

# A Scalable Randomized Method to Compute Link-Based Similarity Rank on the Web Graph

Dániel Fogaras<sup>1,2</sup> and Balázs Rácz<sup>1,2</sup>

<sup>1</sup> Computer and Automation Research Institute of the  
Hungarian Academy of Sciences\*

<sup>2</sup> Budapest University of Technology and Economics  
fd@cs.bme.hu, bracz+c31@math.bme.hu

**Abstract** Several iterative hyperlink-based similarity measures were published to express the similarity of web pages. However, it usually seems hopeless to evaluate complex similarity functions over large repositories containing hundreds of millions of pages. We introduce scalable algorithms computing SimRank scores, which express the contextual similarities of pages based on the hyperlink structure. The proposed methods scale well to large repositories, fulfilling strict requirements about computational complexity. The algorithms were tested on a set of ten million pages, but parallelization techniques make it possible to compute the SimRank scores even for the entire web with over 4 billion pages. The key idea is that randomized Monte Carlo methods combined with indexing techniques yield a scalable approximation of SimRank.

## 1 Introduction

Calculating the similarity of two web pages is a fundamental building block of webmining algorithms. For example, clustering algorithms classify the pages based on some similarity function; related queries of web search engines enumerate the pages most similar to some source page. Both applications require fast computable similarity values that resemble the human users' similarity preferences.

Successful application areas of link analysis include ranking [17,13] and clustering [1], as well as similarity search. The similarity scores of this paper are also evaluated solely from the hyperlink structure of web pages. Link-based methods are efficient since the link structure forms a homogenous language-independent dataset. Recent link-based similarity algorithms iterate some equations expressing basic intuitions about similarity preferences. SimRank recursively refines the cocitation measure [12]. The algorithm of [8] follows the philosophy of HITS [13] claiming that related pages form a dense bipartite subgraph within the link structure. Additional similarity metrics for the web graph are presented in [9,14], for example using network flows in local subgraphs. Several more metrics based on cocitation are evaluated in [7] that are not complex enough (e.g. can be spammed or utilize only the 1–2 neighborhood of the queried pages).

Unfortunately, the available implementations of complex link-based algorithms (SimRank, HITS-based and network flows) do not scale well to massive datasets. All of them

---

\* Research was supported by grants OTKA T 42559 and T 42706 of the Hungarian National Science Fund, and from NKFP-2/0017/2002 project Data Riddle.

require random access to the web graph (link structure), while the size of a web graph can far exceed the available main memory. Additionally, the memory and time requirements of SimRank are quadratic in the number of pages, which is unrealistic over large repositories. In contrast, PageRank requires only linear time iterations, and its external memory issues are solved in [6].

This paper introduces novel algorithms computing SimRank scores that scale well to large repositories. For example, on a collection of 10M pages the basic SimRank algorithm would require up to  $10^{14}$  steps and main memory to compute the SimRank scores for all pairs of pages. On the other hand, our main algorithm precomputes roughly 100 sets of fingerprints with each set of size 800MB, and the SimRank of two pages can be computed with two disk seeks later in each set. We also give heuristics to reduce the total size of the database by a factor of 10 in a distributed system.

Our solution simulates random navigational pathways and stores them as *fingerprints* in the index database. The expected similarity of two fingerprints equals to the SimRank scores that can be derived from the random surfer pair model of [12]. Thus, our randomized algorithm approximates SimRank scores from several independent fingerprints. For text-based similarity search fingerprints and Monte Carlo methods were already successfully applied, see [4,11].

This paper is organized as follows. In Section 2 and 3 we present the main algorithm and the heuristics to compact the index. In Section 4 we formally analyze the error of approximation due to randomization. In Section 5, we show that indexing and query evaluation can be very effectively parallelized to deal with extremely large repositories. Our algorithms were tested on a web crawl with 10M pages, and the method of [10] is applied to numerically compute the overall quality of our output SimRank scores, as described in Section 6.

## 1.1 Algorithms scaling to the whole Web Graph

In this section we declare the strict computational complexity, memory usage and parallelization requirements for our SimRank algorithms. An algorithm that fulfills such requirements can be scaled on a distributed system even to a graph with billions of vertices like the whole webgraph. Similar requirements appear in [18], furthermore these characterize the computational environment in which Google’s PageRank scores are computed [3,17].

The following components of similarity search need to be discussed:

- **Indexing:** Preprocess the web graph and compute a database to support fast queries later. We will refer to this as the *index* database, following the conventions of text based information retrieval [20].
- **Similarity query:** calculate  $\text{sim}(v, w)$  for the webpages  $v$  and  $w$  using the index database.
- **Top query:** for a given query page  $v$  enumerate the  $k$  most similar pages to  $v$ , i.e., the pages with largest  $\text{sim}(v, \cdot)$  scores, where  $k \approx 100 - 500$ .

The main contributions of this paper are the algorithms of Sections 2 and 3 fulfilling the following requirements:

- **Computational complexity:** once the index database is ready the time needed to evaluate a similarity query is independent of the graph’s size; the time needed for a top query is proportional to the size of the result list. Indexing is a linear time algorithm in both the number of vertices and the number of edges of the web graph.
- **Memory usage:** Indexing uses no more than  $\mathcal{O}(V)$  main memory, where  $V$  denotes the number of vertices. The whole hyperlink structure resides in the external memory (disk), so the edges can only be accessed sequentially as a stream. The process of reading the stream of edges will be referred to as an *edge-scan*. Our main goal will be to minimize the required number of edge-scans. This memory access scheme is usually referred to as a *semi-external memory graph algorithm* [15].
- **Database access:** Query performs a constant number of index database accesses.
- **Parallelization:** Both indexing and querying can be implemented to utilize the computing power and storage capacity of tens to thousands of large to medium sized servers, and achieve response times suitable for (human) real-time usage.

## 1.2 SimRank

In this part we briefly recall the definition of SimRank and the random surfer pair model from [12]. According to the recursive intuition behind SimRank “two web pages are similar, if they are referenced by similar pages.” This is formalized by the following *SimRank equations*. If  $v = w$ , then  $\text{sim}(v, w) = 1$ ; otherwise

$$\text{sim}(v, w) = \frac{c}{|I(v)| |I(w)|} \sum_{v' \in I(v)} \sum_{w' \in I(w)} \text{sim}(v', w'),$$

where  $I(x)$  denotes the set of pages (vertices) linking to  $x$ , and  $c \in (0, 1)$  the so called *decay factor* is a constant. If either  $I(v)$  or  $I(w)$  is empty, then  $\text{sim}(v, w)$  is 0 by definition. For a given directed graph and decay factor  $c$  the  $\text{sim}(\cdot, \cdot)$  *SimRank* scores are defined as the solution of the above system of equations for all  $v, w$  vertices.

Besides the recursive intuition and equations SimRank can be also interpreted by the *random surfer pair model*. This interpretation will play a crucial role throughout our paper, so we recall this approach from [12], too. The random surfer pair model assumes that *a pair of vertices (web pages) is similar, if two random walks starting from the corresponding vertices and following the links backwards (are expected to) meet within a few steps*. Based on this a rich family of similarity functions can be defined by varying the weight that is given if the random walks first meet after exactly  $X$  backward steps. The next theorem of [12] states that by choosing weights exponentially decreasing with  $X$  the obtained similarity scores are equivalent to the SimRank scores.

**Theorem 1.** *Let  $X_{v,w}$  denote the first meeting time of two independent uniform random walks started from vertices  $v$  and  $w$  on the transposed webgraph. Then the  $\mathbb{E}(c^{X_{v,w}})$  values solve the SimRank equations with decay factor  $c$ .*

## 2 Our Algorithm

Our novel approach calculates the SimRank values based on Theorem 1 using a Monte Carlo method: to evaluate  $\text{sim}(v, w)$  we will simulate  $N$  independent pair of random

**Algorithm 1** Indexing (external memory method)

$N$ =number of fingerprints,  $L$ =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.

```

1: for  $i := 1$  to  $N$  do
2:   for all  $j$  vertices of the web graph do /*start a path from each vertex*/
3:     PathEnd[ $j$ ] :=  $j$ 
4:   for  $k:=1$  to  $L$  do
5:     NextIn[] := GenRndInEdges()
6:     for all  $j$  vertices of the web graph do /*prolong the current paths with the new in-edges*/
7:       PathEnd[ $j$ ] := NextIn[PathEnd[ $j$ ]]
8:     save PathEnd[] as Path $_k$ []
9:   merge the Path $_k$ [ $j$ ] arrays for  $k = 1..L$  into Fingerprint[ $i$ ][ $j$ ][ $k$ ].

```

walks starting from these pages on the transposed graph, and estimate SimRank by the average of the exponentially weighted meeting time. A naive implementation would just simulate random walks upon a query on  $\text{sim}(v, w)$ . Unfortunately this requires random access to the edge set of the graph which is prohibited by the memory requirements.

## 2.1 Fingerprints and Indexing

We will pre-compute  $N$  random walks for each vertex  $v$ , and store these paths in an index. For practical purposes we truncate the random walks at a finite length  $L$ .<sup>3</sup> Upon a query  $\text{sim}(v, w)$  we load the two sets of paths from the index database and compute the average weighted meeting time. This requires two database accesses.

A random walk of length  $L$  starting from a vertex  $v$  and following the hyperlinks backwards will be referred to as a *fingerprint* of  $v$ . This captures the intuition that fingerprints depict the surroundings of vertices in the web graph.  $\text{Fingerprint}[i][j][k]$  denotes the  $k^{\text{th}}$  node of the  $i^{\text{th}}$  fingerprint of vertex  $j$ , where  $i = 1..N$ ,  $j = 1..V$  and  $k = 1..L$ .

## 2.2 External memory indexing algorithm

Unfortunately the naive indexing approach generates fingerprints using random access to the edges of the graph, which reside in external memory. So instead of simulating random walks for each vertex separately, we simulate them together. Assume we have already computed the first  $k$  elements of all fingerprint paths. Then with one iteration we calculate the  $(k + 1)^{\text{th}}$  elements of all fingerprints. To accomplish this, we generate a random in-edge for each vertex  $v$  of the graph, and prolong those partial fingerprints with this edge that have  $v$  as their last element.<sup>4</sup> (See Algorithm 1.)

*Complexity and Space Requirements.* Once we have the random in-edges, the indexing algorithm is linear in  $V$ , the number of vertices of the graph, requiring  $V$  cells of memory. Furthermore, the required  $N \cdot L$  sets of random in-edges can be generated either

<sup>3</sup> This is equivalent to iterating the SimRank equations  $L$  times. As observed in [12] 5–10 iterations corresponding to  $L \approx 5 - 10$  is sufficient for the applications.

<sup>4</sup> This results in a slight distortion of the original concept, namely that the random walks of different vertices are not totally independent. Fortunately this does not cause a problem since they are independent until the first meeting point, but are *coupled* to stick together after that.

with  $\frac{N \cdot L}{M/V}$  edge-scans (where  $M$  denotes the available memory) over the edges as a data stream, or by  $\mathcal{O}(\log(E/M)) + 1$  edge-scans by sorting the edges with an external memory sorting algorithm and generating all sets of InEdges with one additional scan.

The preparation and evaluation of top queries employ standard methods of inverted indices and are not discussed here, see for example [20].

### 3 Reducing the Index Size

The algorithm described in this section uses significantly smaller sized index. In addition, the response time of top queries is also reduced by replacing frequent database accesses with more intensive main memory usage. Unfortunately, the applied heuristics compute similarity scores slightly differing from the original concept of SimRank; the effects of the distortion will be analyzed experimentally in Section 6.1.

A subgraph  $H$  with exactly one random edge linking to each vertex will be referred to as a *compacted fingerprint*, while the  $L$  sized path in  $H$  starting from  $v$  and following the links backwards<sup>5</sup> will be treated as a *fingerprint of vertex  $v$* . The *compacted SimRank* score  $\text{sim}(v, w)$  is calculated as the exponentially decreasing-weighted first meeting time of the fingerprints of  $v$  and  $w$ , analogously to the previous section.

Notice that the compacted fingerprint  $H$  contains the fingerprints for all vertices, and  $H$  has no more than  $V$  edges, where  $V$  denotes the number of vertices in the original graph. Storing these compacted fingerprints reduces the index size by a factor of  $L$ . In addition, the  $H$  graph with  $V$  edges still fits into the main memory according to the requirements of Section 1.1. This enables us to run more complex graph algorithms on  $H$ , for example the top query will be calculated by a (modified) breadth first search.

#### 3.1 Indexing and Query Evaluation

The indexing method generates a compacted fingerprint  $H$  for each simulation with the same GenRndInEdges() function as in the previous section. The main difference is, that the algorithm does not save the actual paths for each vertex, but it saves the generating InEdges[] array as IndexIn[i][] and the respective graph (i.e. the edges of subgraph  $H$ ) transposed as IndexOut[i][].

To evaluate  $\text{sim}(v, w)$  queries we simulate the random walks from  $v$  and  $w$  respectively by following the edges pointed by IndexIn[]. This requires random access to IndexIn[], which still fits into the main memory as the array has size  $V$ . The top query for vertex  $q$  will be evaluated as depicted on Algorithm 2. Basically for each independent simulation, we have to generate all the paths that intersect the path of vertex  $q$ . So first we calculate the fingerprint path of  $q$  using the IndexIn[] array. Then for each node of that path we enumerate the outgoing paths using the transposed fingerprint IndexOut[]. For the  $k^{\text{th}}$  element of  $q$ 's path we take exactly  $k$  steps forward in all possible directions pointed by IndexOut[]. This will generate us all the vertices with positive similarity to  $q$ , and thus has a time complexity linear in the output size.

<sup>5</sup> This path is unique as each vertex is linked to by one (or zero) edge in  $H$ .

**Algorithm 2** Top query (compacted SimRank)

---

Input: the query page  $q$ ,  $L$ =length of path,  $N$ =number of fingerprints,  $c$ =decay factor.  
Output: list of  $p$  pages with positive similarity to  $q$ , ordered by  $\text{sim}(p, q)$ .  
SimilarPages is an initially empty (hashed) associative container that is indexed by the vertices of the web graph and holds the similarities to  $q$ . Non-existent entries return 0.

```

1: for  $i := 1$  to  $N$  do
2:   Ancestor :=  $q$  /*This variable holds the  $k^{\text{th}}$  element of  $q$ 's fingerprint ( $k = 0$  currently). Wee look for other paths that lead into this point in  $k$  steps.*/
3:   for  $k := 1$  to  $L$  do
4:     Ancestor := IndexIn[ $i$ ][Ancestor]/*One step backward*/
5:      $D := \{\text{Ancestor}\}$ /*The descendants of Ancestor*/
6:     for  $l := 1$  to  $k$  do /* $k$  steps forward*/
7:        $D := \cup_{j \in D} \text{IndexOut}[i][j]$ 
8:     for all  $j \in D$  do
9:       SimilarPages[ $j$ ] := SimilarPages[ $j$ ] +  $\frac{c^k}{N}$ 
10: return SimilarPages sorted by value

```

---

Both similarity and top query algorithms uses random access to  $H$  for each simulation. With the parallelization techniques discussed later in Section 5, it is possible to avoid loading the compacted fingerprints one-by-one to the main memory upon a query.

## 4 How Many Fingerprints are Needed?

In this section we will try to estimate how well our methods approximate the actual SimRank values and how many fingerprints are sufficient for adequate results in a real-world application. It is obvious by Theorem 1, that for a sufficiently large number of fingerprints  $N$ , the results returned by our randomized methods  $\widehat{\text{sim}}(u, v)$  converge to  $\text{sim}(u, v)$ , the actual SimRank. However, storage requirement and run time are linear in  $N$ , so we want to minimize it. In this section we will show that about 100 fingerprints are sufficient for applications.

For a given query  $\text{sim}(u, v)$  the fingerprints give  $N$  independent identically distributed random variables  $X_1, \dots, X_N$ , each of which has an expected value of  $\text{sim}(u, v)$ . By the central limit theorem—which is commonly applied from  $N = 30$ —the result we calculate  $\widehat{\text{sim}}(u, v) = \frac{1}{N} \sum_{i=1}^N X_i$  is of normal distribution around the expected value  $\text{sim}(u, v)$  with standard deviation  $\frac{1}{\sqrt{N}} \mathbb{D}(X)$ . This  $\mathcal{O}(\frac{1}{\sqrt{N}})$  convergence is unfortunately not enough for our purposes.

Instead, note that in implementations we do not require the actual values to be very close to SimRank, but to have the ordering defined by  $\widehat{\text{sim}}$  on a query result set follow closely the actual similarities. In this case we have much better results:

**Theorem 2.** *The probability of interchanging two documents in a query result list converges to 0 exponentially, i.e. if  $\text{sim}(u, v) > \text{sim}(u, w)$  then  $\Pr\{\widehat{\text{sim}}(u, v) < \widehat{\text{sim}}(u, w)\} \rightarrow 0$  exponentially in  $N$ .*

Analyzing the proof (which unfortunately had to be omitted due to space limitations) suggests that  $N = 100$  gives more than adequate precision for a search engine. If the result list is ordered by some other function (for example PageRank), then even less fingerprints are enough, to distinguish between unrelated pages, low similarity and top similarity pages.

## 5 Scaling to the Whole Web

In this section we will shortly summarize the possibilities for parallelizing the algorithms described earlier. Our aim is to apply the methods for graphs as large as the whole web graph, i.e. a few billions of vertices. The platform considered here for such applications is a very large cluster of PC category machines [2], interconnected by some form of network. We will consider requirements for minimum operation, load balancing and fault tolerance to show that our methods are applicable even in the largest publicly available search services.

The key observation is the possibility of distributing the independent fingerprints to  $N$  different machines [19]. We will call this *coarse horizontal parallelization*. In this case the index builds can run in parallel with the necessary edge-scans distributed as broadcast data over the cluster. Each computer uses its own random number generator to calculate a different and independent set of fingerprints. The resulting indices are stored only locally. The query to evaluate will also be broadcasted to all the participating machines, which return their individual result list for merging by the front-end server. The network transfer required is proportional to the length of the result list (the number of pages with positive similarity to the query page) even with the most naive approach. Alternatively, a more advanced distributed top query method can be used for merging the result sets with much less network traffic, or even to permit partial evaluation at the index servers. Distributed top queries were deeply studied recently, see for example [5].

This utilizes at most  $N$  machines, each of which has enough memory for  $V$  cells for indexing, or query with compacted SimRank. For query with the original SimRank the only resource required is database accesses (disk seeks).

Thanks to the robustness of Monte Carlo methods this method offers intrinsic load balancing and fault tolerance support. Assume we have more than  $N$  machines, and each of them has got an index. For an adequate precision result it is enough to ask *any*  $N$  machines and merge the result list. This offers nice and smooth load balancing support for an arbitrary number of index servers. As for fault tolerance, skipping one fingerprint, and merging  $N - 1$  result lists instead of  $N$  does not change the result list significantly, thus *failed servers do not influence proper operation*. This enables a very effective combination of load balancing and fault tolerance: if the query load is larger than the available number of machines can serve, then the number of fingerprints used for a single query is decreased automatically, thus loosing some precision but maintaining adequate response times. Or in case of underload, precision can be increased for free to achieve better results.

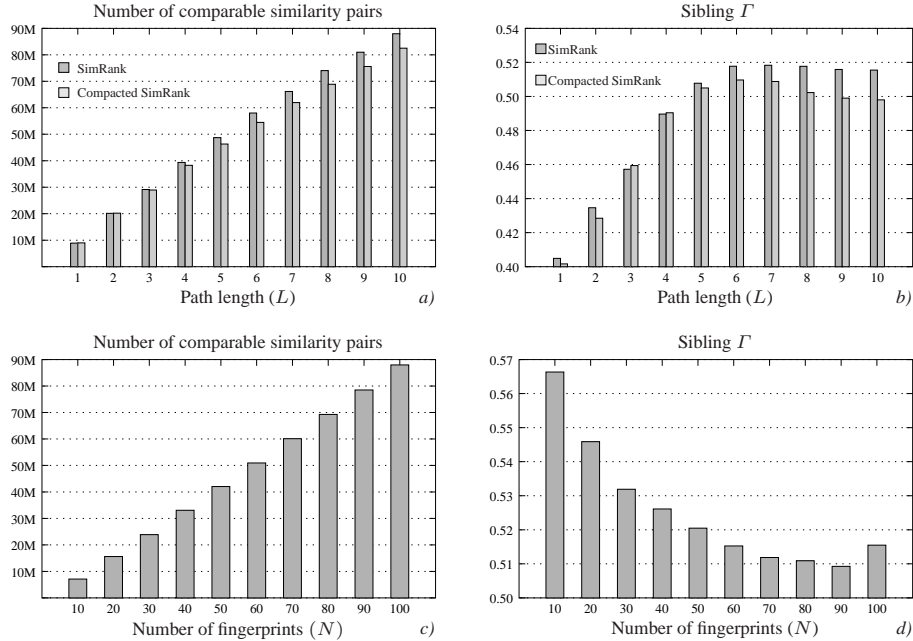
## 6 Experiments

In our experiments the value *sibling  $\Gamma$*  [10] measures the quality of the similarity ranking by comparing it to a ground truth similarity ordering extracted from the Open Directory Project (ODP, [16]) data, a hierarchical collection of webpages. The category tree of ODP provides ground truth *similarity pairs* by claiming that  $\text{sim}(u, v) < \text{sim}(u, w)$  should hold for certain ODP pages  $u$ ,  $v$ , and  $w$ . A similarity pair will be referred to as *comparable* if both  $\text{sim}(u, v)$  and  $\text{sim}(u, w)$  are larger than a minimal similarity threshold, which was set to zero in our experiments. To evaluate sibling  $\Gamma$  for SimRank scores

we need to check if the above inequality holds for the comparable similarity pairs or not. The resulting  $\Gamma \in (-1, 1)$  value is 1, if all the above inequalities hold for the SimRank scores; while  $\Gamma = -1$ , if all pairs are ordered reversely. For our algorithms  $\Gamma \approx 0.5$ , implying that a pair of comparable similarity pairs chosen at random is ordered correctly by our algorithm with probability over 0.75. Notice that high  $\Gamma$  quality can be achieved even with very few comparable pairs. Therefore we decided to measure the number of comparable similarity pairs in addition to sibling  $\Gamma$ . See [10] for the further details about sibling  $\Gamma$  measure.

Evaluating  $\Gamma$  statistics requires a webgraph containing ODP pages with rich hyperlink structure for the SimRank computation. Starting from ODP pages of categories `Computers` and `Science`, our repository was collected by a crawler restricted to follow links within the sites of ODP pages in November, 2003. We obtained a webgraph with  $V = 11\text{M}$  vertices,  $E = 150\text{M}$  links and 161K ODP pages. Our experiments were performed on a machine with an 1.4GHz Intel Xeon processor, 5Gbyte main memory and Linux OS. The fingerprints were computed in 40 minutes, the index was truncated to the 161K ODP pages and had a total size of 1.3Gbyte (including the inverted index).

## 6.1 Experimental results



**Figure 1.** Measuring the quality of SimRank scores with different parameter settings.

Our numerical experiments address the problem of finding the optimal settings over the 3-dimensional parameter space: path length  $L$ , decay factor  $c$  and number of fin-

gerprints  $N$ . To perform a full optimization was beyond the scope of our computing capacity, so we fixed the default values  $L = 10$ ,  $N = 100$  and  $c = 0.65$ ; then the parameters were analyzed independently by varying only one parameter at a time, see Fig. 1.

The left hand side bars of the clustered bar charts of parts *a)* and *b)* on Fig. 1 show that both the number of comparable pairs and the quality increase with  $L$ . Recall that the SimRank scores approximated with path length  $L$  correspond to  $L$  iteration of the recursive equations. Thus, the growing quality testifies the recursive intuition of SimRank. However, more recursion slightly reduces the quality among the pages with low SimRank scores, as  $\Gamma$  was slightly decreasing after  $L = 7$ .

The right hand side bars of parts *a)* and *b)* show the quality loss for the compacted SimRank scores compared to SimRank. According to the charts, the longer the paths are the more quality loss occurs on average. However, in a real application it may be reasonable to trade the quality difference to the advantage of the reduced index database compacted to  $L$  times smaller.

The amount of comparable pairs in figure *c)* increases linearly with  $N$ . The reason is simple: more fingerprint paths cross each other, and thus make more similarity pairs comparable. Part *d)* generally shows quality fall in sibling  $\Gamma$ . We argue that in case of  $N = 10$  the pairs with positive approximated SimRank scores have relatively large SimRank, and such pairs approximate the ground truth similarity better than those having smaller SimRank values.

Due to space limitations the effects of decay factor is omitted from the diagrams. We experienced that the decay factor  $c$  is inversely proportional to sibling  $\Gamma$ , while the number of similarity pairs remains unchanged. When  $c \rightarrow 0$  and  $N$  is fixed, the weight given for a shorter meeting time will largely supercede the weight given for longer meeting times. Our measurements imply that shorter paths should generally be given higher weight than the sum of arbitrary number of longer paths.

## 7 Conclusion and Future Work

We showed a method for computing the SimRank similarity scores for web pages. SimRank is purely based on the link structure of the web graph, just as PageRank, but has previously had algorithms with quadratic memory requirement and running time. Our method uses linear time randomized fingerprint-based calculations and can be implemented on  $V$  pages using  $V$  cells of memory, which can be further reduced to constant by using external memory sorts. When parallelized on a cluster of machines this method can cope with the whole webgraph making it suitable for the largest scale commercial web search engines, featuring load balance and fault tolerance intrinsically. Furthermore, we presented an approximation with lower precision, but drastically reduced index database sizes that can be entirely served from main memory in case of extreme workloads.

We strongly believe that fingerprint-based Monte Carlo methods can have further applications in web-mining. For example, we plan to construct scalable approximation methods for clustering web pages and tracing web communities.

## 8 Acknowledgement

We wish to thank Katalin Friedl, András Benczúr, Tamás Sarlós for valuable discussions and comments; Máté Uher for crawling the test repository of web pages.

## References

1. P. Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002.
2. E. Brewer. Lessons from giant-scale services.
3. S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.
4. A. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, page 21. IEEE Computer Society, 1997.
5. N. Bruno, L. Gravano, and A. Marian. Evaluating top-k queries over web-accessible databases. In *ICDE*, 2002.
6. Y. Y. Chen, Q. Gan, and T. Suel. I/O-efficient techniques for computing PageRank. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 549–557. ACM Press, 2002.
7. M. Cristo, P. Calado, E.S. de Moura, N. Ziviani, and B. Ribeiro-Neto. Link information as a similarity measure in web classification. In *String Processing and Information Retrieval*, pages 43–55. Springer LNCS 2857, 2003.
8. 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.
9. G. Flake, S. Lawrence, C. L. Giles, and F. Coetzee. Self-organization of the web and identification of communities. *IEEE Computer*, 35(3):66–71, 2002.
10. T. H. Haveliwala, A. Gionis, D. Klein, and P. Indyk. Evaluating strategies for similarity search on the web. In *Proceedings of the 11th World Wide Web Conference (WWW)*, pages 432–442. ACM Press, 2002.
11. N. Heintze. Scalable document fingerprinting. In *1996 USENIX Workshop on Electronic Commerce*, November 1996.
12. G. Jeh and J. Widom. SimRank: A measure of structural-context similarity. In *Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2002.
13. J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.
14. W. Lu, J. Janssen, E. Milios, and N. Japkowicz. Node similarity in networked information spaces. In *Proceedings of the 2001 conference of the Centre for Advanced Studies on Collaborative research*, page 11. IBM Press, 2001.
15. U. Meyer, P. Sanders, and J. Sibeyn. *Algorithms for Memory Hierarchies, Advanced Lectures*. LNCS, Springer, 2003.
16. Open Directory Project (ODP). <http://www.dmoz.org>.
17. 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.
18. C. R. Palmer, P. B. Gibbons, and C. Faloutsos. ANF: a fast and scalable tool for data mining in massive graphs. In *Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 81–90. ACM Press, 2002.
19. J. S. Rosenthal. Parallel computing and Monte Carlo algorithms. *Far East J. Theor. Stat.*, 4:207–236, 2000.
20. I. H. Witten, A. Moffat, and T. C. Bell. *Managing gigabytes (2nd ed.): Compressing and indexing documents and images*. Morgan Kaufmann Publishers Inc., 1999.