# Scaling link-based similarity search

Dániel Fogaras
Budapest University of Technology and Economics
Budapest, Hungary, H-1521
fd@cs.bme.hu

Balázs Rácz
Computer and Automation Research Institute of the Hungarian Academy of Sciences
Budapest, Hungary, H-1518
bracz+s65@math.bme.hu

## ABSTRACT

To exploit the similarity information hidden in the hyperlink structure of the web, this paper introduces algorithms scalable to graphs with billions of vertices on a distributed architecture. The similarity of multi-step neighborhoods of vertices are numerically evaluated by similarity functions including SimRank [18], a recursive refinement of cocitation; PSimRank, a novel variant with better theoretical characteristics; and the Jaccard coefficient, extended to multi-step neighborhoods. Our methods are presented in a general framework of Monte Carlo similarity search algorithms that precompute an index database of random fingerprints, and at query time, similarities are estimated from the fingerprints. The performance and quality of the methods were tested on the Stanford Webbase [17] graph of 80M pages by comparing our scores to similarities extracted from the ODP directory [24]. Our experimental results suggest that the hyperlink structure of vertices within four to five steps provide more adequate information for similarity search than single-step neighborhoods.

## Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms*; G.3 [**Mathematics of Computing**]: Probability and Statistics—*Probabilistic algorithms*

## General Terms

Algorithms, Theory, Experimentation

## Keywords

similarity search, link-analysis, scalability, fingerprint

## 1. INTRODUCTION

The development of similarity search algorithms between web pages is motivated by the "related pages" queries of web search engines and web document classification. Both applications require efficient evaluation of an underlying similarity function, which extracts similarities from either the textual content of pages or the hyperlink structure. This paper focuses on computing similarities solely from the hyperlink structure modeled by the *web graph*, with vertices corresponding to web pages and directed arcs to the hyperlinks between pages. In contrast to textual content, link structure is a more homogeneous and language independent source of information that is in general more resistant against spamming. The authors believe that complex link-based similarity functions with scalable implementations can play such an important role in similarity search as PageRank [25] does for query result ranking.

Several link-based similarity functions have been suggested over the web graph. To exploit the information in *multi-step neighborhoods*, SimRank [18] and the Companion [10] algorithms were introduced by adapting link-based ranking schemes [25, 19]. Further methods arise from graph theory such as similarity search based on network flows [21]. We refer to [20], which contains an exhaustive list of link-based similarity search methods.

Unfortunately, no scalable algorithm has so far been published that allows the computation of the above similarity scores in case of a graph with billions of vertices. First, all the above algorithms require random access to the web graph, which does not fit into main memory with standard graph representations. In addition, SimRank iterations update and store a quadratic number of variables: [18] reports experiments on graphs with less than 300K vertices. Finally, related page queries require off-line precomputation, since a document cannot be compared to all the others one-by-one at query time. It is not clear what we could precompute for an algorithm like the one in [21] with no information about the queried page.

In this paper we give scalable algorithms that can be used to evaluate multi-step link-based similarity functions over billions of pages on a distributed architecture. With a single machine, we conducted experiments on a test graph of 80M pages. Our primary focus is SimRank, which recursively refines the cocitation measure analogously to how PageRank refines in-degree ranking [25]. We give an improved SimRank variant; in addition, we also handle a similarity function that naturally extends the Jaccard coefficient from one-step to multi-step neighborhoods. Notice that scalability here is non-trivial, since the the Jaccard coefficient may involve extremely large sets: the multi-step neighborhood of a vertex usually contains a large portion of the pages [4].

All our methods are Monte Carlo: we precompute independent sets of *fingerprints* for the vertices, such that the similarities can be approximated from the fingerprints at query time. We only approximate the exact values; fortunately, the precision of approximation can be easily in-

Technical Report. Last modified: Nov 18, 2004.
Data Mining and Web Search Group, MTA SZTAKI.
http://www.ilab.sztaki.hu/websearch/Publications/

creased on a distributed architecture by precomputing independent sets of fingerprints and querying them in parallel.

We started to investigate the scalability of SimRank in [11], and we gave a Monte Carlo algorithm with the naive representation as outlined in the beginning of Section 2. The main contributions of this paper are summarized as follows:

- In Section 2.1 we present a scalable algorithm to compute approximate SimRank scores by using a database of fingerprint trees, a compact and efficient representation of precomputed random walks.
- In Section 2.2 we introduce and analyze PSimRank, a novel variant of SimRank with better theoretical properties and a scalable algorithm.
- In Section 2.3 Jaccard coefficient is naturally extended to multi-step neighborhoods with a scalable algorithm.
- In Section 3 we show that all the proposed Monte Carlo similarity search algorithms are especially suitable for distributed computing.
- In Section 4 we prove that our Monte Carlo similarity search algorithms approximate the similarity scores with a precision that tends to one exponentially with the number of fingerprints.
- In Section 5 we report experiments about the quality and performance of the proposed methods evaluated on the Stanford WebBase graph of 80M vertices [17].

In the remainder of the introduction we discuss related results, define "scalability," and recall some basic facts about SimRank.

## 1.1 Related Results

Unfortunately the algorithmic details of "related pages" queries in commercial web search engines are not publicly available. We believe that an accurate similarity search algorithm should exploit both the hyperlink structure and the textual content. For example, the pure link-based algorithms like SimRank can be integrated with classical text-based information retrieval tools [1] by simply combining the similarity scores. Alternatively, the similarities can be extracted from the anchor texts referring to pages as proposed by [7, 14].

Recent years have witnessed a growing interest in the scalability issue of link-analysis algorithms. Palmer et al. [26] formulated essentially the same scalability requirements that we will present in Section 1.2; they give a scalable algorithm to estimate the neighborhood functions of vertices. Analogous goals were achieved by the development of PageRank: Brin and Page [25] introduced PageRank algorithm using main memory of size proportional to the number of vertices. Then external memory extensions were published in [8, 13]. A large amount of research was done to attain scalability for personalized PageRank [15, 12]. The scalability of SimRank was also addressed by pruning [18], but this technique could only scale up to a graph with 300K vertices in the experiments of [18]. In addition, no theoretical argument was published about the error of approximating SimRank scores by pruning. In contrast, the algorithms of Section 2 were used to compute SimRank scores on a test graph of 80M vertices, and the theorems of Section 4 give bounds on the error of the approximation.

The key idea of achieving scalability by Monte Carlo algorithms was inspired by the seminal papers of Broder [5] and Cohen [9] estimating the resemblance of text documents and size of transitive closure of graphs, respectively. Both papers utilize min-hashing, the fingerprinting technique for the Jaccard coefficient that was also applied in [14] to scale similarity search based on anchor text. The main contribution of Section 2.3 is that we are able to generate fingerprints for multi-step neighborhoods with external memory algorithms. Monte Carlo algorithms with simulated random walks also play an important role in a different aspect of web algorithms, when a crawler attempts to download a uniform sample of web pages and compute various statistics [16, 27, 2] or page decay [3]. We refer to the book of Motwani and Raghavan [23] for more theoretical results about Monte Carlo algorithms solving combinatorial problems.

## 1.2 Scalability Requirements

In our framework similarity search algorithms serve two types of queries: the output of a $\mathsf{sim}(u, v)$ *similarity query* is the similarity score of the given pages $u$ and $v$; the output of a $\mathsf{related}_\alpha(u)$ *related query* is the set of pages for which the similarity score with the queried page $u$ is larger than the threshold $\alpha$. To serve queries efficiently we allow off-line precomputation, so the scalability requirements are formulated in the *indexing-query model*: we precompute an *index database* for a given web graph off-line, and later respond to queries on-line by accessing the database.

We say that a similarity search algorithm is *scalable* if the following properties hold:

- **Time:** The index database is precomputed within the time of a sorting operation, up to a constant factor. To serve a query the index database can only be accessed a constant number of times.
- **Memory:** The algorithms run in *external memory* [22]: the available main memory is constant, so it can be arbitrarily smaller than the size of the web graph.
- **Parallelization:** Both precomputation and queries can be implemented to utilize the computing power and storage capacity of tens to thousands of servers interconnected with a fast local network.

Observe that the time constraint implies that the index database cannot be too large. In fact our databases will be linear in the number $V$ of vertices (pages).

The memory requirements do not allow random access to the web graph. We will first sort the edges by their ending vertices using external memory sorting. Later we will read the entire set of edges sequentially as a stream, and repeat this process a constant number of times.

## 1.3 Preliminaries about SimRank

SimRank was introduced by Jeh and Widom [18] to formalize the intuition that "*two pages are similar if they are referenced by similar pages.*" The recursive *SimRank iteration* propagates similarity scores with a constant *decay factor* $c \in (0, 1)$ for vertices $u \neq v$:

$$\mathsf{sim}_{\ell+1}(u, v) = \frac{c}{|I(u)|\ |I(v)|} \sum_{u' \in I(u)} \sum_{v' \in I(v)} \mathsf{sim}_\ell(u', v')\,,$$

where $I(x)$ denotes the set of vertices linking to $x$; if $I(u)$ or $I(v)$ is empty, then $\mathsf{sim}_{\ell+1}(u, v) = 0$ by definition. For a vertex pair with $u = v$ we simply let $\mathsf{sim}_{\ell+1}(u, v) = 1$. The SimRank iteration starts with $\mathsf{sim}_0(u, v) = 1$ for $u = v$

and $\mathsf{sim}_0(u, v) = 0$ otherwise. The *SimRank score* is defined as the limit $\lim_{\ell\to\infty} \mathsf{sim}_\ell(u, v)$; see [18] for the proof of convergence. Throughout this paper we refer to $\mathsf{sim}_\ell(u, v)$ as a SimRank score, and regard $\ell$ as a parameter of SimRank.

The SimRank algorithm of [18] calculates the scores by iterating over all pairs of web pages, thus each iteration requires $\Theta(V^2)$ time and memory, where $V$ denotes the number of pages. Thus the algorithm does not meet the scalability requirements by its quadratic running time and random access to the web graph.

We recall two generalizations of SimRank from [18], as we will exploit these results frequently. *SimRank framework* refers to the natural generalization that replaces the average function in SimRank iteration by an arbitrary function of the similarity scores of pairs of in-neighbors. Obviously, the convergence does not hold for all the algorithms in the framework, but still $\mathsf{sim}_\ell$ is a well-defined similarity ranking. Several variants are introduced in [18] for different purposes.

For the second generalization of SimRank, suppose that a random walk starts from each vertex and follows the links backwards. Let $\tau_{u,v}$ denote the random variable equal to the first meeting time of the walks starting from $u$ and $v$; $\tau_{u,v} = \infty$, if they never meet; and $\tau_{u,v} = 0$, if $u = v$. In addition, let $f$ be an arbitrary function that maps the meeting times $0, 1, \ldots, \infty$ to similarity scores.

*Definition 1.* The *expected $f$-meeting distance* for vertices $u$ and $v$ is defined as $\mathbb{E}(f(\tau_{u,v}))$.

The above definition is adapted from [18] apart from the generalization that we do not assume uniform, independent walks of infinite length. In our case the walks may be pairwise independent, correlated, finite or infinite. For example, we will introduce PSimRank as an expected $f$-meeting distance of pairwise coupled random walks in Section 2.2.

The following theorem justifies the expected $f$-meeting distance as a generalization of SimRank. It claims that SimRank is equal to the expected $f$-meeting distance with uniform independent walks and $f(t) = c^t$, where $c$ denotes the decay factor of SimRank with $0 < c < 1$.

THEOREM 1. *For uniform, pairwise independent set of reversed random walks of length $\ell$, the equality $\mathbb{E}(c^{\tau_{u,v}}) = \mathsf{sim}_\ell(u, v)$ holds, whether $\ell$ is finite or not.*

The proof is published in [18] for the infinite case, and it can be easily extended to the finite case.

## 2. MONTE CARLO SIMILARITY SEARCH ALGORITHMS

In this section we give the first scalable algorithm to approximate SimRank scores. In addition, we introduce new similarity functions accompanied by scalable algorithms: PSimRank and the extended Jaccard coefficient.

All the algorithms fit into the framework of *Monte Carlo similarity search algorithms* that will be introduced through the example of SimRank. Recall that Theorem 1 expressed SimRank as the expected value $\mathsf{sim}_\ell(u, v) = \mathbb{E}(c^{\tau_{u,v}})$ for vertices $u, v$. Our algorithms generate reversed random walks, calculate the first meeting time $\tau_{u,v}$ and estimate $\mathsf{sim}_\ell(u, v)$ by $c^{\tau_{u,v}}$. To improve the precision of approximation, the sampling process is repeated $N$ times and the independent samples are averaged. The computation is shared between indexing and querying as shown in Algorithm 1, a naive implementation. During the precomputation phase we generate and store $N$ independent reversed random walks of length $\ell$ for each vertex, and the first meeting time $\tau_{u,v}$ is calculated at query time by reading the random walks from the precomputed index database.

**Algorithm 1** Indexing (naive method) and similarity query

$N$=number of fingerprints, $\ell$=path length, $c$=decay factor.
Indexing: Uses random access to the graph.

```
for i := 1 to N do
    for every vertex j of the web graph do
        Fingerprint[i][j][]:=random reversed path of
        length ℓ starting from j.
```

Query $\mathsf{sim}(u,v)$:

```
sim:=0
for i := 1 to N do
    Let k be the smallest offset with
    Fingerprint[i][u][k]=Fingerprint[i][v][k]
    if such k exists then
        sim:=sim+c^k
return sim/N
```

The main concept of Monte Carlo similarity search already arises in this example. In general *fingerprint* refers to a random object (a random walk in the example of SimRank) associated with a node in such a way, that the expected similarity of a pair of fingerprints is the similarity of their nodes. The Monte Carlo method precomputes and stores fingerprints in an index database and estimates similarity scores at query time by averaging. The main difficulties of this framework are as follows:

- During indexing (generating the fingerprints) we have to meet the scalability requirements of Section 1.2. For example, generating the random walks with the naive indexing algorithm requires random access to the web graph, thus we need to store all the links in main memory. To avoid this, we will first introduce algorithms utilizing $\Theta(V)$ main memory and then algorithms using memory of constant size, where $V$ denotes the number of vertices. These computational requirements are referred to as *semi-external memory* and *external memory* models [22], respectively. The parallelization techniques will be discussed in Section 3.
- To achieve a reasonably sized index database, we need a compact representation of the fingerprints. In the case of the previous example, the index database (including an inverted index for related queries) is of size $2 \cdot V \cdot N \cdot \ell$. In practical examples we have $V \approx 10^9$ vertices and $N = 100$ fingerprints of length $\ell = 10$, thus the database is in total 8000 gigabytes. We will show a compact representation that allows us to encode the fingerprints in $2 \cdot V \cdot N$ cells, resulting in an index database with a size of 800 gigabytes.
- We need efficient algorithms for evaluating queries. For queries the main idea is that the similarity matrix is sparse, for a page $u$ there are relatively few other pages that have non-negligible similarity to $u$. We will give algorithms that enumerate these pages in time proportional to their number.

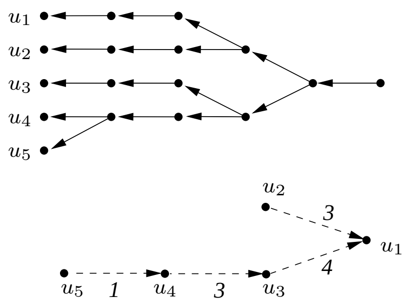


**Figure 1: Representing the first meeting times of coalescing reversed walks of $u_1$, $u_2$, $u_3$, $u_4$ and $u_5$ (above) with a fingerprint graph (below). For example, the fingerprints of $u_2$ and $u_5$ first meet at time $\tau_{u_2,u_5} = \max\{3, 4\} = 4$.**

## 2.1 SimRank

The main idea of this section is that we do not generate totally independent sets of reversed random walks as in Algorithm 1. Instead, we generate a set of *coalescing* walks: each pair of walks will follow the same path after their first meeting time. (This coupling is commonly used in the theory of random walks.) More precisely, we start a reversed walk from each vertex. In each time step, the walks at different vertices step independently to an in-neighbor chosen uniformly. If two walks are at the same vertex, they follow the same edge.

Notice that we can still estimate $\mathsf{sim}_\ell(u, v) = \mathbb{E}(c^{\tau_{u,v}})$ from the first meeting time $\tau_{u,v}$ of coalescing walks, since any pair of walks are independent until they first meet. We will show that the meeting times of coalescing walks can be represented in a surprisingly compact way by storing only one integer for each vertex instead of storing walks of length $\ell$. In addition, coalescing walks can be generated more efficiently by the algorithm discussed in Section 2.1.3 than totally independent walks.

### 2.1.1 Fingerprint trees

A set of coalescing reversed random walks can be represented in a compact and efficient way. The main idea is that we do not need to reconstruct the actual paths as long as we can reconstruct the first meeting times for each pair of them. To encode this, we define the *fingerprint graph* (FPG) for a given set of coalescing random walks as follows.

The vertices of FPG correspond to the vertices of the web graph indexed by $1, 2, \ldots, V$. For each vertex $u$, we add a directed edge $(u, v)$ to the FPG for at most one vertex $v$ with

(1) $v < u$ and the fingerprints of $u$ and $v$ first meet at time $\tau_{u,v} < \infty$;

(2) among vertices satisfying (1) vertex $v$ has earliest meeting time $\tau_{u,v}$;

(3) and given (1-2), the index of $v$ is minimal.

Furthermore we label the edge $(u, v)$ with $\tau_{u,v}$. An example for a fingerprint graph is shown as Fig. 1.

The most important property of the compact FPG representation that it still allows us to reconstruct $\tau_{u,v}$ values with the following algorithm. For a pair of nodes $u$ and $v$ consider

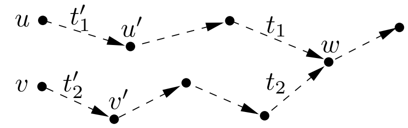


**Figure 2: Notation of specific vertices and edge labels of a fingerprint graph. In the example $|P(u, w)| = 3$ and $|P(v, w)| = 4$.**

the unique paths in the FPG starting from $u$ and $v$. If these paths have no vertex in common, then $\tau_{u,v} = \infty$. Otherwise take the parts until the first intersection; let $t_1$ and $t_2$ denote the labels of the last edges in the parts we selected; and let $t_1 = 0$ (or $t_2 = 0$), if $u$ (or $v$) is the first intersection point. Then $\tau_{u,v} = \max\{t_1, t_2\}$, see the example of Fig. 1. The correctness of this algorithm with further properties of the FPG is summarized by the following lemma.

LEMMA 2. *Consider the fingerprint graph for a set of coalescing random walks. This graph is a directed acyclic graph, each node has out-degree at most 1, thus it is a forest of rooted trees with edges directed towards the roots.*

*Consider the unique path in the fingerprint graph starting from vertex $u$. The indices of nodes it visits are strictly decreasing, and the labels on the edges are strictly increasing.*

*With the algorithm detailed above all $\tau_{u,v}$ values can be determined.*

PROOF. The first two statements naturally follow from the definition of fingerprint graphs. Now, we prove that for any two vertices $u$, $v$ the first meeting time $\tau_{u,v}$ can be calculated by the algorithm detailed above the lemma.

First we prove that $\tau_{u,v} < \infty$ iff $P(u)$ and $P(v)$ intersect each other, where $P(x)$ denotes the unique path in the FPG starting from $x$. If a directed edge connects two vertices in the FPG, then they have a finite meeting time. Notice that the relation $\{\, (u, v) : \tau_{u,v} < \infty \,\}$ is transitive, due to the coalescing property of the walks. Thus any two vertices $u$ and $v$ in the same (undirected) connected component of the fingerprint graph have finite meeting time. On the other hand, each connected component of an FPG is a rooted tree with edges directed towards the root. If $\tau_{u,v} < \infty$ would hold for $u$ and $v$ in two different trees (components), then the same relation would hold for the roots of these trees by transitivity, and there would exist an FPG edge starting from the root with larger index, which is a contradiction. So far, we have seen that $\tau_{u,v} < \infty$ iff the vertices $u$ and $v$ fall into the same component of the FPG. The latter is equivalent with saying $P(u)$ and $P(v)$ intersect each other, since the components are reversed rooted trees.

Now, we will show that $\tau_{u,v} = \max\{t_1, t_2\}$ holds for any vertices $u, v$ with $\tau_{u,v} < \infty$ as calculated by the algorithm of the lemma. Let us denote by $|P(x, w)|$ the number of edges in $P(x)$ from $x$ to $w$, and $x'$ the first edge of $P(x)$, if $|P(x, w)| > 0$ for $x = u, v$. Furthermore we will refer to the labels of $u'$ and $v'$ as $t_1'$ and $t_2'$; the first intersection point of $P(u)$ and $P(v)$ will be denoted by $w$. Recall that $t_1$ and $t_2$ denote the labels of the edges of $P(u)$ and $P(v)$ with ending vertex $w$; and $t_1 = 0$ (or $t_2 = 0$) if $|P(u, w)| = 0$ (or $|P(v, w)| = 0$). We refer to Fig. 2 summarizing the notation.

We will proceed induction on $k = |P(u, w)| + |P(v, w)|$ to prove that $\tau_{u,v} = \max\{t_1, t_2\}$ holds for any vertices $u, v$ with

$\tau_{u,v} < \infty$. The case of $k = 1$ is trivial, as it implies that the vertices $u$ and $v$ are connected by an edge in the FPG and the label of this edge equals $\tau_{u,v}$. Furthermore one of $t_1$ and $t_2$ equals this label, and the other is zero.

The following property of coalescing walks will be referred to as *generalized transitivity.* For any vertices $u, v, z$

$$\tau_{u,v} < \infty \text{ and } \tau_{v,z} \leq \tau_{u,v} \implies \tau_{u,v} = \tau_{u,z}.$$

The statement is trivial, since the first meeting time $\tau_{u,v}$ of the walks of $u$ and $v$ can be expressed as the meeting time $\tau_{u,z}$, if the walks of $v$ and $z$ coalesce not later than $\tau_{u,v}$.

To proceed the induction from $k$ to $k+1$ suppose that $u = w$ or $v = w$. Without loss of generality, we assume that $u = w$ and $v \neq w$. Since the indices of the vertices visited by $P(v)$ decreases, $w = u < v$ holds. By the definition of the FPG, among the vertices with smaller index than $v$ the meeting time $\tau_{v,v'}$ is minimal, thus $\tau_{v,v'} \leq \tau_{u,v}$ holds. Then by applying generalized transitivity we get $\tau_{u,v} = \tau_{u,v'}$, which is equal to $\max\{t_1, t_2\} = t_2$ by induction.

In case of $u \neq w$ and $v \neq w$ we suppose that $t'_2 \leq t'_1$ without loss of generality. If $u < v$, then $\tau_{v,v'} \leq \tau_{u,v}$ by the definition of the FPG. Analogously, if $u > v$, then $\tau_{u,u'} \leq \tau_{u,v}$, and by applying this inequality we get $\tau_{v,v'} = t'_2 \leq t'_1 = \tau_{u,u'} \leq \tau_{u,v}$. In both cases the inequality $\tau_{v,v'} \leq \tau_{u,v}$ holds, so we get $\tau_{u,v} = \tau_{u,v'}$ by the generalized transitivity. By induction $\tau_{u,v} = \tau_{u,v'} = \max\{t_1, t_2\}$, if $v' \neq w$; otherwise $\tau_{u,v} = \tau_{u,v'} = \max\{t_1, 0\} = \max\{t_1, t_2\}$, where the last equality follows from $t_1 \geq t'_1 \geq t'_2 = t_2$. This completes the proof. □

By the lemma, the fingerprint graph is a collection of rooted trees referred to as *fingerprint trees.* The main observation for storage and query is that the partition of nodes into trees preserves the locality of the similarity function.

### 2.1.2 Fingerprint database and query

The first advantage of the fingerprint graph is that it represents all first meeting times for a set of coalescing walks of length $\ell$ in compact manner. It is compact, since every vertex has at most one out-edge in an FPG, so the size of one graph is $V$, and $N \cdot V$ bounds the total size.[1] This is a significant improvement of the naive representation of the walks with a size of $N \cdot V \cdot \ell$.

The second important property of the fingerprint graph is that two vertices have non-zero estimated similarity iff they fall into the same fingerprint tree. Thus, when serving a related$(u)$ query it is enough to read and traverse from each of the $N$ fingerprint graphs the unique subtree containing $u$. Therefore in a *fingerprint database*, we store the fingerprint graphs ordered as a collection of fingerprint trees, and for each vertex $u$ we also store the identifiers of the $N$ trees containing $u$. By adding the identifiers the total size of the database is no more than $2 \cdot N \cdot V$.

A related$(u)$ query requires $N + 1$ accesses to the fingerprint database: one for the tree identifiers and then $N$ more for the fingerprint trees of $u$. A sim$(u, v)$ query accesses the fingerprint database at most $N + 2$ times, by loading two lists of identifiers and then the trees containing both $u$ and $v$. For both type of queries the trees can be traversed in time linear compared to the size of the tree.

[1] To be more precise we need $V(\lceil\log(V)\rceil + \lceil\log(\ell)\rceil)$ bits for an FPG to store the labelled edges. Notice that the weights require no more than $\lceil\log(\ell)\rceil = 4$ bits for each vertex for typical value of $\ell = 10$.

**Algorithm 2** Indexing (using $2 \cdot V$ main memory)

$N$=number of fingerprints, $\ell$=length of paths. Uses subroutine GenRndInEdges that generates a random in-edge for each vertex in the graph and stores its source in an array.

**for** $i := 1$ to $N$ **do**
  **for** every vertex $j$ of the web graph **do**
    PathEnd[$j$] := $j$ */*start a path from j*/*
  **for** $k$:=1 to $\ell$ **do**
    NextIn[] := GenRndInEdges();
    **for** every vertex $j$ with PathEnd[$j$]≠"stopped" **do**
      PathEnd[$j$]:=NextIn[PathEnd[$j$]]
      */*extend the path*/*
    SaveNewFPGEdges(PathEnd)
  Collect edges into trees and save as FPG$_i$.

Notice that the query algorithms do not meet all the scalability requirements: although the number of database accesses is constant (at most $N+2$), the memory requirement for storing and traversing one fingerprint tree may be as large as the number of pages $V$. Thus, theoretically the algorithm may use as much as $V$ memory.

Fortunately, in case of web data the algorithm performs as an external memory algorithm. As verified by our numerical experiments on 80M pages in Section 5.3 the average sizes of fingerprint trees are approximately 100–200 for reasonable path lengths. Even the largest trees in our database had at most 10K–20K vertices, thus 50Kbytes of data needs to be read for each database access in worst case.

### 2.1.3 Building the fingerprint database

It remains to show a scalable algorithm to generate coalescing sets of walks and compute the fingerprint graphs.

As opposed to the naive algorithm generating the fingerprints one-by-one, we generate all fingerprints together. With one iteration we extend all partially generated fingerprints by one edge. To achieve this, we generate one uniform in-edge $e_j$ for each vertex $j$ independently. Then extend with edge $e_j$ each of those fingerprints that have the same last node $j$. This method generates a coalescing set of walks, since a pair of walks will be extended with the same edge after they first meet, but they were independent before.

The pseudo-code is displayed as Algorithm 2, where NextIn[$j$] stores the starting vertex of the randomly chosen edge $e_j$, and PathEnd[$j$] is the ending vertex of the partial fingerprint that started from $j$. To be more precise, if a group of walks already met, then PathEnd[$j$]="stopped" for every member $j$ of the group except for the smallest $j$. The SaveNewFPGEdges subroutine detects if a group of walks meets in the current iteration, saves the fingerprint tree edges corresponding to the meetings and sets PathEnd[$j$]="stopped" for all non-minimal members $j$ of the group. SaveNewFPGEdges detects new meetings by a linear time counting sort of the non-stopped elements of PathEnd array.

The subroutine GenRndInEdges may generate a set of random in-edges with a simple external memory algorithm if the edges are sorted by the ending vertices. Notice that a significant improvement can be achieved by generating and saving all the required random edge-sets together during a *single scan* over the edges of the web graph. Thus, all the $N \cdot \ell$ edge-scans can be replaced by one edge-scan and saving

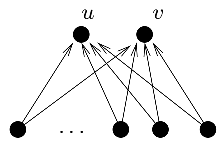


**Figure 3: When SimRank fails: pages $u$ and $v$ have $k$ witnesses for similarity, yet their SimRank score is smaller than $\frac{1}{k}$.**

the sets of in-edges. Then GenRndInEdges sequentially reads the $N \cdot \ell$ arrays of size $V$ from disk.

The algorithm outlined above fits into the semi-external memory model, since it utilizes $2 \cdot V$ main memory to store the PathEnd and NextIn arrays. (The counter sort operation of SaveNewFPGEdges may reuse NextIn array, so it does not require additional storage capacity.) The algorithm can be easily converted into the external memory model by keeping PathEnd and NextIn arrays on the disk and by replacing Lines 6-8 of Algorithm 2 with external sorting and merging processes. Furthermore, at the end of the indexing the individual fingerprint trees can be collected with $\ell$ sorting and merging operations, as the longest possible path in each fingerprint tree is $\ell$ (due to Lemma 2 the labels are strictly increasing but cannot grow over $\ell$).

In a *distributed system*, where up to hundreds of modest capacity machines are available with fast network connections between them, we can eliminate all the disk I/O for the precomputation phase.

We split the web graph so that each participating computer gets a part of the vertices so, that it can hold the (in-)edge set associated with those vertices in its main memory, along with an array of *tokens* sized roughly the number of vertices it is responsible for. Each token represents a partial fingerprint that has its current vertex from the set associated with the current host. Each host generates a set of random in-edges for those vertices it is responsible for, and advances the tokens in its property with the respective edges. Then the tokens are transferred on the network to their new owner. Now the walks that have just met are in the main memory of the machine which is responsible for the meeting point vertex, thus are easily found and the required edge in the fingerprint graph can be outputted.

## 2.2 PSimRank

In this section we give a new SimRank variant with properties extending those of Minimax SimRank [18], a non-scalable algorithm that cannot be formulated in our framework. The new similarity function will be expressed as an expected $f$-meeting distance by modifying the distribution of the set of random walks and by keeping $f(t) = c^t$.

A deficiency of SimRank can be best viewed by an example. Consider two very popular web portals. Many users link to both pages on their personal websites, but these pages are not reported to be similar by SimRank. An extreme case is depicted on Fig. 3 with portals $u$ and $v$ having the same in-neighborhood of size $k$. Though the $k$ pages are totally dissimilar in the link-based sense, we would still intuitively regard $u$ and $v$ as similar. Unfortunately SimRank is counter-intuitive in this case, as $\mathsf{sim}_\ell(u,v) = c \cdot \frac{1}{k}$ converges to zero with the number $k$ of common in-neighbors.

### 2.2.1 Coupled random walks

We define PSimRank as the expected $f$-meeting distance of a set of random walks, which are not independent, as in case of SimRank, but are *coupled* so that a pair of them can find each other more easily.

We solve the deficiency of SimRank by allowing the random walks to meet with higher probability when they are close to each other: a pair of random walks at vertices $u', v'$ will advance to the same vertex (i.e., meet in one step) with probability of the Jaccard coefficient $\frac{|I(u') \cap I(v')|}{|I(u') \cup I(v')|}$ of their in-neighborhoods $I(u')$ and $I(v')$.

*Definition 2.* PSimRank is the expected $f$-meeting distance with $f(t) = c^t$ (for some $0 < c < 1$) of the following set of random walks. For each vertex $u$, the random walk $X_u$ makes $\ell$ uniform independent steps on the transposed web graph starting from point $u$. For each pair of vertices $u$, $v$ and time $t$, assume the random walks are at position $X_u(t) = u'$ and $X_v(t) = v'$. Then

- with probability $\frac{|I(u') \cap I(v')|}{|I(u') \cup I(v')|}$ they both step to the same uniformly chosen vertex of $I(u') \cap I(v')$;
- with probability $\frac{|I(u') \setminus I(v')|}{|I(u') \cup I(v')|}$ the walk $X_u$ steps to a uniform vertex in $I(u') \setminus I(v')$ and the walk $X_v$ steps to an independently chosen uniform vertex in $I(v')$;
- with probability $\frac{|I(v') \setminus I(u')|}{|I(u') \cup I(v')|}$ the walk $X_v$ steps to a uniform vertex in $I(v') \setminus I(u')$ and the walk $X_u$ steps to an independently chosen uniform vertex in $I(u')$.

We give a set of random walks satisfying the coupling of the definition. For each time $t \geq 0$ we choose an independent random permutation $\sigma_t$ on the vertices of the web graph. At time $t$ if the random walk from vertex $u$ is at $X_u(t) = u'$, it will step to the in-neighbor with smallest index given by the permutation $\sigma_t$, i.e.,

$$X_u(t+1) = \operatorname*{argmin}_{u'' \in I(u')} \sigma_t(u'')$$

It is easy to see that the random walk $X_u$ takes uniform independent steps, since we have a new permutation for each step. The above coupling is also satisfied, since for any pair $u', v'$ the vertex $\operatorname{argmin}_{w \in I(u') \cup I(v')} \sigma_t(w)$ falls into the sets $I(u') \cap I(v')$, $I(u') \setminus I(v')$, $I(v') \setminus I(u')$ with respective probabilities

$$\frac{|I(u') \cap I(v')|}{|I(u') \cup I(v')|}, \frac{|I(u') \setminus I(v')|}{|I(u') \cup I(v')|} \text{ and } \frac{|I(v') \setminus I(u')|}{|I(u') \cup I(v')|}.$$

### 2.2.2 PSimRank in SimRank framework

Now we prove that PSimRank is in the SimRank framework, i.e., the scores can be formulated by iterations that propagate similarities over the pairs of in-neighbors analogously to SimRank. The PSimRank-iterations provide an exact quadratic algorithm to compute PSimRank scores. Furthermore, the iterative formulation indicates that PSimRank scores are determined by Definition 2 and the values do not depend on the actual choice of the coupling.

Let $\tau_{u,v}$ denote the first meeting time of the walks of $X_u, X_v$ starting from vertices $u, v$; and $\tau_{u,v} = \infty$ if the walks never meet. Then PSimRank scores for path length $\ell$ can be expressed by definition as $\mathsf{psim}_\ell(u,v) = \mathbb{E}(c^{\tau_{u,v}})$. It is trivial that $\mathsf{psim}_0(u,v) = 1$, if $u = v$; and otherwise $\mathsf{psim}_0(u,v) = 0$.

By applying the law of total expectation on the first step of the walks $X_u$ and $X_v$, and time shift we get the following *PSimRank iterations*:

$$\begin{aligned}
\mathsf{psim}_{\ell+1}(u,v) &= 1, \text{ if } u=v;\\
\mathsf{psim}_{\ell+1}(u,v) &= 0, \text{ if } I(u)=\emptyset \text{ or } I(v)=\emptyset;\\
\mathsf{psim}_{\ell+1}(u,v) &= c\cdot\Bigg[\tfrac{|I(u)\cap I(v)|}{|I(u)\cup I(v)|}\cdot 1+\\
&+\tfrac{|I(u)\setminus I(v)|}{|I(u)\cup I(v)|}\cdot\tfrac{1}{|I(u)\setminus I(v)||I(v)|}\sum_{\substack{u'\in I(u)\setminus I(v)\\ v'\in I(v)}}\mathsf{psim}_\ell(u',v')+\\
&+\tfrac{|I(v)\setminus I(u)|}{|I(u)\cup I(v)|}\cdot\tfrac{1}{|I(v)\setminus I(u)||I(u)|}\sum_{\substack{v'\in I(v)\setminus I(u)\\ u'\in I(u)}}\mathsf{psim}_\ell(u',v')\Bigg].
\end{aligned}$$

### 2.2.3 Computing PSimRank

To achieve a scalable algorithm for PSimRank we modify the SimRank indexing and query algorithms introduced in Section 2.1. The following result allows us to use the compact representation of fingerprint graphs.

LEMMA 3. *Any set of random walks satisfying the PSimRank requirements are coalescing, i.e., any pair follows the same path after their first meeting time.*

PROOF. Let $u$ and $v$ be arbitrary nodes. By the first coupling requirement, if at time $t$ the random walks $X_u$ and $X_v$ are at the same nodes $u'=v'$, then $I(u')=I(v')$, thus with probability $\frac{|I(u')\cap I(v')|}{|I(u')\cup I(v')|}=1$ they proceed to the same vertex. □

To apply the indexing algorithm of SimRank, we only need to ensure the pairwise coupling. This can be accomplished by simply replacing the GenRndInEdges procedure. Recall, that for SimRank this procedure generated one independent, uniform in-edge for each vertex $v$ in the graph. In case of PSimRank, GenRndInEdges chooses a permutation $\sigma$ at random; and then for each vertex $v$ the in-neighbor with smallest index under the permutation $\sigma$ is selected, i.e., vertex $\operatorname{argmin}_{v'\in I(v)}\sigma(v')$ is chosen.

As in the case of the GenRndInEdges for SimRank, all the required sets of random in-edges can be generated within a single scan over the edges of the web graph, if the edges are sorted by the ending vertices. The random permutations can be stored in small space by random linear transformations as in [6]. With this method the external memory implementation of SimRank can be extended to PSimRank.

## 2.3 Extended Jaccard coefficient

In this section we formally define the extended Jaccard coefficient, and give efficient (Monte Carlo) approximation algorithms in the indexing-query model by applying min-hashing [5], the well-known fingerprinting technique for estimating Jaccard coefficient between arbitrary sets. The main contribution of this section is that we give semi-external memory, external memory and distributed algorithms similar to PageRank iterations [25, 8] that compute the min-hash fingerprints for the multi-step neighborhoods of vertices. The proposed methods can be further parallelized using the methods described in Section 3.

The extended Jaccard coefficient is defined as the exponentially weighted sum of the Jaccard coefficients of larger neighborhoods.

*Definition 3.* Let $I_k(v)$ be the $k$-in-neighborhood of $v$, i.e., the set of vertices from where vertex $v$ can be reached using at most $k$ directed edges. The *extended Jaccard coefficient, XJaccard* for length $\ell$ of vertices $u$ and $v$ is defined as

$$\mathsf{xjac}_\ell(u,v)=\sum_{k=1}^{\ell}\frac{|I_k(u)\cap I_k(v)|}{|I_k(u)\cup I_k(v)|}\cdot c^k(1-c)$$

We will use the following *min-hash* fingerprinting technique for Jaccard coefficients [5]: take a random permutation $\sigma$ of the vertices and represent each set $I_k(v)$ with the minimum value of this permutation over the set $I_k(v)$ as a fingerprint. Then for each distance $k$ and vertices $u$, $v$ the probability of these fingerprints to match equals the Jaccard coefficient $\frac{|I_k(u)\cap I_k(v)|}{|I_k(u)\cup I_k(v)|}$. We can use this for each $k=1,\dots,\ell$ to get an $\ell$ sized fingerprint of each vertex, from which the extended Jaccard coefficients can be approximated for any pair of vertices.

More precisely, we calculate the following fingerprint for each vertex $v$ and each $k=1,\dots,\ell$:

$$\mathrm{fp}_k(v)=\min_{v'\in I_k(v)}\sigma(v')$$

Then by taking these as random variables the following statement holds (note that we use the same random permutation $\sigma$ for each step).

LEMMA 4.

$$\mathsf{xjac}_\ell(u,v)=\mathbb{E}\left(\sum_{k=1}^{\ell}c^k(1-c)\,\mathbb{1}\{\mathrm{fp}_k(u)=\mathrm{fp}_k(v)\}\right)$$

PROOF. Using the linearity of expectation and the well-known fingerprinting technique for Jaccard coefficient the statement follows. □

Using this probabilistic formulation we can take $N$ independent sample to generate $N$ sets of fingerprints. Upon a query $\mathsf{xjac}_\ell(u,v)$ we load all the fingerprints for $u$ and $v$, and average the results of them to get an unbiased estimate of $\mathsf{xjac}_\ell(u,v)$. For serving related queries we load the fingerprints of the queried page and use standard inverted indexing techniques to find all the pages that have matching parts in their fingerprints.

Serving XJaccard queries requires a database of size $2\cdot V\cdot N\cdot\ell$, a similarity query uses two database accesses, and a related query uses up to $1+N\cdot\ell$ database accesses. As we will show in Section 5, the preferred length of fingerprints is approximately $\ell=4$ on the web graph, thus these figures are still reasonable. Furthermore, the factor $N$ can be eliminated by using $N$-way parallelization, as discussed in Section 3.

### 2.3.1 Precomputation of extended Jaccard coefficient

We give a semi-external memory algorithm first. The key observation is that we use the same permutation for generating all steps of the fingerprint, which allows the following recursion:

$$\mathrm{fp}_k(u)=\min_{u'\in I(u)\cup\{u\}}\mathrm{fp}_{k-1}(u')$$

Using this formula we can extend the fingerprints by one step using one edge-scan and the fingerprints of the previous step (see Algorithm 3).

**Algorithm 3** Precomputing extended Jaccard coefficients

$N$=number of fingerprints, $\ell$=length of fingerprints.

> **for** $i := 1$ to $N$ **do**
>   generate a random permutation $\sigma$.
>   **for** every vertex $j$ of the web graph **do**
>     $\mathsf{NFP}[j]{:=}\sigma(j)$ */*start the fingerprint*/*
>   **for** $k$:=1 to $\ell$ **do**
>     $\mathsf{FP}[]{:=}\mathsf{NFP}[]$
>     **for** every edge $(u,v)$ of the web graph **do**
>       $\mathsf{NFP}[v]{:=}\min(\mathsf{NFP}[v],\mathsf{FP}[u])$
>     save array $\mathsf{NFP}[]$ as $\mathsf{FP}_k[]$
> Merge arrays $\mathsf{FP}_k$, and create inverted index.

### 2.3.2 External memory and distributed indexing

Algorithm 3 for semi-external memory indexing of extended Jaccard coefficients is very similar to the classic PageRank computing method using power-iteration: each iteration scans the entire edge-set and updates a vector (indexed by the vertices) using the vector computed by the previous iteration. This allows us to adapt the external memory algorithms designed for PageRank [8, 13], and the distributed indexing technique by the authors [12]. Due to space constraints we will not quote these algorithms here.

In total with $N = 100$ and $\ell = 4$ the precomputation costs for extended Jaccard coefficients are thus similar to the precomputation cost for 400 PageRank iterations, with one remarkable difference: while PageRank can only be computed sequentially, the precomputation of extended Jaccard coefficients can be parallelized up to $N$-way.

## 3. MONTE CARLO PARALLELIZATION

In this section we discuss the parallelization possibilities of our methods. We show that all of them exhibit features (such as fault tolerance, load balancing and dynamic adaptation to workload) which makes them extremely applicable in large-scale web search engines.

All similarity methods we have given in this paper are organized around the same concepts:

- we compute a similarity measure by averaging $N$ independent samples from a certain random variable;
- the independent samples are stored in $N$ instances of an index database, each capable of producing a sample of the random variable for any pair of vertices.

The above framework allows a straightforward parallelization of both the indexing and the query: the computation of independent index databases can be performed on up to $N$ different machines. Then the databases are transferred to the backend computers that serve the query requests. When a request arrives to the frontend server, it asks all (up to $N$) backend servers, averages their answers and returns the results to the user.

The Monte Carlo parallelization scheme has many advantages that make it perfectly suitable to large-scale web search engines:

*The parallelization of queries and indexing* can be performed differently. For example, if indexing requires large capacity computers, then one can use a few of them to compute all the index databases. As the scarce resource for query is typically database access (disk seeks), and only little memory and computation is required, these databases can then be distributed to $N$ different backend servers.

*Fault tolerance.* If one or more backend servers cannot respond to the query in time, then the frontend can aggregate the results of the remaining ones and calculate the estimate from the available answers. This will not influence service availability, but results only in a slight loss of precision.

*Load balancing.* In case of very high query loads, more than $N$ backend servers (database servers) can be employed. A simple solution is to replicate the individual index databases. Better results are achieved if one calculates an independent index database for all the backend servers. In this case it suffices to ask *any* $N$ backend servers for a proper precision answer. This allows seamless load balancing, i.e., you can add more backend servers one-by-one as the demand increases.

Furthermore, this parallelization allows *dynamic adaptation to workload.* During times of excessive load the number of backend servers asked for each query ($N$) can be automatically reduced to maintain fast response times and thus service integrity. Meanwhile, during idle periods, this value can be increased to get higher precision for free (along with better utilization of resources). We believe that this feature is extremely important in the applicability of our results.

## 4. ERROR OF APPROXIMATION

As we have seen in earlier sections, a crucial parameter of our methods is the number $N$ of fingerprints. The index database size, indexing time, query time and database accesses are all linear in $N$. In this section we formally analyze the number of fingerprints needed for a proper precision approximation. Our theorems show that even a modest number of fingerprints (e.g., $N = 100$) suffices for the purposes of a web search engine.

To state our results we need a general model of Monte Carlo similarity functions that can accommodate our methods for SimRank, PSimRank and XJaccard. We will generalize similarity search over a set $\mathcal{V}$ of items. Let $M$ denote a random variable with a range being an arbitrary set $\mathcal{S}$. Consider a pair $(M, \{\, g_{u,v} : u, v \in \mathcal{V}\,\})$, where for each pair $u, v$ of items the function $g_{u,v} : \mathcal{S} \mapsto [0, 1]$ transforms the value of $M$ into an estimate of the similarity of $u$ and $v$.

*Definition 4.* A *Monte Carlo similarity function* $\widehat{\mathsf{sim}}(\cdot,\cdot)$ over a set $\mathcal{V}$ of items is calculated by taking $N$ independent instances $M_1, \ldots, M_N$ of the random variable $M$, and averaging the results of their transformations as $\widehat{\mathsf{sim}}(u, v) = \frac{1}{N}\sum_{i=1}^{N} g_{u,v}(M_i)$ for each pair $u, v \in \mathcal{V}$. Furthermore, we refer to $\mathsf{sim}(u, v) = \mathbb{E}(\, g_{u,v}(M)\,)$ as the *underlying similarity function*[2].

*Example 1.* In case of our SimRank approximation method, the value of the random variable $M$ is the set of fingerprint paths (for all vertex $u$). The transformation $g_{u,v}$ selects the paths for $u$ and $v$, calculates their first meeting time $\tau_{u,v}$, and returns $c^{\tau_{u,v}}$, where $c$ is the decay parameter of SimRank.

*Example 2.* In the general case, the set $\mathcal{S}$ is the set of all possible index databases, $g_{u,v}$ is the similarity query, i.e., the algorithm that takes an index database and calculates

[2]Naturally, the Monte Carlo similarity function $\widehat{\mathsf{sim}}(u, v)$ is an unbiased estimation of the underlying similarity function $\mathsf{sim}(u, v)$.

the estimated similarity of $u$ and $v$ using only that index database. The $\widehat{\mathsf{sim}}$ averaging is the role of the frontend, that distributes the queried node pair to all the participating backend servers (each of them owning an independent index database, i.e., an independent realization $M_i$ of the random variable $M$), collects their estimates and averages them.

Notice that the above definition of Monte Carlo similarity functions allows arbitrary correlation/dependence of different similarity scores within the same index database. This is essential, as our actual computable methods exhibit such dependence e.g., by coalescing random walks. Still we have strong results concerning the convergence of the estimates.

THEOREM 5. *For any Monte Carlo similarity function $\widehat{\mathsf{sim}}$ the absolute error converges to zero exponentially in the number of fingerprints $N$ and uniformly over the pair of items $u, v$. More precisely, for any $u, v \in \mathcal{V}$ and any $\delta > 0$ we have*

$$\Pr\{|\widehat{\mathsf{sim}}(u,v) - \mathsf{sim}(u,v)| > \delta\} < 2e^{-\frac{6}{7}N\delta^2}$$

PROOF. We shall use Bernstein's inequality in the following form: for any independent, identically distributed random variables $Z_i : i = 1, 2, \ldots, N$ that have a bounded range $[a, b]$, for any $\delta > 0$:

$$\Pr\{|\frac{1}{N}\sum_{i=1}^{N} Z_i - \mathbb{E}\,Z| > \delta\} \le 2e^{-N\frac{\delta^2}{2\,\mathrm{Var}\,Z + 2\delta(b-a)/3}}$$

Applying this for $Z_i = g_{u,v}(M_i)$ and using the bounds $Z_i \in [0, 1]$, $\mathrm{Var}\,Z_i \le \frac{1}{4}$, and $\delta < 1$ the statement follows. $\square$

Notice that the bound uniformly applies to all graphs and all similarity functions, such as SimRank, PSimRank and XJaccard. However, this bound concerns the convergence of the similarity score for one pair of vertices only. In the web search scenario, we typically use related queries, thus are interested in the relative order of pages according to their similarity to a given query page $u$.

THEOREM 6. *For any Monte Carlo similarity function $\widehat{\mathsf{sim}}$ and any fixed item $u$, the probability of interchanging two items in the similarity ranking of item $u$ converges to zero exponentially in the number of fingerprints $N$. More precisely, for each page $v$ and $w$, such that $\mathsf{sim}(u,v) > \mathsf{sim}(u,w)$ we have*

$$\Pr\{\widehat{\mathsf{sim}}(u,v) < \widehat{\mathsf{sim}}(u,w)\} < e^{-0.3N\delta^2}$$

*where $\delta = \mathsf{sim}(u,v) - \mathsf{sim}(u,w)$.*

Though a similar statement follows easily from the previous theorem, we give an independent (but similar) proof to achieve better constants.

PROOF. We shall use Bernstein's inequality one-sided: for any independent, identically distributed random variables $Z_i : i = 1, 2, \ldots, N$ that have a bounded range $[a, b]$, for any $\delta > 0$:

$$\Pr\{\frac{1}{N}\sum_{i=1}^{N} Z_i - \mathbb{E}\,Z < -\delta\} \le e^{-N\frac{\delta^2}{2\,\mathrm{Var}\,Z + 2\delta(b-a)/3}}$$

We set $Z_i = g_{u,v}(M_i) - g_{u,w}(M_i)$. Then $\frac{1}{N}\sum_{i=1}^{N} Z_i = \widehat{\mathsf{sim}}(u,v) - \widehat{\mathsf{sim}}(u,w)$, its expected value is $\mathsf{sim}(u,v) - \mathsf{sim}(u,w)$. We can bound the values: $Z_i \in [-1, 1]$ and thus the variance: $\mathrm{Var}\,Z_i \le 1$. We set $\delta = \mathsf{sim}(u,v) - \mathsf{sim}(u,w)$, thus we get

$$\Pr\{\widehat{\mathsf{sim}}(u,v) - \widehat{\mathsf{sim}}(u,w) < 0\} \le e^{-N\frac{\delta^2}{2+4/3}}$$

$\square$

These theorems mean that the Monte Carlo approximation can efficiently capture the big differences among the similarity scores. But when it comes to small differences, then the error of approximation obscures the actual similarity ranking, and an almost arbitrary reordering is possible. We believe, that for a web search inspired similarity ranking it is sufficient to distinguish between very similar, modestly similar, and dissimilar pages. We can formulate this requirement in terms of a slightly weakened version of classical information retrieval measures *precision* and *recall* [1].

Consider a related query for page $u$ with similarity threshold $\alpha$, i.e., the problem is to return the set of pages $S = \{v \in \mathcal{V} : \mathsf{sim}(u,v) > \alpha\}$. Our methods approximate this set with $\widehat{S} = \{v \in \mathcal{V} : \widehat{\mathsf{sim}}(u,v) > \alpha\}$. We weaken the notion of precision and recall to exclude a small, $\delta$ sized interval of similarity scores around the threshold $\alpha$: let $S_{+\delta} = \{v \in \mathcal{V} : \mathsf{sim}(u,v) > \alpha + \delta\}$, $S_{-\delta} = \{v \in \mathcal{V} : \mathsf{sim}(u,v) > \alpha - \delta\}$. Then the *expected $\delta$-recall* of a Monte Carlo similarity function is $\frac{\mathbb{E}(|\widehat{S} \cap S_{+\delta}|)}{|S_{+\delta}|}$ while the *expected $\delta$-precision* is $\frac{\mathbb{E}(|\widehat{S} \cap S_{-\delta}|)}{\mathbb{E}(|\widehat{S}|)}$. Furthermore, we introduce the notation $S^c_{-\delta} = \mathcal{V} \setminus S_{-\delta}$.

THEOREM 7. *For any Monte Carlo similarity function $\widehat{\mathsf{sim}}$, any page $u$, similarity threshold $\alpha$ and $\delta > 0$ the expected $\delta$-recall is at least*

$$1 - e^{-\frac{6}{7}N\delta^2}$$

*and the expected $\delta$-precision is at least*

$$1 - \frac{|S^c_{-\delta}|}{|S_{+\delta}|}\frac{1}{e^{\frac{6}{7}N\delta^2} - 1}\ .$$

PROOF. First we bound the expected $\delta$-recall.

$$\begin{aligned}
\mathbb{E}\left(|\widehat{S} \cap S_{+\delta}|\right) &= \mathbb{E}\Big(\sum_{v \in S_{+\delta}} \mathbb{1}\{v \in \widehat{S}\}\Big) \\
&= \sum_{v \in S_{+\delta}} \Pr\{v \in \widehat{S}\} \\
&\ge \sum_{v \in S_{+\delta}} \left(1 - e^{-\frac{6}{7}N\delta^2}\right) \\
&= |S_{+\delta}| \cdot \left(1 - e^{-\frac{6}{7}N\delta^2}\right),
\end{aligned}$$

where the second equation follows from the linearity of expectation; and the inequality follows from the one-sided absolute error bound $\Pr\{\widehat{\mathsf{sim}}(u,v) - \mathsf{sim}(u,v) < -\delta\} < e^{-\frac{6}{7}N\delta^2}$ that can be proved analogously to Theorem 5.

Now we turn to expected $\delta$-precision:

$$\begin{aligned}
1 - \frac{\mathbb{E}(|\widehat{S} \cap S_{-\delta}|)}{\mathbb{E}(|\widehat{S}|)} &= \frac{\mathbb{E}(|\widehat{S} \cap S^c_{-\delta}|)}{\mathbb{E}(|\widehat{S}|)} \\
&\le \frac{|S^c_{-\delta}|e^{-6/7 \cdot N\delta^2}}{|S_{+\delta}|(1 - e^{-6/7 \cdot N\delta^2})} \\
&= \frac{|S^c_{-\delta}|}{|S_{+\delta}|}\frac{1}{e^{6/7 \cdot N\delta^2} - 1}\ ,
\end{aligned}$$

where the inequality follows from $\mathbb{E}(\widehat{S}) \geq \mathbb{E}(|\widehat{S} \cap S_{+\delta}|)$ with the lower bound derived for the proof of expected $\delta$-recall; and from the bound $\mathbb{E}(|\widehat{S} \cap S^c_{-\delta}|) \leq |S^c_{-\delta}| \cdot e^{-6/7 \cdot N\delta^2}$ that can be proved with essentially the same steps as the lower bound on $\mathbb{E}(|\widehat{S} \cap S_{+\delta}|)$. $\square$

This theorem shows, that the expected $\delta$-recall converges to 1 exponentially and uniformly over all possible similarity functions, graphs and queried vertices of the graphs, while the expected $\delta$-precision converges to 1 exponentially for any fixed similarity function, graph and queried node. The $|S^c_{-\delta}|$ factor in the precision is not surprising, as there can be many items with just less than $\alpha - \delta$ similarity, and these can get into the result set. To prove stronger bounds we have to make assumptions (for example power law) about the distribution of similarity scores.

# 5. EXPERIMENTS

This section presents our experiments on the repository of 80M pages crawled by the Stanford WebBase project in 2001. The following problems are addressed by our experiments:

- How do the parameters $\ell$, $N$ and $c$ effect the quality of the similarity search algorithms? The dependence on path length $\ell$ show that multi-step neighborhoods of pages contain more valuable similarity information than single-step neighborhoods for up to $\ell \approx 5$.
- How do the qualities of SimRank, PSimRank and XJaccard relate to each other? We conclude that PSimRank outperforms all the other methods.
- What are the average and maximal sizes of fingerprint trees for SimRank and PSimRank? Recall that the running time and memory requirement of query algorithms are proportional to these sizes. We measured sizes as small as $100 - 200$ on average implying fast running time with low memory requirement.

## 5.1 Measuring the Quality of Similarity Scores

We briefly recall the method of Haveliwala et al. [14] to measure the quality of similarity search algorithms.

The similarity search algorithms will be compared to a ground truth similarity ordering extracted from the Open Directory Project (ODP, [24]) data, a hierarchical collection of webpages managed by thousands of volunteer editors. The ODP category tree implicitly encodes the similarity information, which can be decoded as follows. The ODP tree is collapsed into a fixed depth, such that the leaves contain the classes of documents (urls). Given a page $u$ the rest of the documents fall into the *same* class as $u$, a *sibling* class, a *cousin* class, etc. This induces a partial ordering of the documents, which will be referred to as the *familial ordering* with respect to $u$. The key assumption is that the true similarity to a page $u$ decreases monotonically with the familial ordering.

Intuitively we want to express the expected quality of a similarity ordering to a query page $u$ in comparison with the familial ordering of $u$, where $u$ is chosen uniformly at random. The two orderings are compared by the Kruskal-Goodman $\Gamma$ measure that gives score $+1$ to a pair $v$, $w$ if the two orderings agree on the similarity ordering of the pair, and it gives $-1$ if they order the pair reversely. As both orderings are partial, the $\Gamma$ value is defined as the average of scores over all pairs that are comparable by both orderings. To obtain a more precise measure focusing on the top region of the familial ordering, *sibling* $\Gamma$ measure [14] restricts the averaging to vertices that either fall into the same or a sibling class of $u$.

Finally, we enumerate the subtle differences between the sibling $\Gamma$ measure defined above and the original sibling $\Gamma$ introduced in [14]. The goal of the modifications was to make sibling $\Gamma$ more suitable for measuring the qualities of related queries.

- For each page $u$ we computed sibling $\Gamma$ on a truncated list of 100 pages with highest similarity to $u$. This truncation is reasonable, since for example the $\Gamma$ quality of a long list of 10,000 pages is almost independent of the quality of the first 100 resulting pages, which is the main interest of typical users of related queries.
- Recall that Kruskal-Goodman $\Gamma$ measures the quality of a similarity ranking to a given query page $u$. In our experiments we extended $\Gamma$ to an overall measure by *micro-averaging*: we computed $\Gamma$ for each page $u$, and then averaged these $\Gamma$ scores. In contrast, the method of [14] applies *macro-averaging* by averaging over all vertices $u, v, w$ the $+1$ and $-1$ credits given for ordering the pair $(\mathsf{sim}(u, v), \mathsf{sim}(u, w))$ correctly or not. With probabilistic terminology micro-averaging describes the quality of a $\mathsf{related}(u)$ query with uniformly chosen $u$, while macro-averaging describes the expected quality of ordering the pair $(\mathsf{sim}(u, v), \mathsf{sim}(u, w))$ for uniformly chosen pairs. We decided on micro-averaging, since our primary focus was related query, and we experienced that essentially the same tendencies can be measured by macro-averaging with slightly higher $\Gamma$ values than our micro-averaging method combined with list truncation.
- We discarded the page $u$ itself, when we evaluated the quality of the similarity ranking of $u$. This modification significantly decreased $\Gamma$ values by approximately 0.1, since our algorithms estimated $\mathsf{sim}(u, u) = 1$ perfectly. Even larger differences occurred for the parameter settings path length $\ell = 1, 2, 3$ and number of fingerprints $N = 10, 20, 30$. It is not surprising, since reducing $\ell$ and $N$ values decreases the number of pairs with non-zero estimated similarity scores. This modification caused the main difference between the $\Gamma$ values of this paper and those presented in [11] about SimRank.

## 5.2 Comparing the Qualities of the Methods with Various Parameter Settings

All the experiments were performed on a web graph of 78,636,371 pages crawled and parsed by the Stanford WebBase project in 2001. In our copy of the ODP tree 218,720 urls were found falling into 544 classes after collapsing the tree. The indexing process took 4 hours for SimRank, 14 hours for PSimRank and 27 hours for extended Jaccard coefficient with path length $\ell = 10$ and $N = 100$ fingerprints. We ran a semi-external memory implementation on a single machine with 2.8GHz Intel Pentium 4 processor, 2Gbytes main memory and Linux OS. The total size of the the computed database was 68Gbytes for (P)SimRank and 640Gbytes for XJaccard. Since sibling $\Gamma$ is based on similarity scores between vertices of the ODP pages, we only saved the fingerprints of the 218,720 ODP pages. A nice property

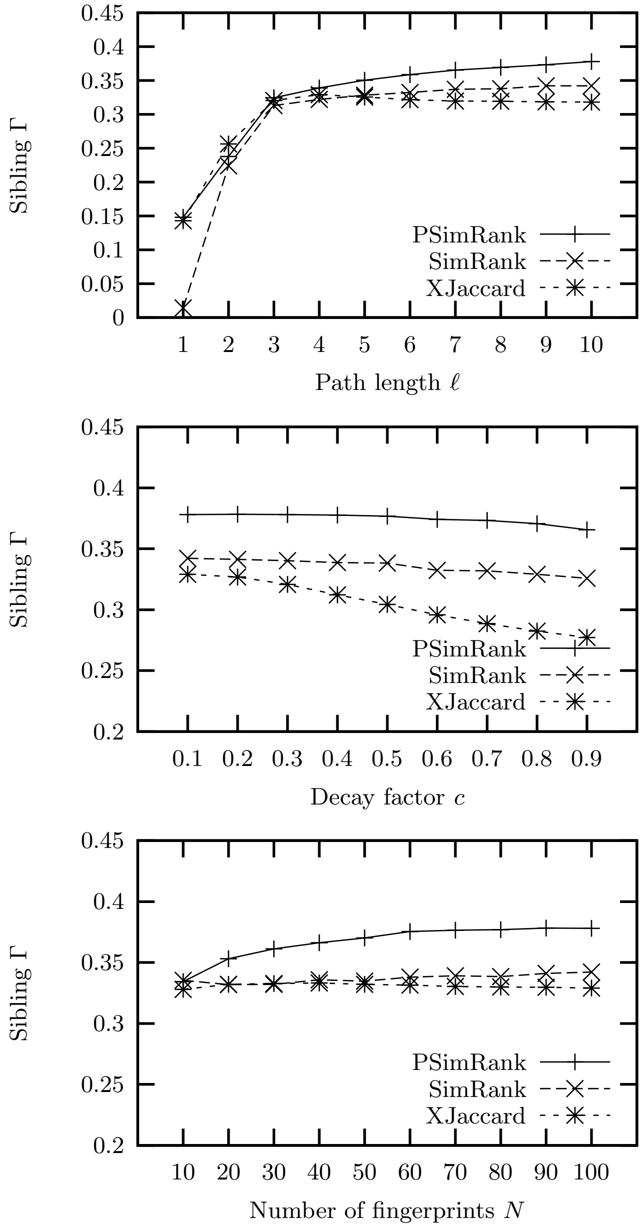


**Figure 4: Varying algorithm parameters independently with default settings $\ell = 10$ for SimRank and PSimRank $\ell = 4$ for XJaccard, $c = 0.1$, and $N = 100$.**

of our methods is that this truncation (resulting in sizes of 200Mbytes and 1.8Gbytes respectively) does not affect the returned scores for the ODP pages.

The results of the experiments are depicted on Fig. 4. Recall that sibling $\Gamma$ expresses the average quality of similarity search algorithms with $\Gamma$ values falling into the range $[-1, 1]$. The extreme $\Gamma = 1$ result would show that similarity scores completely agree with the ground truth similarities, while $\Gamma = -1$ would show the opposite. Our $\Gamma = 0.3 - 0.4$ values imply that our algorithms agree with the ODP familial ordering in $65 - 70\%$ of the pairs.

The radically increasing $\Gamma$ values for path length $\ell = 1, 2, 3, 4$ on the top diagram supports our basic assumption that the multi-step neighborhoods of pages contain valuable similarity information. The quality slightly increases for larger values of $\ell$ in case of PSimRank and SimRank, while sibling $\Gamma$ has maximum value for $\ell = 4$ in case of XJaccard. Notice the difference between the scale of the top diagram and the scales of the other two diagrams.

The middle diagram shows the tendency that the quality of similarity search can be increased by smaller decay factor. This phenomenon suggests that we should give higher priority to the similarity information collected in smaller distances and rely on long-distance similarities only if necessary. The bottom diagram depicts the changes of $\Gamma$ as a function of the number $N$ of fingerprints. The diagram shows slight quality increase as the estimated similarity scores become more precise with larger values of $N$.

Finally, we conclude from all the three diagrams that PSimRank scores introduced in Section 2.2 outperform all the other similarity search algorithms.

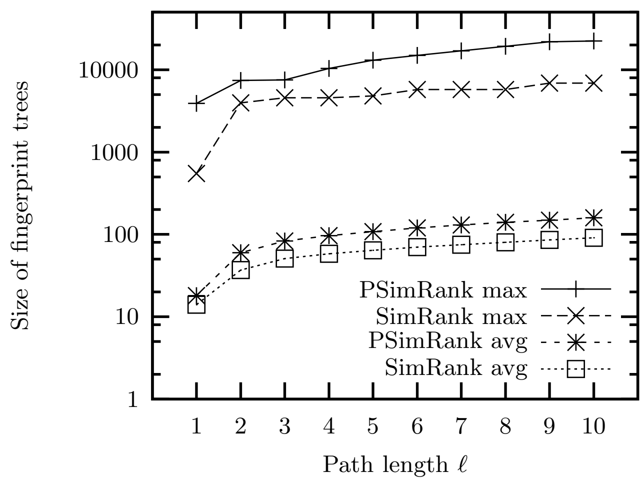


**Figure 5: Fingerprint tree sizes for 80M pages with $N = 100$ samples.**

## 5.3 Time and memory requirement of fingerprint tree queries

Recall from Section 2.1.2 that for SimRank and PSimRank queries $N$ fingerprint trees are loaded and traversed. $N$ can be easily increased with Monte Carlo parallelization, but the sizes of fingerprint trees may be as large as the number $V$ of vertices. This would require both memory and running time in the order of $V$, and thus violate the requirements of Section 1.2. The experiments verify that this problem does not occur in case of real web data.

Fig. 5 shows the growing sizes of fingerprint trees as a function of path length $\ell$ in databases containing fingerprints for all vertices of the Stanford WebBase graph. Recall that the trees are growing when random walks meet and the corresponding trees join into one tree. It is not surprising that the tree sizes of PSimRank exceed that of SimRank, since the correlated random walks meet each other with higher probabilities than the independent walks of SimRank.

We conclude from the lower curves of Fig. 5 that the the average tree sizes read for a query vertex is approximately 100–200, thus the algorithm performs like an external-memory algorithm on average in case of our web graph. Even the largest fingerprint trees have no more than 10–20K vertices, which is still very small compared to the 80M pages.

# 6. CONCLUSION

We introduced the framework of link-based Monte Carlo similarity search to achieve scalable algorithms for similarity functions evaluated from the multi-step neighborhoods of web pages. Within this framework, we presented the first algorithm to approximate SimRank scores with a near linear external memory method and parallelization techniques sufficient for large scale computation. In addition, we defined new similarity functions PSimRank and the extended Jaccard coefficient with scalable algorithms. Our experiments conducted on the Stanford WebBase graph of 80M pages demonstrate scalability and suggest that PSimRank outperforms SimRank and extended Jaccard coefficient in terms of quality.

# 7. ACKNOWLEDGEMENT

We would like to thank András Benczúr, Katalin Friedl, Dániel Marx, Tamás Sarlós and Andrew Twigg for valuable discussions on this research and for improving this manuscript by several comments and suggestions. Furthermore, we wish to acknowledge the Stanford WebBase project for providing us with the web graph for the experiments.

# 8. REFERENCES

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.

[2] Z. Bar-Yossef, A. Berg, S. Chien, J. Fakcharoenphol, and D. Weitz. Approximating aggregate queries about web pages via random walks. *Proc. of VLDB*, 2000.

[3] Z. Bar-Yossef, A. Z. Broder, R. Kumar, and A. Tomkins. Sic transit gloria telae: towards an understanding of the web's decay. *Proc. of WWW13*, 2004.

[4] A. Z. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. *Proc. of WWW9*, 2000.

[5] A. Z. Broder. On the resemblance and containment of documents. *Proc. of Compression and Complexity of Sequences (SEQUENCES'97)*, pages 21–29. IEEE Computer Society, 1997.

[6] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *J. Comput. Syst. Sci.*, 60(3):630–659, 2000.

[7] S. Chakrabarti, B. E. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. *Proc. of SIGMOD*, 1998.

[8] Y.-Y. Chen, Q. Gan, and T. Suel. I/O-efficient techniques for computing PageRank. *Proc. of CIKM*, 2002.

[9] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55(3):441–453, 1997.

[10] J. Dean and M. R. Henzinger. Finding related pages in the World Wide Web. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1467–1479, 1999.

[11] D. Fogaras and B. Rácz. A scalable randomized method to compute link-based similarity rank on the web graph. *Proc. of Clustering Information over the Web (ClustWeb), in conjunction with EDBT*, 2004.

[12] D. Fogaras and B. Rácz. Towards scaling fully personalized PageRank. *Proc. of Third Workshop on Algorithms and Models for the Web-Graph (WAW) in conjunction with FOCS*, 2004.

[13] T. H. Haveliwala. Efficient computation of PageRank. Technical Report 1999-31, Stanford University, 1999.

[14] T. H. Haveliwala, A. Gionis, D. Klein, and P. Indyk. Evaluating strategies for similarity search on the web. *Proc. of WWW11*, 2002.

[15] T. H. Haveliwala, S. Kamvar, and G. Jeh. An analytical comparison of approaches to personalizing PageRank. Technical report, Stanford University, 2003.

[16] M. R. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork. On near-uniform url sampling. *Proc. of WWW9*, 2000.

[17] J. Hirai, S. Raghavan, H. Garcia-Molina, and A. Paepcke. Webbase: a repository of web pages. *Proc. of WWW9*, 2000.

[18] G. Jeh and J. Widom. SimRank: A measure of structural-context similarity. *Proc. of SIGKDD*, 2002.

[19] J. Kleinberg. Authoritative sources in a hyperlinked environment. *J. of the ACM*, 46(5):604–632, 1999.

[20] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. *Proc. of CIKM*, pages 556–559. ACM Press, 2003.

[21] W. Lu, J. Janssen, E. Milios, and N. Japkowicz. Node similarity in networked information spaces. *Proc. of 2001 conference of the Centre for Advanced Studies on Collaborative research*, page 11. IBM Press, 2001.

[22] U. Meyer, P. Sanders, and J. Sibeyn. *Algorithms for Memory Hierarchies, Advanced Lectures*. LNCS, Springer, 2003.

[23] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[24] Open Directory Project (ODP). `www.dmoz.org`.

[25] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[26] C. R. Palmer, P. B. Gibbons, and C. Faloutsos. ANF: a fast and scalable tool for data mining in massive graphs. *Proc. of SIGKDD*, 2002.

[27] P. Rusmevichientong, D. M. Pennock, S. Lawrence, and C. L. Giles. Methods for sampling pages uniformly from the world wide web. *Proc. of AAAI Fall Symposium on Using Uncertainty Within Computation*, pages 121–128, 2001.