Node Similarity with q-Grams for Real-World Labeled Networks

Alessio Conte
NII, Tokyo
conte@nii.ac.jp

Gaspare Ferraro
University of Pisa
ferraro@gaspa.re

Roberto Grossi
University of Pisa
grossi@di.unipi.it

Andrea Marino
University of Pisa
marino@di.unipi.it

Kunihiko Sadakane
University of Tokyo
sada@mist.i.u-tokyo.ac.jp

Takeaki Uno
NII, Tokyo
uno@nii.jp

ABSTRACT

We study node similarity in labeled networks, using the label sequences found in paths of bounded length q leading to the nodes. (This recalls the q-grams employed in document resemblance, based on the Jaccard distance.) When applied to networks, the challenge is two-fold: the number of q-grams generated from labeled paths grows exponentially with q, and their frequency should be taken into account: this leads to a variation of the Jaccard index known as Bray-Curtis index for multisets. We describe nSimGram, a suite of fast algorithms for node similarity with q-grams, based on a novel blend of color coding, probabilistic counting, sketches, and string algorithms, where the universe of elements to sample is exponential. We provide experimental evidence that our measure is effective and our running times scale to deal with large real-world networks.

ACM Reference Format:
Alessio Conte, Gaspare Ferraro, Roberto Grossi, Andrea Marino, Kunihiko Sadakane, and Takeaki Uno. 2018. Node Similarity with q-Grams for Real-World Labeled Networks. In *Proceedings of ACM SIGKDD conference (KDD'18).* ACM, New York, NY, USA, Article 4, 9 pages.

1 INTRODUCTION

Heterogeneous networks are prevalent in the real world (e.g., social, bibliographic, biological networks), and take into account the fact that objects and relations are of various types (e.g. see [23]). Many papers for similarity in networks involve structural features but a few exploits their heterogeneous structure, such as node labeling. Node labels in this domain help filter out a lot of paths [10, Sect.4].

This paper considers the real-world networks that, in addition to their linked structure, have labels that characterize the properties of the nodes. We investigate how to apply q-grams and sketching algorithms to network analysis in this scenario, where q-grams are strings of q consecutive symbols from an alphabet Σ. Our suite of fast algorithms for node similarity with q-grams, called nSimGram (node Similarity with q-Grams), proceeds in two phases. First, at preprocessing time, it builds a data structure from the network, where the paths traversing q nodes are color-coded and their concatenated labels form q-grams. Second, for any two nodes a and b queried by the user, it computes their similarity using that data structure and these q-grams. (This easily extends to sets of nodes A and B.) As an example, the table below shows the top-15 venues similar to KDD using q-grams.[1]

Top-15 publication venues similar to KDD			
1	KDD	9	PAKDD
2	ICDM	10	CIKM
3	SDM	11	ICDE
4	Data Min. Knowl. Discov.	12	SIGMOD Conference
5	SIGKDD Explorations	13	VLDB
6	PKDD	14	WWW
7	IEEE Trans. Knowl. Data Eng.	15	VLDB J.
8	Knowl. Inf. Syst.		

This and more examples are discussed in Sections 2 and 4. We borrow ideas from document processing and sequence analysis, which make great use of q-grams for many large-scale applications in several areas, ranging from web engines to bioinformatics. Perhaps the best known is document similarity using the sketches for the Jaccard index [5]. We adapt these ideas to labeled graphs.

Multisets of q-grams. Given an undirected labeled network $G = (V, E, \ell)$, where $\ell : V \mapsto \Sigma$ is the node labeling over an alphabet Σ, we consider the q-grams related to a node. Specifically, for an integer $q > 0$ and a node $u \in V$, consider all the simple paths of length $q - 1$ landing in u. (In this way, we are looking also at the nodes in $N^{<q}(u)$, namely, those at distance less than q from u.) Each such path will give rise to a q-gram from Σ^q, i.e., a sequence of q labels $\ell(x)$ obtained by traversing the nodes x along the path, following the direction from its beginning to the destination u. We denote by $L(u)$ the *multiset* of q-grams thus obtained (see Figure 1).

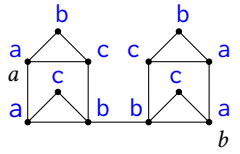

Figure 1: A graph with labels a, b, c**, and the multisets of 3-grams for nodes** a **and** b**:** $L(a)$ = [baa, bca, bca, caa, cba] **and** $L(b)$ = [baa, bba, bca, caa, cba, cba]**. Note that** $BC(a, b) = \frac{4+4}{5+6}$**.**

Similarity index. To compare any two nodes a and b, we adopt the Bray-Curtis similarity index $BC(a, b)$ that is a Jaccard related index defined on multisets $L(a)$ and $L(b)$ using the frequency of their q-grams. This index takes into account a bounded context of the neighborhoods $N^{<q}(a)$ and $N^{<q}(b)$. It is illustrated in the

Work partially supported by JST CREST, Grant Number JPMJCR1401, Japan, and partially done while AC, RG, and AM were visiting NII Tokyo under its visitor grant.

KDD'18, August 2018, London, United Kingdom, UK

[1] It is not necessary to use graphs and q-grams in this example, but it scales well to arbitrary graphs, e.g. NetInf in Section 2.

example below and formally described in Section 2. It is based on the observation that a and b are similar if they have relationships with neighbours of the same kind and also their neighbours are similar, iterating this idea q levels for both a and b.

Example. Consider the CPA graph built on the DBIS conference dataset [26], and suppose that we want to establish similarity of conferences as illustrated in the top-15 table from the previous page. Each conference is seen as a collection of papers, and each paper is seen as a set of authors. Thus CPA is a tripartite graph with node set $C \cup P \cup A$, where each conference node in C is linked to its papers in P, and each paper node in P is also linked to its authors in A.

For any two conferences a and b, we have a plethora of similarity indices based on their neighborhoods $N(x)$ and $N(y)$. The highly cited survey [21] reports many similarity indices based on $N(a)$ and $N(b)$ and their sizes. Let us use the Bray-Curtis index $BC^0(a,b) = \frac{2|N(a)\cap N(b)|}{|N(a)|+|N(b)|}$ (also called Sørensen's index) from [21, eq.5]. To extend this index to q-grams, we need first to label the nodes of the CPA graph: we label conferences in C with the same symbol c, papers in P with the same symbol p, and each author in A with its unique name or ID.

What if we just use the individual labels instead of the q-grams? We can extend the Bray-Curtis index, redefining it as $BC'(a,b) = \frac{2|L'(a)\cap L'(b)|}{|L'(a)|+|L'(b)|}$ where $L'(a)$ is the multiset of labels of the neighbors of a, i.e. $|N(a)|$ copies of symbol p (same for $L'(b)$). This is not satisfying as conferences are considered similar iff they accept roughly the same number of papers. We need to involve the nodes at distance 2 or less from of x and y, and take their multisets of labels $L''(a)$ and $L''(b)$. As the labels of authors from A appear in these multisets, in the resulting index $BC''(a,b) = \frac{2|L''(a)\cap L''(b)|}{|L''(a)|+|L''(b)|}$ conferences are similar iff they share common authors.

Yet authors publish frequently in some conferences and occasionally in some other conferences, and thus the above index is biased: all the conferences where an author publishes are equally considered, even though the author publishes once in some of them.

Let us see how q-grams can improve over this. Given $u = a, b$ and $q = 3$, each q-gram in the multiset $L(u)$ is of the form $\mathtt{w}_i\,\mathtt{p}\,\mathtt{c}$ for some author w_i. The resulting index $BC(a,b) = \frac{2|L(a)\cap L(b)|}{|L(a)|+|L(b)|}$ now weighs how *frequently* the shared authors publish in *both* conferences, so as to obtain the aforementioned top-15 table.

Novelty. Although the idea of using path labels in nSimGram has been exploited by PathSim [26] and other papers on link prediction, our similarity index has a different goal and takes advantage of the widespread success of q-grams on large-scale analysis [27].

We use Figure 1 to illustrate this observation, where nodes a and b are quite similar for our index $BC(a,b)$: even though a and b do *not* share common paths of 3 nodes, their corresponding multisets of 3-grams have a large overlap. We model in this way the fact that both a and b interact in the same way with their neighborhood.

This is in contrast with the above similarity measures relying on shared paths of given length, e.g. connecting a and b. In our example a and b can be made far apart each other (just replace the edge with both endpoints labeled b by an arbitrarily long path or large-diameter subgraph) but still remain similar, which is not necessarily true with other measures, where the number of connecting paths of unbounded length can explode. Indeed we will see that one of the features of nSimGram is its efficient performance. We refer the reader to Sections 4–5 for further discussion of the related work.

Contributions. nSimGram provides the following operational context. Given an unordered labeled graph G with n nodes and m edges, and a positive integer $q = O(\log n)$, it provides an $O(2^q m)$-time randomized algorithm, called $\mathtt{preprocess}_q(G)$, that builds a data structure on a suitable set of the q-grams from G. After that, it can answer user queries of the following form. For any two nodes a and b, and a sampling size r chosen by the user, a call to $\mathtt{query}_r(a,b)$ provably returns an *unbiased estimator* for the Bray-Curtis similarity index $BC(a,b)$ for the set of q-grams stored in the data structure, that is, a randomized method whose expected outcome is $BC(a,b)$. This takes $O(rq)$ time, using a sample of r q-grams, where $r = O(q \log n)$ in our experiments.

Note that the brute-force approach to compute the BC index does not scale, whereas nSimGram gives a guaranteed performance on real-world networks. For instance, it allows us to approximate the similarity between any two movies in the IMDB network, which has more than 280 million edges, in relatively few seconds for $q = 6$. The number of paths of length q is simply unmanageable otherwise. In summary, nSimGram gives the following contributions.

- Exploit the usage of q-grams in large-scale network analysis, and adopt a node similarity based on variations of the Jaccard index that apply also to multisets of q-grams. We show that our measure outperforms popular state of the art node similarity measures on real world datasets.
- Show how to estimate node similarity in a randomized fashion, as its exact computation is too demanding in terms of space and running time. We give theoretical bounds and experimental running times on the data sets.
- Employ several ideas to handle this situation: randomized color coding of the nodes of G, probabilistic counting and sketches, string algorithms, etc. The resulting code is easy to make parallel on multi-core machines.

Other similarity indices. Among the 30 similarity indices surveyed in [21, Sect.3], several are based on neighborhood $N(u)$ and thus can benefit of the ideas in nSimGram. There is a computational hurdle as the q-grams can exponentially explode for increasing values of q and $|\Sigma|$. Contrarily to the q-grams found in a document of n symbols, which are at most $n - q + 1$, the q-grams in a graph with n nodes can be $|\Sigma|^q \gg n$ in number. To worsen this situation, the multiset of all q-grams in G is potentially generated by n^q paths in G. This gives a twist to our problem, as we cannot afford to go through all those paths in large graphs, unless Σ, q, n are very small.

nSimGram proves to be very useful in this scenario because of its guaranteed performance. Note that it does not belong to the family of graph-based methods in supervised learning, such as [7, 18, 20], as the implicit assumption that there is no supervising data makes the similarity problem intrinsically different [22].

2 LABELED PATHS AND NODE SIMILARITY

Consider an undirected labeled graph $G = (V, E, \ell)$ over an alphabet Σ of symbols. We use $n = |V|$, $m = |E|$, and $N(u)$ as the set of neighbors of node u. Observe that $|\Sigma| \le n$ without loss of generality.

For a fixed integer $q > 0$, consider an arbitrary simple path $P = u_1, u_2, \ldots, u_q$ traversing $q - 1$ edges in G (i.e. $u_i \neq u_j$ and

$\{u_i, u_{i+1}\} \in E$ for $1 \le i < j \le q$). We call the orientation $u_1 \to u_2 \to \cdots \to u_q$ of P a *q-path* leading to u_q, and its *q-gram* $L(P) = \ell(u_1)\ell(u_2)\cdots\ell(u_q) \in \Sigma^q$ is the corresponding string obtained by concatenating the labels of its nodes.[2]

For a node $a \in V$, we define $L(a)$ as the corresponding *multiset* of q-grams for all q-paths P leading to a.[3]

$$L(a) = [x \in \Sigma^q : \exists\, q\text{-path } P \text{ leading to } a \text{ with } L(P) = x]$$

For a q-gram x, we consider the *frequency* of x within multiset $L(a)$, i.e., the number of q-paths P leading to a whose q-grams are x, and define a frequency vector ranging over all strings $x \in \Sigma^q$ (a.k.a. graph kernel)

$$f_a[x] = |\{P : P \text{ is a } q\text{-path leading to } a \text{ and } L(P) = x\}|$$

Note that each multiset $L(a)$ can be equivalently seen as the set of pairs $\{\langle x, f_a[x]\rangle : x \in \Sigma^q \text{ and } f_a[x] > 0\}$. For example, in Figure 1 $L(a)$ can be seen as $\{\langle \mathsf{baa}, 1\rangle, \langle \mathsf{bca}, 2\rangle, \langle \mathsf{caa}, 1\rangle, \langle \mathsf{cba}, 1\rangle\}$, where $f_a[\mathsf{baa}] = 1$, $f_a[\mathsf{bca}] = 2$, and so on.

Given a graph G, let $\mathcal{L} \subseteq \Sigma^q$ be the set of distinct q-grams found in the q-paths of G. Applying the Bray-Curtis similarity index, we get the following measure for any two nodes a and b,

$$BC(a, b) = \frac{2 \times \sum_{x \in \Sigma^q} \min(f_a[x], f_b[x])}{\sum_{x \in \Sigma^q} f_a[x] + f_b[x]}$$

Note that ranging x over $\mathcal{L}$, instead of Σ^q, is necessary and sufficient in the above formula for any a and b. As a side note, Bray-Curtis is a relevant index for multisets, and is also known as Steinhaus similarity, Pielou's Similarity, Sørensen's quantitative, and Czekanowski's similarity [16]. The numeral ecology book [16, p.265] reports also some historical notes.

The BC index applied to NetInf is an interesting study case. It is a graph representing the flow of information on the web among blogs and news websites (see Section 4 for more details). The edges represent the "who copies who" relationships. We label the nodes of NetInf using labels a, b, c, d: a node with label a is ranked among the top 4% by Amazon's Alexa ranking [2]; a node with label b is in the following 15%, with label c is in the following 30%, and the rest with label d. Surprisingly, this simple labeling gives a high-quality similarity index, as it can be inspected in the table below.

Web Site	Web Site	BC index
nytimes.com	huffpost.com	0.760524
nytimes.com	washingtonpost.com	0.732766
nytimes.com	sportingnews.com	0.330400
nytimes.com	rollingstone.com	0.056660

NetInf, similarity between pairs of news web sites

3 EFFICIENT COMPUTATION OF NODE SIMILARITY

For any two nodes $a, b \in V$, the similarity indices described in Section 2 require the computation of the frequency vectors $f_a[\,]$ and $f_b[\,]$, which is one of the main hurdles to overcome as *(i)* the size of each vector is potentially $|\Sigma^q|$ and *(ii)* its definition requires to explore n^q q-paths. In this section we propose a random estimator based on color coding, sketching, and string algorithms, which can be computed efficiently and is unbiased, that is, its expected value is the actual similarity index.

We proceed in steps. First, we address issue *(ii)* and use color coding to reduce the number of potentially explored q-paths from n^q to $2^{O(q)}n$, making it thus feasible for large n and $q = O(\log n)$. Then, we address issue *(i)* by computing sketches of $f_a[\,]$ and $f_b[\,]$, with the constraint that we cannot explicitly go through all the strings in the universe Σ^q. The size of the sketches is small compared to $|\Sigma|^q$, which is a significant benefit when Σ or q are large.

Algorithm 1: $\texttt{preprocess}_q(G)$:

Input : $G = (V, E)$ undirected graph and an integer $q > 0$ for color coding.
Output: M = dynamic programming table for color coding.

```
parallel foreach u ∈ V do M_{1,u} = ⟨χ(u), 1⟩
for i ∈ {2, 3, ..., q} do
    parallel foreach u ∈ V do
        foreach v ∈ N(u) do
            foreach ⟨C, f⟩ ∈ M_{i−1,v} such that χ(u) ∉ C do
                f' ← M_{i,u}(C ∪ {χ(u)})
                M_{i,u} ← ⟨C ∪ {χ(u)}, f' + f⟩
return M
```

3.1 $\texttt{preprocess}_q(G)$: color coding of the q-paths

We here describe the preprocessing of the input graph G. Following color coding [4], we restrict our attention to $q = O(\log n)$ and assign a random coloring $\chi : V \to [q]$ to the nodes of G, where $[q] \equiv [1, \ldots, q]$. We remark that, in addition to its input label $\ell(u)$, each node $u \in V$ now has color $\chi(u)$ independently and uniformly chosen from $[q]$. We say that a q-path $u_1, u_2, \ldots, u_q$ is *colorful* iff $\chi(u_i) \neq \chi(u_j)$ for $1 \le i < j \le q$ (hence all the q different colors appear in the q-path). As a colorful q-path can use $q!$ colorings of its nodes out of q^q possible ones, the probability that a q-path is colorful is $q!/q^q \ge e^{-q}$.

We use color coding for two reasons: (1) to guarantee that we choose simple paths, rather than walks, and (2) to reduce the number of q-paths by roughly a factor of $q!/q^q \ge 1/e^q$ to colorful q-paths.

Algorithm 1 reports the dynamic programming approach in [4]. We create a new node s that is connected to all the other nodes. With a little abuse of notation, denote by G the resulting graph, and let n be the number of nodes and m the number of edges. To list all colorful q-paths in the original graph, we can equivalently list all the colorful $(q+1)$-paths in G starting from s. In order to alleviate the notation, we can equivalently discuss how to list the colorful q-paths starting from s (just rename $q+1$ as q).

Algorithm 1 returns a table M where $M_{i,j}$ stores the collection of pairs $\langle C, f\rangle$ where $C \subseteq [q]$ is a color set such that $|C| = i$ and there are f colorful i-paths from s to node j, with each i-path using all colors in C. As $q = O(\log n)$, we store each set C as a bit vector of length q in which there are i 1s, which fits a machine word and can be interpreted as an integer of size polynomial in n. Using bit

[2] Even if the graph is undirected, traversing P in one direction can be seen as an oriented path of q nodes, the q-path. Each path gives rise to two q-paths, one for each traversal direction. In this way, the cardinality of the set of q-paths in G equals the cardinality of the multiset of q-grams in G.
[3] Overloading the usage of $L()$ is not a problem as it is clear from the context.

Algorithm 2: query: COLORFUL-SAMPLER

Input : $X = \{a, b\}$ a pair of nodes from graph G; M = color coding table for G; r = number of colorful paths to sample.
Output: W = random sample set of colorful grams $x \in L(X)$ with probability $p_X(x)$.

$R \leftarrow []$
parallel for $j \in [r]$ **do**
  $u \leftarrow$ randomly chosen $v \in X$ with probability $p_v = \frac{M_{q,v}([q])}{\sum_{z \in X} M_{q,z}([q])}$
  $P \leftarrow$ RANDOM-PATH-TO(u)
  Add P to R
return $W = \{L(P) : P \in R\}$
Function *RANDOM-PATH-TO(u)*
  $P \leftarrow \langle u \rangle \quad D \leftarrow [q] \setminus \{\chi(u)\}$
  for $i \in \{q-1, \ldots, 1\}$ **do**
    $u \leftarrow$ randomly chosen $v \in N(u)$ with probability $p_v = \frac{M_{i,v}(D)}{\sum_{z \in N(u)} M_{i,z}(D)}$
    $P \leftarrow u \cdot P \quad D \leftarrow D \setminus \{\chi(u)\}$
  return P

manipulations, we can implement in $O(1)$ time the update of any entry in M. Note that $M_{i,j}$ contains at most $\binom{q}{i}$ sets, each with i colors. Hence computing row i from row $i-1$ requires $O(m\binom{q}{i-1})$ time (as we scan all the adjacency lists). The entire computation requires thus $O(m \sum_{i=1}^{q} \binom{q}{i-1}) = O(m\, 2^q)$ time.

LEMMA 3.1. *Given an undirected graph G of n nodes and m edges, such that G has a random coloring in $[q]$, where $q = O(\log n)$, Algorithm 1 (*preprocess$_q(G)$*) returns the dynamic programming table M of color coding in $O(m\, 2^q)$ time and space.*

REMARK 1. From now on, *colorful path* is an equivalent term for colorful q-path, and a colorful q-gram is the q-gram corresponding to a colorful path. To avoid cumbersome notation, we reuse notation for L, f_a, $\mathcal{L}$, etc. for colorful paths as the domain is clear at this point.

As $q!/q^q \geq 1/e^q$, color coding asks to repeat the random coloring of nodes for $e^q t$ times, so that success probability becomes $\geq 1 - e^{-t}$. However this is just to establish if a q-path exists (when $q = n$, it is the famous NP-complete Hamiltonian path problem [9]). As it is unreasonable to generate all possible q^n colorings, we use the idea of balanced hashing [3], which requires just $O(e^{q+O(\log^3 q)} \log n)$ colorings, so that each subset of q nodes from V is colorful roughly the same number of times, apart from a multiplicative constant. On the other end, counting exactly q-paths is #W[1]-complete, thus difficult to parameterize [8]. In practice, choosing a single random coloring is working pretty well on real-world networks.

3.2 query$_r(a, b)$: sampling and sketching colorful paths

We have reduced the number of q-paths to examine using Algorithm 1, and it is not difficult to modify it to list also the colorful q-grams, printing $L(P)$ for each colorful path P. This provides an inefficient implementation of query$_r(a, b)$ for two nodes a and b. Indeed we still have an issue as it could be $\mathcal{L} \approx \Sigma^q$.

Our key idea is to single out a sample of r colorful q-grams from $\mathcal{L}$, *without* actually exploring all colorful paths (which is expensive). Sketching is performed usually on an explicit set, whereas we have to sketch an implicit set $\mathcal{L}$ and the colorful paths. If we achieve this goal, our sampling cost can be made proportional to a user-selectable parameter $r < |\mathcal{L}| \leq |\Sigma|^q \leq n^q$.

Algorithm NSIMGRAM-COUNT to answer query$_r(a, b)$ works as follows.

(1) Compute a suitable sample $W \subseteq \mathcal{L}$ such $\tau = |W|$ is at most r, using Algorithm 2.
(2) Compute $f_a[x]$ and $f_b[x]$ for each $x \in W$, using Algorithm 3.
(3) Approximate $BC(a, b)$ by returning the value of $BC_W(a, b)$ shown next.

$$BC_W(a, b) = \frac{2}{|W|} \times \sum_{x \in W} \frac{\min(f_a[x], f_b[x])}{f_a[x] + f_b[x]}.$$

Phase 1. Algorithm 2 can sample from the multiset $L(X)$ of colorful paths for a pair of nodes $X = \{a, b\}$. Here $x \in L(X)$ is sampled with probability $p_X(x) = \frac{f_a[x]+f_b[x]}{\sum_{y \in \mathcal{L}} f_a[y]+f_b[y]}$.

Using table M computed by Algorithm 1, a suitable set W is sampled from the colorful q-grams in $\mathcal{L} \subseteq \Sigma^q$. In particular the sample depends on the frequencies of the q-grams ending in a and b, as in the case of consistent weighted sampling, where more frequent q-grams need to be sampled more often. One additional challenge in our case is that we do know a priori the frequency of q-grams before sampling.

Phase 2. We generate only the q-grams $x \in W$ that originate from colorful paths ending in a or b. In the case of a (the same is done for b), we proceed as described in Algorithm 3 for steps $i = 1, 2, \ldots, q$, by expanding in BFS order only the i-paths ending in a and having i-grams that are suffixes of W (this computation can be made more space efficient by using tries). We maintain a multiset T of these i-grams, each represented by a triple $\langle z, x, C \rangle$ to indicate that there is an i-path starting from z and ending in a, whose i-gram is x and colorset is C (note that $\langle z, x, C \rangle$ can appear more than once in T as there might be more i-paths from z to u labeled with the same i-gram x).

Phase 3. We will show that we obtain an unbiased for the approximation of $BC(a, b)$ in Section 3.3.

LEMMA 3.2. *For any two nodes $a, b \subseteq V$, the running time of NSIMGRAM-COUNT is $O(rq)$ time and space where $r < |\mathcal{L}| \leq |\Sigma|^q$ and $q = O(\log n)$.*

3.3 Estimating Bray-Curtis Index

We show that $BC_W(a, b)$ is an unbiased estimator for $BC(a, b)$, namely, $BC(a, b) = \mathbb{E}[BC_W(a, b)]$ where W is the sample and $\tau = |W| \leq r$ in Algorithm 2 (COLORFUL-SAMPLER). As we are dealing with multisets, W depends on a and b, as is in consistent weighted sampling [12, 30]: intuitively we need to sample more frequently the q-grams that are more frequent in $L(a)$ and $L(b)$ (and we cannot resort to their frequency in $L(V)$ as it can be totally different). This is what is done in Algorithm 2.

THEOREM 3.3. $BC(a, b) = \mathbb{E}[BC_W(a, b)]$.

Algorithm 3: nSimGram-count, exactly counting frequencies of sampled q-grams

Input : a = a node from graph G; W = sample of its colorful q-grams.
Output: $f_a[x]$ = frequency of each $x \in W$.

1 $T \leftarrow [\,]$ // step $i = 1$
2 $T \leftarrow T \cup [\langle a, \ell(a), \{\chi(a)\}\rangle]$
3 **for** $i \in \{2, 3, \ldots, q\}$ **do**
4 $T' \leftarrow [\,]$
5 **parallel foreach** $\langle z, x, C\rangle \in T$ **do**
6 **foreach** $v \in N(z)$ *such that* $\chi(v) \notin C$ **do**
7 **if** $\ell(v) \cdot x$ *is a suffix of a* q*-gram in* W **then**
8 $T' \leftarrow T' \cup [\langle v, \ell(v) \cdot x, C \cup \{\chi(v)\}\rangle]$ //critical s.
9 $T \leftarrow T'$
10 $f_a \leftarrow (0, \ldots, 0)$
11 **foreach** $\langle z, x, C\rangle \in T$ **do** $f_a[x] \leftarrow f_a[x] + 1$
12 **return** f_a

Without loss of generality, we will deal with the case in which $\tau = |W| = r = 1$, meaning that just one label $l \in \mathcal{L}$ is considered to build the sketches of f_a and f_b. Our estimator can be seen as the average result of $r > 1$ experiments (as in the case of min-r sketches). Hence, to prove Theorem 3.3 it suffices to prove that:

$$\sum_{l \in \mathcal{L}} Pr[W = \{l\}] \cdot \frac{2 \times \min(f_a[l], f_b[l])}{f_a[l] + f_b[l]} = BC(a, b), \qquad (1)$$

where $Pr[W = \{l\}]$ is the probability of choosing l in Algorithm 2. As we can see, to get an unbiased estimator we must prove that Algorithm 2 provides a sample $W = \{l\}$ such that $Pr[W = \{l\}] = \frac{f_a[l]+f_b[l]}{\sum_{j\in\mathcal{L}} f_a[j]+f_b[j]}$, as by plugging this in Equation 1 we get Theorem 3.3.

In the following we show that Algorithm 2 chooses q-grams according to this desired probability, as stated by the following lemma.

Lemma 3.4. *Given a and b, Algorithm 2 selects a q-gram l with probability $\frac{f_a[l]+f_b[l]}{\sum_{j\in\mathcal{L}} f_a[j]+f_b[j]}$.*

For a node $z \in \{a, b\}$ denote as E_z the event that a colorful path ending in z is sampled and with E_z^l the event that a colorful path with q-gram l and ending in z is sampled. Moreover, denote as $\mathcal{P}_z$ and $\mathcal{P}_z^l$ respectively the number of colorful paths ending in z and the only ones among these having q-gram l. First of all notice that $\sum_{j\in\mathcal{L}} f_a[j] = \mathcal{P}_a$ and $\sum_{j\in\mathcal{L}} f_b[j] = \mathcal{P}_b$,[4] as they are respectively the total number of colorful paths ending in a and b. In order to prove Lemma 3.4, we use the following result.

Lemma 3.5. *Procedure random-path-to(u) in Algorithm 2 selects a colorful path ending in u with probability $1/\mathcal{P}_u$.*

Proof of Lemma 3.5. Let $P = v_{q-1}, \ldots, v_1, x$ be the path sampled by the procedure and let $\mathcal{P}_{\langle v_i, \ldots, v_1, x\rangle}$ (with $i \leq q-1$) be the number of colorful paths having $\langle v_i, \ldots, v_1, x\rangle$ as a suffix. According to the choices done in the algorithm: the probability Pr_1 that v_1 is selected among the compatible neighbors of x is $\mathcal{P}_{\langle v_1, x\rangle}/\mathcal{P}_x$, the probability Pr_2 that v_2 is selected given v_1 is $\mathcal{P}_{\langle v_2, v_1, x\rangle}/\mathcal{P}_{\langle v_1, x\rangle}$, and so on. As the probability of getting P is $Pr_1 \cdot Pr_2 \cdot \ldots \cdot Pr_{q-2}$, we obtain $1/\mathcal{P}_x$. □

Proof of Lemma 3.4. We have:

$$Pr[W = \{l\}] = Pr[E_a^l | E_a] \cdot Pr[E_a] + Pr[E_b^l | E_b] \cdot Pr[E_b], \qquad (2)$$

where the first part is the probability that l has been sampled through a colorful path ending in a and, analogously, the second part is the probability that l has been sampled for b.

By Lemma 3.5 we also obtain:

$$Pr[E_a^l | E_a] = \frac{\mathcal{P}_a^l}{\mathcal{P}_a}, \qquad Pr[E_b^l | E_b] = \frac{\mathcal{P}_b^l}{\mathcal{P}_b}. \qquad (3)$$

Indeed, once the ending point is fixed as a or b, the probability of choosing a colorful path is uniform at random, i.e. meaning that they are respectively $\frac{1}{\mathcal{P}_a}$ and $\frac{1}{\mathcal{P}_b}$. This implies that the probability of extracting a q-gram l is proportional to the number of paths having that q-gram, as stated in Equation 3. As the probability of choosing $v \in \{a, b\}$ is $\mathcal{P}_v/(\mathcal{P}_a + \mathcal{P}_b)$ at line 3, by Equations 2 and 3 we obtain:

$$Pr[W = \{l\}] = \frac{\mathcal{P}_a^l + \mathcal{P}_b^l}{\mathcal{P}_a + \mathcal{P}_b} = \frac{f_a[l] + f_b[l]}{\sum_{j\in\mathcal{L}} f_a[j] + f_b[j]}.$$

□

In the following we relate the number of experiments, i.e. r, to an absolute error for the estimate of BC index. As our values are between 0 and 1, and our estimates are unbiased, we can use the classical Hoeffding bound [6] to upper bound the absolute error ϵ with high probability. In particular by applying this bound, we obtain that

Lemma 3.6. *The absolute error is bounded by ϵ w.h.p. by setting $r = \Omega\left(\frac{\log \mathcal{P}_a + \mathcal{P}_b}{\epsilon^2}\right)$*

Note that the latter value can be ensured choosing $r = \Omega(q\epsilon^{-2} \log n)$ and depends logarithmically on the number of colorful paths leading to a and b. In our experiments, we estimate $\mathcal{P}_a + \mathcal{P}_b$ by looking at our color coding dynamic programming table M computed by Algorithm 1 ($\texttt{preprocess}_q(G)$).

3.4 Estimating Frequencies of q-grams

In some instances Algorithm 3 can explore many colorful paths. To alleviate this issue, we show in Section 3.4 that we can avoid to execute Algorithm 3.

In this section we show how to improve Phase 2 of nSimGram-count to avoid to explicitly generate all the q-grams $x \in W$ as in Algorithm 3 to get their frequencies. In particular, once W has been chosen using our Phase 1, in order to approximate $BC_W(a, b)$ we show how to approximate $f_a[x]$ and $f_b[x]$. However, it is not enough to replace $f_a[x]$, $f_b[x]$ in $BC_W(a, b)$ with their approximation, as the ratio of approximated quantities could easily lead to a biased estimator.

To overcome this obstacle, we rewrite our estimators $BC_W(a, b)$ in a different equivalent way and reuse our suitable sample W to approximate them.

[4] Recall that $\mathcal{P}_a = M_{q,a}([q])$ and $\mathcal{P}_b = M_{q,b}([q])$.

Algorithm nSimGram-count to answer $\texttt{query}_r(a, b)$ works as follows.

(1+2) Compute $f'_a[x]$ and $f'_b[x]$ using Algorithm 3 where the last line $W = \{L(P) : P \in R\}$ is replaced by the code
$f'_a = (0, \ldots, 0)$; **foreach** $P \in R$ ending in a **do** $f'_a[L(P)] = f'_a[L(P)] + 1$
$f'_b = (0, \ldots, 0)$; **foreach** $P \in R$ ending in b **do** $f'_b[L(P)] = f'_b[L(P)] + 1$

(3) Approximate $BC(a, b)$ by returning the value of $BC_W(a, b)$ shown next.

$$BC_W(a,b) = 2 \sum_{l \in W} \min\left(\frac{f'_a[l]}{r}, \frac{f'_b[l]}{r}\right)$$

Recall that we denote as $\mathcal{P}_z$ the number of paths ending in a node $z \in V$, and as $\mathcal{P}_z^l$ the ones with q-gram l. The BC_W approximation can be rewritten as follows.

$$BC_W(a,b) = 2 \sum_{l \in W} \min\left(\frac{\mathcal{P}_a^l}{\mathcal{P}_a + \mathcal{P}_b}, \frac{\mathcal{P}_b^l}{\mathcal{P}_a + \mathcal{P}_b}\right)$$

Now, consider R as the set of paths sampled by Algorithm 2 and let Q_z be the number of the paths in R and ending in $z \in \{a, b\}$ and Q_z^l the number of the ones having label l. We have that the following Equation can be used to approximate $BC_W(a, b)$ thanks to Theorem 3.7.

$$\sum_{l \in W} \min\left(\frac{Q_a^l}{r}, \frac{Q_b^l}{r}\right). \tag{4}$$

Theorem 3.7. *Given the set R of r paths sampled by Algorithm 2 and given $z \in \{a, b\}$, $\frac{Q_z^l}{r}$ is an unbiased estimator for $\frac{\mathcal{P}_z^l}{\mathcal{P}_a + \mathcal{P}_b}$.*

Proof. By using Lemma 3.4 and Lemma 3.5, we have that Algorithm 2 samples a path ending in a or b uniformly at random. As the sample set R of r paths is chosen uniformly at random, $\frac{Q_a^l}{r}$ can be seen as

$$\begin{aligned} Pr[P \text{ ends in } a, L(P) = l, P \in R | P \text{ ends in } a \text{ or } b, P \in R] = \\ \frac{Pr[P \text{ ends in } a, L(P) = l, P \in R]}{Pr[P \text{ ends in } a \text{ or } b, P \in R]} = \\ \frac{Pr[P \text{ ends in } a, L(P) = l] \cdot Pr[P \in R]}{Pr[P \text{ ends in } a \text{ or } b] \cdot Pr[P \in R]} = \\ Pr[P \text{ ends in } a, L(P) = l \mid P \text{ ends in } a \text{ or } b] = \frac{\mathcal{P}_a^l}{\mathcal{P}_a + \mathcal{P}_b} \end{aligned}$$

This corresponds to the fraction of paths ending in a having q-gram l with respect to the size of the sampling space. □

4 EXPERIMENTS

We describe the experimental evaluation for nSimGram, our suite of algorithms. The computing platform is a machine with Intel(R) Xeon(R) CPU E5-2620 v3 at 2.40GHz, 24 virtual cores, 128 Gb RAM, running Ubuntu Linux version 4.4.0-22-generic. Code written in C++ and compiled with g++ version 5.4.1 with OpenMP. As described in Section 3, nSimGram preprocesses the input graph using Algorithm 1, and performs the queries using nSimGram-simple and nSimGram-count. The source code is open and available at github.com/GaspareG/SubgraphSimilarity.

4.1 Data sets

DBIS. In order to compare our measure with popular state of the art measures, we use the same dataset as [26]. This dataset contains 464 venues and top-5000 authors from the database and information system area and has been chosen in [26] instead of the full DBLP dataset to alleviate the high computational costs of P-PageRank and SimRank. Authors are linked to papers by authorship and papers are linked to the venues in which they appeared. *Considered query:* compute the top-k similar venues to a given one.

NetInf. This graph was computed by the *NetInf* approach, as part of the *SNAP* project [25], by tracking cascades of information diffusion to reconstruct "who copies who" relationships. Each node represents a blog or news website, and a website is connected to those who frequently copy their content. The graph contains 854 nodes and 3824 edges. We labelled websites according to their importance, using Amazon's Alexa ranking [2]: the labels correspond to respectively the websites ranked in the top 4%, the following 15%, the following 30%, and the remaining 51% (i.e. $|\Sigma| = 4$). *Considered query:* compute the similarity of between two websites or two sets of websites.

IMDb. In this graph, taken from the *Internet Movie Database* [11], nodes correspond to movies, and there is a link between two movies if their casts share at least one actor. The graph contains 1 060 209 movies (nodes) and 288 008 472 edges. Each movie is labeled with one of $|\Sigma| = 36$ genres. *Considered query:* similarity between movies.

4.2 Comparison with Other Measures

Our algorithms in nSimGram are unsupervised, so we do not consider graph-based methods that make extensive use of supervised learning, like [7, 18, 20].[5] We compare our method with respect to the ones in [26], comparing P-PageRank [14], SimRank [13], rw (random walk), prw (pairwise random walk), and PathSim [26] for the DBIS dataset.

Table 1 reports the competing methods when finding the top-10 venues similar to PKDD, where data in columns are taken from [26] except the last one. Our ranking has been computed with nSimGram-simple, with $q = 3$ and $r = 2000$.[6] When setting nSimGram-simple, conferences have the same category-label, papers also, and authors have their own labels. With respect to the first columns of Table 1, our approach prefers venues that are more similar to PKDD both for topics and popularity. With respect to PathSim, it prefers KDD to SDM as there is a higher correlation among authors publishing in PKDD and KDD with respect to PKDD and SDM, and our sampling procedure is guided by these frequencies. For the same reason, our ranking prefers DaWak to KDID: the former is a second tier conference (active since 1999) on the same topic while the latter is not a well-known venue (active from 2002 to 2006) as pointed out in [26].

[5] The availability of data for supervising makes the problem intrinsically different [22]. Moreover, the setup phase of such approaches limits their scalability as it may require hours on graphs with just thousands of nodes.

[6] This is relatively small wrt the number of q-paths in the graph.

rank	P-PageRank	SimRank	rw	prw	PathSim	ours
1	PKDD	PKDD	PKDD	PKDD	PKDD	PKDD
2	KDD	Local Pattern Detection	KDD	Local Pattern Detection	ICDM	ICDM
3	ICDE	KDID	ICDM	DB Support for DM Appl.	SDM	KDD
4	VLDB	KDD	PAKDD	Constr.-Bsd. Min. & Induc. DB	PAKDD	PAKDD
5	SIGMOD	Large-Scale Paral. Data Min.	SDM	KDID	KDD	SDM
6	ICDM	SDM	TKDE	MCD	Data Min. Knowl. Disc.	Data Min. Knowl. Discov.
7	TKDE	ICDM	SIGKDD Expl.	Pattern Detection and Discovery	SIGKDD Expl.	SIGKDD Expl.
8	PAKDD	SIGKDD Expl.	ICDE	RSKD	Knowl. Inf. Syst.	Knowl. Inf. Syst.
9	SIGIR	Constr.-Bsd. Min.	SEBD	WImBI	J. Intell. Inf. Syst.	J. Intell. Inf. Syst.
10	CIKM	Induc. DB TKDD	CIKM	Large-Scale Paral. Data Min.	KDID	DaWaK

Table 1: The top-10 venues similar to PKDD according to the measures in [26] and our measure.

Method	Accuracy (nDCG)
P-PageRank	0.5552
SimRank	0.6289
RW	0.7061
PRW	0.5284
PathSim	0.7446
Our	**0.9128**

Table 2: Average ranking accuracy for a sample of venues in the DBIS dataset [26].

As in [26], we also considered the top-15 results for 15 queries from venue type (SIGMOD, VLDB, ICDE, PODS, EDBT, DASFAA, KDD, ICDM, PKDD, SDM, PAKDD, WWW, SIGIR, TREC and AP-Web) in the DBIS dataset, and we have labeled each result object with relevance score as three levels: 0–non-relevant, 1–some-relevant, and 2–very-relevant. To perform this assignment we have used the manual assignment done by experts, available at the url webdocs.cs.ualberta.ca/~zaiane/htmldocs/ConfRanking.html.

Table 2 reports the evaluation of the quality of the output ranking using the measure Normalized Discounted Cumulative Gain (nDCG, with value between 0 and 1, the higher the better) and shows that our method significantly outperforms the competitors.

4.3 Results

In order to validate the efficiency of our approach, we compare nSimGram-count and nSimGram-simple against the baseline algorithm base, that finds random paths using simple random walks and compute the BC index using the q-grams from these paths. We use the exact value computed by brute force as a reference (on small graphs). We evaluate the approximation of these approaches and their computational costs.

4.3.1 Approximation. By using the brute force approach in the NetInf dataset, we have computed the exact value of $BC(a, b)$ for randomly chosen nodes a and b and different values of q. We have then fixed the relative error ϵ and, for each approach, we have computed the size of the sample r and the time T needed to get a relative error ϵ. We have repeated the experiment 100 times and reported the results in Table 3. For every q and ϵ, nSimGram-count clearly has the best precision: it needs always a small sample r to get a relative error ϵ and has a small variance. This is at the cost of the highest running time, as once the q-grams have been sampled, their frequencies must be computed exactly. base performs surprisingly fast but its precision gets much worst for increasing q: since base does not sample paths carefully, it needs to sample many more paths to improve its accuracy obtaining also the worst time. This is not the case of nSimGram-simple, which spends more time to choose the paths to sample, but provides a guaranteed error on top of a good practical performance.

		nSimGram-count			nSimGram-simple			base		
q	ϵ	r	T	VAR	r	T	VAR	r	T	VAR
3	0.20	10	2	0.0011	200	1	0.0014	200	1	0.0010
3	0.10	50	2	0.0017	400	2	0.0029	500	1	0.0030
3	0.05	80	3	0.0003	500	2	0.0002	600	1	0.0004
4	0.20	10	1	0.0079	500	1	0.0054	1 000	1	0.0239
4	0.10	20	4	0.0037	1 000	3	0.0039	2 000	2	0.0264
4	0.05	100	5	0.0006	2 000	4	0.0020	8 000	3	0.0157
5	0.20	15	7	0.0073	3 000	9	0.0012	10 000	36	0.0327
5	0.10	30	12	0.0052	4 000	12	0.0087	30 000	83	0.0437
5	0.05	100	35	0.0010	8 000	20	0.0055	80 000	287	0.0187

Table 3: Size of the sample r and time T needed to get a relative error ϵ, for several values of q in the NetInf dataset (average over 100 experiments). VAR is the variance.

4.3.2 Preprocessing. In Table 4, we show the preprocessing time and space needed by NetInf and IMDb datasets for increasing q, showing that we can deal with relatively long q-paths even in a network with millions of nodes. We remark that the preprocessing time needed by our approach in these networks is negligible with respect to the time needed by other methods. Moreover, in Figure 2 we show the behaviour of a parallel implementation of our preprocessing phase when increasing the number of cores and dealing with the IMDb dataset setting $q = 3$ and $q = 4$. The plot shows that our preprocessing scales quite well and the benefit of parallelization is steady for both $q = 3$ and $q = 4$.

Dataset	q	Time	Space
NetInf	3	0.39s	11.20MiB
NetInf	4	0.81s	22.63MiB
NetInf	5	1.66s	45.21MiB
NetInf	6	3.47s	90.93MiB
IMDb	3	48.22s	17.94MiB
IMDb	4	105.94s	34.91MiB
IMDb	5	241.22s	69.01MiB
IMDb	6	557.48s	137.26MiB

Table 4: Preprocessing Time and Space of nSimGram-count and nSimGram-simple varying q

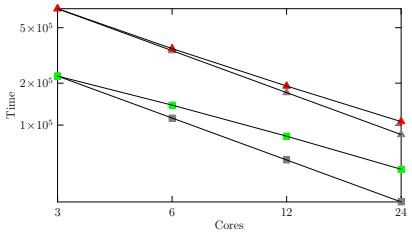

Figure 2: Scalability of the preprocessing phase varying the number of cores in the IMDb dataset with respect to the ideal scaling (gray line). $q = 4$ red, $q = 3$ green.

4.3.3 Query. We study the time performance in our experiments for several values of q and r on the NetInf and IMDb dataset.

For the NetInf dataset, in Figure 3(a) and (b), we fix $q = 7, 8$ and show the time needed by our methods for increasing r. base and nSimGram-simple have a much lower running time, which increases following the same trend. In Figure 3(c) and (d), we fix $r = 500, 1000$ and show the running times for increasing q. In this case, the running time of the exact approach rapidly increases and becomes unfeasible. Also nSimGram-count is costly when q grows or the network is big, as it has to compute the exact frequency of the sampled q-grams. base is very fast but as shown in Table 3 it is not precise. nSimGram-simple provides the best tradeoff as it can deal with bigger values of q while ensuring guarantees.

In Table 5, we report similar results for the IMDb dataset, where both the exact approach and nSimGram-count do not terminate within reasonable time due to the size of the network (1 hour for the exact and 15 minutes for nSimGram-count), while nSimGram-simple and base can deal with relatively large values of q.

		Time (ms)			
Dataset	q	Exact	nSimGram-count	nSimGram-simple	base
IMDb	3	-	846	32	3
	4	-	-	110	5
	5	-	-	295	6
	6	-	-	552	8

Table 5: Time needed by the query to measure the BC index between two nodes a and b randomly chosen and setting the size of the sample $r = 100$. Average over 100 experiments.

5 RELATED WORK

A plethora of specific similarity measures has been designed over the years and each of them makes sense depending on the application. An intrinsic issue of any measure aimed at quantifying structural properties of nodes is that there are always alternatives to describe them, thus leading inevitably to a multitude of different measures [21]. Structural measures can be roughly divided in the following categories.

- Local measures: this kind of measures often count the common neighbors of the queried nodes and they differ for the way they normalize this quantity, like [17].
- Global measures: this is the case of well-known measures like Katz index [15], random-walk with restart [28], SimRank [13], personalized PageRank [14]. Some of these are costly to compute and for this reason variations have been considered [22].
- Quasi-local: these measures are a good tradeoff of accuracy and computational complexity [21], like local random walk and superposed random walk [19].

These measures do not directly extend in the case of labeled networks. Some work to address this task has been done in the case of heterogeneous or typed networks, where social networks are seen as a collection of entities (as schools, locations, users etc.) that can be annotated with the respective category [7, 20, 26]. In this area of research, there are measures for which the similarity can be expressed through a closed formula like [26] or the result of an iterative process like [13], or the ones which are a result of a supervised learning process [7, 20, 26].

The latter works are mainly focused on predicting links and they do supervised learning aimed to do this task. Our work is unsupervised, so we have not considered graph-based methods that make extensive use of supervised learning, as in [22]. In the former class of measures, there are measures which also uses labeled paths to compare nodes. Among these, the most closely related measure to our similarity measure is PathSim [26] and its variations [23, 31]. However, they just consider the paths connecting the queried nodes x and y which are of a given type and fixed length q. As a result, they neglect possibly similar typed paths starting from x and y. Moreover, when applied to general graphs (not necessarily multipartite), this allows to measure just the similarity of nodes connected and within distance q (using the prescribed paths). This is even more restrictive in a scenario where there are many different types, like real social networks.

Some variations of PathSim make use of triangle inequality property to speed up the computation applying local sensitive hashing, while maintaining similar quality performance [31]. There are extensions of PathSim [24] that allows to measure the similarity between nodes of different types (see [23] for a survey), which is not the problem considered here. When the labeled graph is (multi)partite also random-walk with restart [28], SimRank [13], and personalized PageRank [14] can be applied. Other works do similarity analysis to study social influence in heterogenous networks, but they use more information from social influence side which is available depending on the application [29].

We remark that counting cycles and paths of length q, parameterized by q, is #W[1]-complete [8]. Interestingly, in practice, a recent work [1] allows to count graphlets of fixed size q in large networks, but we observe that $q \leq 4$. We argue the need of approximation algorithms to deal with bigger q and larger networks, focusing on frequent q-grams.

6 CONCLUSIONS AND FUTURE WORK

We presented randomized algorithms and data structures for sketching node similarity. Samples are relatively small (near logarithmic) with respect to the size of the universe, and exploit the distributions of the q-grams involved. We have shown that our similarity measure

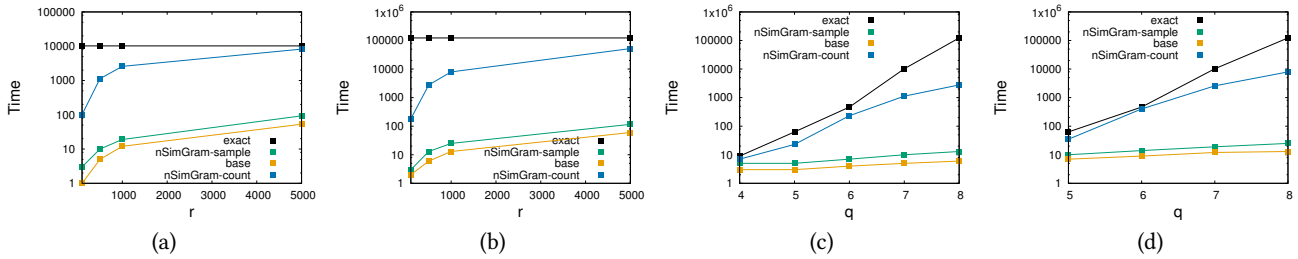

Figure 3: Time needed by the exact approach, nSimGram-count, nSimGram-simple, base varying the size of the sample r, in (a) setting $q = 7$ and in (b) setting $q = 8$, or varying q, in (c) setting $r = 500$ and in (d) setting $r = 1000$ (average over 100 experiments).

significantly outperforms popular state-of-the-art works for heterogeneous networks is able to spot ground-truth similarities, while showing good practical performance. Moreover, the algorithms we propose, nSimGram-simple and nSimGram-count, guarantee a good approximation (as unbiased estimators) compared to a less refined baseline sampler. The steady running time of nSimGram-simple on networks with hundreds of millions of edges suggests its usefulness as an estimator on very large networks.

The assumptions that the graph is undirected and with one label per node can be easily removed from our model: it would be interesting to further study similarity indices relevant in real-world scenarios that can be sketched with our algorithms.

REFERENCES

[1] Nesreen K Ahmed, Jennifer Neville, Ryan A Rossi, and Nick Duffield. 2015. Efficient graphlet counting for large networks. In *Data Mining (ICDM), 2015 IEEE International Conference on*. IEEE, 1–10.

[2] Alexa. Accessed October 2017. Website Traffic, Statistics and Analytics. https://www.alexa.com/siteinfo/. (Accessed October 2017).

[3] Noga Alon and Shai Gutner. 2010. Balanced Families of Perfect Hash Functions and Their Applications. *ACM Trans. Algorithms* 6, 3, Article 54 (July 2010), 12 pages. https://doi.org/10.1145/1798596.1798607

[4] Noga Alon, Raphael Yuster, and Uri Zwick. 1995. Color-coding. *Journal of the ACM (JACM)* 42, 4 (1995), 844–856.

[5] Andrei Z. Broder. 2000. Identifying and Filtering Near-Duplicate Documents. In *Combinatorial Pattern Matching, 11th Annual Symposium, CPM 2000, Montreal, Canada, June 21-23, 2000, Proceedings*. 1–10. https://doi.org/10.1007/3-540-45123-4_1

[6] David Eppstein and Joseph Wang. 2001. Fast approximation of centrality. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 228–229.

[7] Yuan Fang, Wenqing Lin, Vincent W Zheng, Min Wu, Kevin Chen-Chuan Chang, and Xiao-Li Li. 2016. Semantic proximity search on graphs with metagraph-based learning. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*. IEEE, 277–288.

[8] Jörg Flum and Martin Grohe. 2004. The Parameterized Complexity of Counting Problems. *SIAM J. Comput.* 33, 4 (Aug. 2004), 892–922. https://doi.org/10.1137/S0097539703427203

[9] Michael R. Garey and David S. Johnson. 1979. *Computers and intractability : a guide to the theory of NP-completeness /*. San Francisco : W. H. Freeman, 1979, 338 p. CALL NUMBER: QA76.6 .G35.

[10] P-L Giscard and RC Wilson. 2017. The All-Paths and Cycles Graph Kernel. *arXiv preprint arXiv:1708.01410* (2017).

[11] IMDb. Accessed October 2017. IMDb Datasets. http://www.imdb.com/interfaces/. (Accessed October 2017).

[12] Sergey Ioffe. 2010. Improved consistent sampling, weighted minhash and l1 sketching. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 246–255.

[13] Glen Jeh and Jennifer Widom. 2002. SimRank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 538–543.

[14] Glen Jeh and Jennifer Widom. 2003. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web*. Acm, 271–279.

[15] Leo Katz. 1953. A new status index derived from sociometric analysis. *Psychometrika* 18, 1 (1953), 39–43.

[16] P. Legendre and L.F.J. Legendre. 1998. *Numerical Ecology*. Elsevier Science.

[17] Elizabeth A Leicht, Petter Holme, and Mark EJ Newman. 2006. Vertex similarity in networks. *Physical Review E* 73, 2 (2006), 026120.

[18] Jure Leskovec and Julian J Mcauley. 2012. Learning to discover social circles in ego networks. In *Advances in neural information processing systems*. 539–547.

[19] Weiping Liu and Linyuan Lü. 2010. Link prediction based on local random walk. *EPL (Europhysics Letters)* 89, 5 (2010), 58007.

[20] Zemin Liu, Vincent W Zheng, Zhou Zhao, Fanwei Zhu, Kevin Chen-Chuan Chang, Minghui Wu, and Jing Ying. 2017. Semantic Proximity Search on Heterogeneous Graph by Proximity Embedding.. In *AAAI*. 154–160.

[21] Linyuan Lü and Tao Zhou. 2011. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications* 390, 6 (2011), 1150 – 1170. https://doi.org/10.1016/j.physa.2010.11.027

[22] Sascha Rothe and Hinrich Schütze. 2014. Cosimrank: A flexible & efficient graph-theoretic similarity measure. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1. 1392–1402.

[23] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and S Yu Philip. 2017. A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering* 29, 1 (2017), 17–37.

[24] Chuan Shi, Chong Zhou, Xiangnan Kong, Philip S Yu, Gang Liu, and Bai Wang. 2012. HeteRecom: a semantic-based recommendation system in heterogeneous networks. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1552–1555.

[25] SNAP. Accessed October 2017. NetInf. http://snap.stanford.edu/netinf/. (Accessed October 2017).

[26] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. 2011. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment* 4, 11 (2011), 992–1003.

[27] The Google Ngram Viewer Team, part of Google Research. Accessed February 2018. Google Books Ngram Viewer. https://books.google.com/ngrams/info. (Accessed February 2018).

[28] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast random walk with restart and its applications. (2006).

[29] Guan Wang, Qingbo Hu, and Philip S Yu. 2012. Influence and similarity on heterogeneous networks. In *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 1462–1466.

[30] Wei Wu, Bin Li, Ling Chen, and Chengqi Zhang. 2017. Consistent Weighted Sampling Made More Practical. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1035–1043.

[31] Yun Xiong, Yangyong Zhu, and S Yu Philip. 2015. Top-k similarity join in heterogeneous information networks. *IEEE Transactions on Knowledge and Data Engineering* 27, 6 (2015), 1710–1723.