: if $\mathsf{diff}[x'[i]] \leq 0$, then obviously there were more letters $x'[i]$ in $y'$ than in the window, thus we need to decrease, while, if $\mathsf{diff}[x'[i]] > 0$, then there were at least as many letters $x'[i]$ in the window as in $y'$, and taking one out increases the distance. The complementary reasoning applies to the newly added letter $x'[i+m]$. The value of $\delta_i$ never goes below the number of occurrences of $a_y$ in $y'$ (it is equal to that, when all other elements of $\mathsf{diff}$ are 0) and represents the $q$-gram distance between $y$ and $x^i$, the corresponding window of length $m$ starting at position $i$ in $xx$.

*Analysis.* In Step 1, constructing $\mathsf{SA}$, $\mathsf{iSA}$, and $\mathsf{LCP}$ of $xxy$ can be done in time and extra space $\mathcal{O}(m+n)$ (Section 2). Furthermore, the construction of $x'$, $y'$, $\mathcal{P}(y')$, $\mathsf{diff}$, and $\delta_0$ is done with the same time and space cost. In Step 2, updating $\mathsf{diff}$ and $\delta_0$ after reading each letter takes constant time, as we execute two operations, thus $\mathcal{O}(m)$ in total. Constant time is required for each iteration in Step 3 to compute the value of $\delta_i$, $1 \leq i < m$, and update $\mathsf{diff}$, since a constant number of operations are executed, thus $\mathcal{O}(m)$ in total. Hence, we can solve the CSC problem for $\beta = 1$ in time and space $\mathcal{O}(m+n)$.

*General Algorithm for $\beta \geq 1$.* We can now generalize this algorithm to solve the CSC problem for any $\beta \geq 1$, which gives algorithm $\mathsf{saCSC}$. We maintain a Parikh vector for each block, and apply the above basic algorithm for each pair of blocks, computing their $q$-gram distance. If we denote by $\mathcal{P}_j(y')$ and $\mathsf{diff}_j$, for all $0 \leq j < \beta$, the $\beta$ Parikh vectors of $y'$ and of the $q$-gram distances, respectively, as well as by $\delta_{i,j}$ the $q$-gram distance between the $j$th block of $y$ and $x^i$, then the updates will be given by the formulae below. Hence, at each position $i < m$, we can update all of the $\beta$ Parikh vectors corresponding to the blocks, as previously described, in time $\mathcal{O}(\beta)$. As an example, see here the modification of the previous Step 3, with the other two steps being easily adapted in a similar fashion.

**Step 3′:** When shifting the window one position to the right from position $i$, update the information for every $\mathsf{diff}_j$, where $0 \leq j < \beta$, as follows

$$\mathsf{diff}_j[x'[i+\frac{jm}{\beta}]] = \mathsf{diff}_j[x'[i+\frac{jm}{\beta}]] + 1$$
$$\mathsf{diff}_j[x'[i+\frac{(j+1)m}{\beta}]] = \mathsf{diff}_j[x'[i+\frac{(j+1)m}{\beta}]] - 1,$$

and calculate $\delta_{i+1,j}$, based on this information, sequentially applying the two following rules

$$\delta_{i+1,j} = \begin{cases} \delta_{i,j} - 1, \text{ if } \mathsf{diff}_j[x'[i+\frac{jm}{\beta}]] \leq 0 \\ \delta_{i,j} + 1, \text{ if } \mathsf{diff}_j[x'[i+\frac{jm}{\beta}]] > 0 \end{cases}$$

$$\delta_{i+1,j} = \begin{cases} \delta_{i+1,j} - 1, \text{ if } \mathsf{diff}_j[x'[i + \frac{(j+1)m}{\beta}]] \geq 0 \\ \delta_{i+1,j} + 1, \text{ if } \mathsf{diff}_j[x'[i + \frac{(j+1)m}{\beta}]] < 0. \end{cases}$$

**Theorem 6.** *Algorithm saCSC solves the CSC problem in $\mathcal{O}(\beta m + n)$ time and space.*

## 4 Experimental Results

We implemented algorithms nCSC, hCSC, and saCSC as the program CSC. Given one of the three methods, two sequences $x$ and $y$ in (Multi)FASTA format, the number $\beta$ of blocks, and the length $q$ of the $q$-grams, CSC finds the rotation of $x$ (or an approximation of it) that minimizes its $\beta$-blockwise $q$-gram distance from $y$. The implementation is distributed under the GNU General Public License (GPL), and it is available at `http://github.com/solonas13/csc`. For comparison purposes, we also implemented a naïve algorithm that compares all rotations of $x$ against $y$ using the Needleman-Wunsch algorithm [25] with substitution matrices and affine gap penalty scores [18]; we denote this by cNW.

The following experiments were conducted on a desktop computer using one core of Intel® Core™ i7-2600 CPU at 3.4GHz and 12GB of RAM under 64-bit GNU/Linux. All programs were compiled with `gcc` version 4.7.3. We used both synthetic data (Sections 4.1–4.2) and real data (Section 4.3). All input datasets referred to in this section are publicly maintained at the same web-site.

### 4.1 Accuracy

We began with simulating three DNA sequence datasets using INDELible [16], with each dataset consisting of 12 sequences, each of length approximately 2,500 base pairs (bp). INDELible produces linear sequences with substitutions, insertions, and deletions at rates defined by the user. Three unique substitution rates were set per dataset using the substitution model JC69 (Jukes-Cantor, 69): 5%, 20%, and 35%. The insertion and deletion rates were set, respectively, to 4% and 6%, relative to substitution rate of 1, similar to those observed in MtDNA in primates and mammals [14]. We refer to these datasets as *Original*.

To allow for comparison of the performance of the algorithms in realigning randomly-rotated sequences, which should be similar to those obtained from sequencing circular DNA structures, such as MtDNA, one random rotation was generated from each sequence in all datasets, creating new datasets which will be referred to as *Random*. Using the three *Random* datasets allowed us to test the accuracy of hCSC and saCSC; notice that nCSC and saCSC always return the same rotation. For each *Random* dataset, an all-against-all sequence comparison was performed. That is, all possible pairs, 66 in total, of sequences in each dataset were input to both hCSC and saCSC. $\beta$ was set to 50 and $q$ was set to 6. The resultant re-rotated sequences were aligned using EMBOSS Needle [31] and the similarity scores were compared to those of the *Original* and *Random* datasets, which were input directly to EMBOSS Needle. The results can be found in Fig. 1.

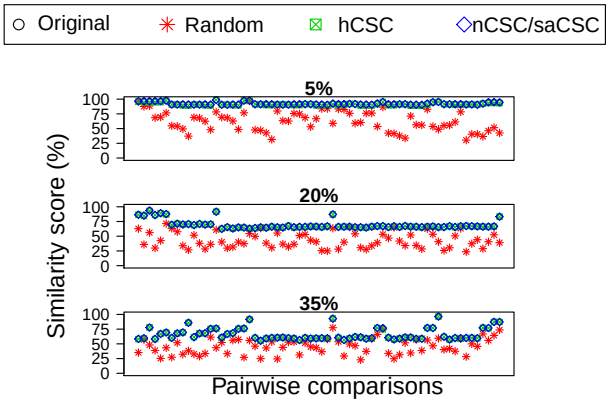


Fig. 1: Accuracy comparison for substitution rates 5%, 20%, and 35%; the black, green, and blue points coincide implying that algorithms hCSC, nCSC, and saCSC return the rotation maximizing the similarity score for all pairwise comparisons

The results show that: (a) hCSC and saCSC yield significantly improved similarity scores compared to those obtained from inputting *Random* datasets directly to EMBOSS Needle; and (b) hCSC and saCSC yield similarity scores that are identical or almost identical—notice that the black (Original), green (hCSC), and blue (nCSC/saCSC) points coincide—to those obtained from inputting *Original* datasets directly to EMBOSS Needle. This implies that algorithms hCSC, nCSC, and saCSC return the rotation maximizing the similarity score for all pairwise comparisons.

Hence what we establish here is that the introduced distance measure coupled with the respective algorithms consistently yield a very high accuracy, compared to the standard measure [25, 18, 31], for both *low* and *high* substitution rates.

### 4.2 Time Performance

We then compared the time performance of the algorithms. Each algorithm was given a pair of randomly generated sequences starting from $m = n = 50$ bp and doubling 8 times to a length of $m = n = 12,800$ bp. It was expected that the slowest algorithm would be cNW which runs in time $\mathcal{O}(nm^2)$. Then it would be algorithm nCSC which runs in time $\mathcal{O}(m(m+n))$, then algorithm hCSC, which runs in time $\mathcal{O}(\beta(m+n) + \frac{m(m+n)}{\beta})$, and lastly algorithm saCSC, which runs in time $\mathcal{O}(\beta m + n)$.

Initially, $\beta$ was set to $\lceil\sqrt{m}\rceil$ and $q$ was set to $\lceil \log m / \log |\Sigma| \rceil$. The results in Fig. 2 demonstrate orders-of-magnitude superiority of saCSC compared to cNW and nCSC, confirming our theoretical findings. hCSC is the second fastest. Although $\beta$ was set to $\lceil\sqrt{m}\rceil$, saCSC clearly outperforms hCSC, due to the use of a highly optimized implementation of the suffix-array construction, thus high-

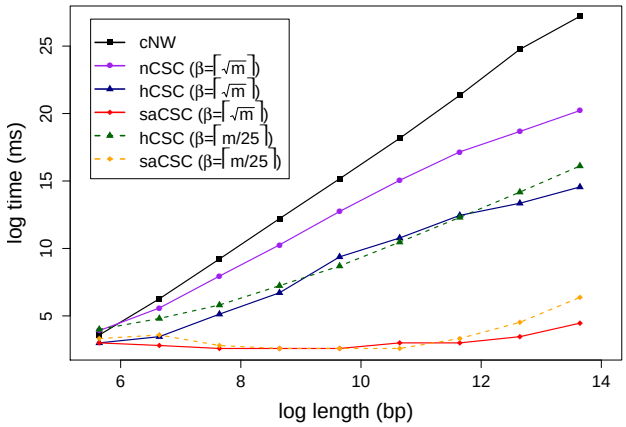


Fig. 2: Elapsed-time comparison

lighting the importance of suitably implemented data structures such as suffix arrays.

Since the time complexities of hCSC and saCSC depend on $\beta$, we repeated the same experiment with these two algorithms setting $\beta$ to $\lceil m/25 \rceil$ and $q$ to $\lceil \log m / \log |\Sigma| \rceil$—notice that $q$ does not affect the time efficiency of the algorithms. The results in Fig. 2 show that hCSC and saCSC are still the fastest, even though $m = \mathcal{O}(\beta)$, and that saCSC is clearly the fastest of all. As expected for $m = \mathcal{O}(\beta)$, we observe that hCSC and saCSC become gradually slower when $m$ grows.

More algorithms could have been included in the comparison but their (at least) quadratic time complexity [23, 5] prevents them to compete with saCSC.

### 4.3 Application to Real Data

As the input dataset, we used two real sequences from GenBank [2]: human (NC_001807) and chimpanzee (NC_001643) MtDNA sequences. The MtDNA genome size for human is 16,571 bp and for chimpanzee is 16,554 bp. Their pairwise sequence alignment using EMBOSS Needle with the default parameters (Gap opening penalty 10.0 and Gap extension penalty 0.5) gives a similarity of 85.1%. We used saCSC to obtain the rotation of NC_001807 that minimizes its $\beta$-blockwise $q$-gram distance from NC_001643, for $\beta = 850$ and $q = 5$. We obtained rotation 578 of NC_001807 and used EMBOSS Needle to align this rotation with NC_001643. EMBOSS Needle gave a significantly improved similarity of 91%. This rotation is exactly the rotation we obtained after naïvely searching for the rotation of NC_001807 that maximizes similarity using cNW. Finding this rotation took approximately 28 hours for cNW and only a quarter

of a second for saCSC. We repeated this experiment with the human and gorilla (NC_011120) MtDNA sequences. The MtDNA genome size for gorilla is 16,412 bp. Their pairwise sequence alignment using EMBOSS Needle with the default parameters gives a similarity of 83.5%. After using saCSC to rotate sequence NC_001807, EMBOSS Needle gave a significantly improved similarity of 88.4%.

## 5 Final Remarks

In this paper, we introduced a new distance measure for sequence comparison based on $q$-grams, and showed how it can be applied *effectively* and computed *efficiently* for circular sequence comparison. Furthermore, we presented an experimental study, using both real and synthetic data, demonstrating orders-of-magnitude superiority of our approach, in terms of efficiency, while maintaining an accuracy which is very competitive to the state of the art.

Our immediate target is twofold: (a) implement algorithm saCSC in BEAR [1], a state-of-the-art tool for improving multiple circular sequence alignment; and (b) evaluate alternative methods for circular sequence comparison based on local alignment heuristics [8].

## References

1. Barton, C., Iliopoulos, C.S., Kundu, R., Pissis, S.P., Retha, A., Vayani, F.: Accurate and efficient methods to improve multiple circular sequence alignment. In: 14th SEA. LNCS, vol. 9125, pp. 247–258 (2015)
2. Benson, D.A., Karsch-Mizrachi, I., Lipman, D.J., Ostell, J., Rapp, B.A., Wheeler, D.L.: GenBank. Nucleic Acids Res 28(1), 15–18 (2000)
3. Bray, N., Pachter, L.: MAVID: constrained ancestral alignment of multiple sequences. Genome Res 14(4), 693–699 (2004)
4. Brodie, R., Smith, A.J., Roper, R.L., Tcherepanov, V., Upton, C.: Base-By-Base: Single nucleotide-level analysis of whole viral genome alignments. BMC Bioinform 5(1), 96 (2004)
5. Bunke, H., Buhler, U.: Applications of approximate string matching to 2D shape recognition. Pattern Recognit 26(12), 1797–1812 (1993)
6. Burcsi, P., Cicalese, F., Fici, G., Lipták, Z.: Algorithms for jumbled pattern matching in strings. Int J Found Comput Sci 23(2), 357–374 (2012)
7. Burkhardt, S., Crauser, A., Ferragina, P., Lenhof, H.P., Rivals, E., Vingron, M.: $q$-gram based database searching using a suffix array (QUASAR). In: 3rd RECOMB. pp. 77–83 (1999)
8. Chao, K.M., Zhang, J., Ostell, J., Miller, W.: A tool for aligning very similar DNA sequences. CABIOS 13(1), 75–80 (1997)
9. Cohen, S., Houben, A., Segal, D.: Extrachromosomal circular DNA derived from tandemly repeated genomic sequences in plants. Plant J 53(6), 1027–1034 (2008)
10. Craik, D.J., Allewell, N.M.: Thematic minireview series on circular proteins. J Biol Chem 287(32), 26999–27000 (2012)
11. Crochemore, M., Hancart, C., Lecroq, T.: Algorithms on Strings. Cambridge University Press, New York, NY, USA (2007)

12. Del Castillo, C.S., Hikima, J.i., Jang, H.B., Nho, S.W., Jung, T.S., Wongtavatchai, J., Kondo, H., Hirono, I., Takeyama, H., Aoki, T.: Comparative sequence analysis of a multidrug-resistant plasmid from Aeromonas hydrophila. Antimicrob Agents Chemother 57(1), 120–129 (2013)
13. Ehlers, T., Manea, F., Mercaş, R., Nowotka, D.: $k$-abelian pattern matching. In: 18th DLT. LNCS, vol. 8633, pp. 178–190 (2014)
14. Fernandes, F., Pereira, L., Freitas, A.T.: CSA: An efficient algorithm to improve circular DNA multiple alignment. BMC Bioinform 10(1), 1–13 (2009)
15. Fischer, J.: Inducing the LCP-Array. In: 12th WADS. LNCS, vol. 6844, pp. 374–385 (2011)
16. Fletcher, W., Yang, Z.: INDELible: A flexible simulator of biological sequence evolution. Mol Biol Evol 26(8), 1879–1888 (2009)
17. Goios, A., Pereira, L., Bogue, M., Macaulay, V., Amorim, A.: mtDNA phylogeny and evolution of laboratory mouse strains. Genome Res 17(3), 293–298 (2007)
18. Gotoh, O.: An improved algorithm for matching biological sequences. J Mol Biol 162(3), 705–708 (1982)
19. Helinski, D.R., Clewell, D.B.: Circular DNA. Annu Rev Biochem 40(1), 899–942 (1971)
20. Lee, T., Na, J.C., Park, H., Park, K., Sim, J.S.: Finding consensus and optimal alignment of circular strings. Theor Comput Sci 468, 92–101 (2013)
21. Maes, M.: On a cyclic string-to-string correction problem. IPL 35(2), 73–78 (1990)
22. Manber, U., Myers, E.W.: Suffix arrays: A new method for on-line string searches. SIAM J Comput 22(5), 935–948 (1993)
23. Marzal, A., Barrachina, S.: Speeding up the computation of the edit distance for cyclic strings. In: 15th ICPR. vol. 2, pp. 891–894 (2000)
24. Mosig, A., Hofacker, I.L., Stadler, P.F.: Comparative Analysis of Cyclic Sequences: Viroids and other Small Circular RNAs. In: GCB. LNI, vol. 83, pp. 93–102. GI (2006)
25. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J Mol Biol 48(3), 443–453 (1970)
26. Peterlongo, P., Sacomoto, G.T., do Lago, A.P., Pisanti, N., Sagot, M.F.: Lossless filter for multiple repeats with bounded edit distance. Algorithm Mol Biol 4(3) (2009)
27. Peterlongo, P., Pisanti, N., Boyer, F., do Lago, A.P., Sagot, M.F.: Lossless filter for multiple repetitions with Hamming distance. JDA 6(3), 497–509 (2008)
28. Pisanti, N., Giraud, M., Peterlongo, P.: Filters and seeds approaches for fast homology searches in large datasets. In: Elloumi, M., Zomaya, A.Y. (eds.) Algorithms in computational molecular biology, chap. 15, pp. 299–320. John Wiley & sons (2010)
29. Ponting, C.P., Russell, R.B.: Swaposins: circular permutations within genes encoding saposin homologues. Trends Biochem Sci 20(5), 179–180 (1995)
30. Rasmussen, K., Stoye, J., Myers, E.: Efficient $q$-gram Filters for finding all epsilon-matches over a given length. J Comput Biol 13(2), 296–308 (2006)
31. Rice, P., Longden, I., Bleasby, A.: EMBOSS: The European Molecular Biology Open Software Suite. Trends Genet 16(6), 276–277 (2000)
32. Ukkonen, E.: Approximate string-matching with $q$-grams and maximal matches. Theor Comput Sci 92(1), 191–211 (1992)
33. Wang, Z., Wu, M.: Phylogenomic reconstruction indicates mitochondrial ancestor was an energy parasite. PLoS ONE 10(9), e110685 (2014)
34. Weiner, J., Bornberg-Bauer, E.: Evolution of circular permutations in multidomain proteins. Mol Biol Evol 23(4), 734–743 (2006)