

Towards Scaling Fully Personalized PageRank: algorithms, lower bounds and experiments^{*}

Dániel Fogaras^{1,2}, Balázs Rácz^{1,2}, Károly Csalogány^{1,3}, and Tamás Sarlós^{1,3}

¹ Computer and Automation Research Institute of the
Hungarian Academy of Sciences

² Budapest University of Technology and Economics

³ Eötvös University, Budapest

fd@cs.bme.hu, bracz+p91@math.bme.hu, {cskarezs, stamas}@ilab.sztaki.hu

Abstract Personalized PageRank expresses link-based page quality around user-selected pages in a similar way as PageRank expresses quality over the entire Web. Existing personalized PageRank algorithms can however serve on-line queries only for a restricted choice of pages. In this paper we achieve full personalization by a novel algorithm that precomputes a compact database, and using this database it can serve on-line responses to arbitrary user selected personalization. The algorithm uses simulated random walks; we prove that for a fixed error probability, the size of our database is linear in the number of web pages. We justify our estimation approach by asymptotic worst-case lower bounds: we show that on some sets of graphs exact personalized PageRank values can only be obtained from a database of size quadratic in the number of vertices. Furthermore, we evaluate the precision of approximation experimentally on the Stanford WebBase graph.

1 Introduction

The idea of topic sensitive or personalized ranking appears since the beginning of the success story of Google's PageRank [7,33] and other hyperlink-based quality measures [30,6]. Topic sensitivity is either achieved by precomputing modified measures over the entire Web [19] or by ranking the neighborhood of pages containing the query word [30]. These methods however work only for restricted cases or when the entire hyperlink structure fits into the main memory.

In this paper we address the computational issues [19,26] of personalized PageRank [33]. Just as all hyperlink based ranking methods, PageRank is based on the assumption that *the existence of a hyperlink $u \rightarrow v$ implies that page u votes for the quality of v* . Personalized PageRank (PPR) enters user preferences by assigning more importance to edges in the neighborhood of certain pages at the user's selection. Unfortunately the naive computation of PPR requires a power iteration algorithm over the entire web graph, making the procedure infeasible for an on-line query response service.

^{*} Research was supported by grants OTKA T 42481, T 42559, T 42706 and T 44733 of the Hungarian National Science Fund, and NKFP-2/0017/2002 project Data Riddle.

Earlier personalized PageRank (PPR) algorithms restricted personalization to a few topics [19], a subset of popular pages [26] or to hosts [27]; see [21] for an analytical comparison of these methods. The state of the art Hub Decomposition algorithm [26] can answer queries for up to some 100,000 personalization pages, an amount relatively small even compared to the number of categories in the Open Directory Project [1].

In contrast to earlier PPR algorithms, we achieve *full personalization*: our method enables on-line serving of personalization queries for *any* set of pages. We introduce a novel, scalable Monte Carlo algorithm that precomputes a compact database. As described in Section 2, the precomputation uses simulated random walks, and stores the ending vertices of the walks in the database. PPR is estimated on-line with a few database accesses.

The price that we pay for full personalization is that our algorithm is randomized and less precise than power-iteration-like methods; the formal analysis of the error probability is discussed in Section 3. We theoretically and experimentally show that we give sufficient information for all possible personalization pages while adhering to the strong implementation requirements of a large-scale web search engine.

According to Section 4, some approximation seems to be unavoidable since the exact personalization requires a database as large as $\Omega(V^2)$ bits in worst case over graphs with V vertices. Though no worst case theorem applies to the webgraph or one particular graph, the theorems show the nonexistence of a general exact algorithm that computes a linear sized database on any graph. To achieve full personalization in future research, one must hence either exploit special features of the webgraph or relax the exact problem to an approximate one as in our scenario. Of independent interest is another consequence of our lower bounds that there is indeed a large amount of information in personalized PageRank vectors since, unlike uniform PageRank, it can hold information of size quadratic in the number of vertices.

In Section 5 we experimentally analyze the precision of approximation on the Stanford WebBase graph and conclude that our randomized approximation method provides sufficiently good approximation for the top personalized PageRank scores.

Though our approach might give results of inadequate precision in certain cases (for example for pages with large neighborhood), the available personalization algorithms can be combined to resolve these issues. For example we can precompute personalization vectors for certain topics by topic-sensitive PR [19], for popular pages with large neighborhoods by the hub skeleton algorithm [26]), and use our method for those millions of pages not covered so far. This combination gives adequate precision for most queries with large flexibility for personalization.

1.1 Related Results

We compare our method with known personalized PageRank approaches as listed in Table 1 to conclude that our algorithm is the first that can handle on-line per-

sonalization on arbitrary pages. Earlier methods in contrast either restrict personalization or perform non-scalable computational steps such as power-iteration in query time or quadratic disk usage during the precomputation phase. The only drawback of our algorithm compared to previous ones is that its approximation ratio is somewhat worse than that of the power iteration methods.

The first known algorithm [33] (Naive in Table 1) simply takes the personalization vector as input and performs power iteration at query time. This approach is clearly infeasible for on-line queries. One may precompute the power iterations for a well selected set of personalization vectors as in the Topic Sensitive PageRank [19]; however full personalization in this case requires $t = V$ precomputed vectors yielding a database of size V^2 for V web pages. The current size $V \approx 10^9 - 10^{10}$ hence makes full personalization infeasible.

The third algorithm of Table 1, BlockRank [27] restricts personalization to hosts. While the algorithm is attractive in that the choice of personalization is fairly general, a reduced number of power iterations still need to be performed at query time that makes the algorithm infeasible for on-line queries.

The remarkable Hub Decomposition algorithm [26] restricts the choice of personalization to a set H of top ranked pages. Full personalization however requires H to be equal to the set of all pages, thus V^2 space is required again. The algorithm can be extended by the Web Skeleton [26] to lower estimate the personalized PageRank vector of arbitrary pages by taking into account only the paths that goes through the set H . Unfortunately, if H does not overlap the few-step neighborhood of a page, then the lower estimation provides poor approximation for the personalized PageRank scores.

The Dynamic Programming approach [26] provides full personalization by precomputing and storing sparse approximate personalized PageRank vectors. The key idea is that in a k -step approximation only vertices within distance k have nonzero value. However the rapid expansion of the k -neighborhoods increases disk requirement close to V^2 after a few iterations that limits the usability of this approach. Furthermore, a possible external memory implementation would require significant additional disk space. The space requirements of Dynamic Programming for a single vertex is given by the average neighborhood size $\text{Neighb}(k)$ within distance k as seen in Fig. 1. The average size of the sparse vectors exceeds 1000 after $k \geq 4$ iterations, and on average 24% of all vertices are reached within $k = 15$ steps⁴. For example the disk requirement for $k = 10$ iterations is at least $\text{Neighb}(k) \cdot V = 1,075,740 \cdot 80\text{M} \approx 344$ Terabytes. Note that the best upper bound of the approximation is still $(1 - c)^{10} = 0.85^{10} \approx 0.20$ measured by the L_1 -norm.

⁴ The neighborhood function was computed by combining the size estimation method of [10] with our external memory algorithm discussed in [17].

Method	Personalization	Limits of scalability	Postitive aspects	Negative aspects
Naive [33]	any page	power iteration in query-time		infeasible to serve on-line personalization
Topic-Sensitive PageRank [19]	restricted to linear combination of t topics, e.g. $t = 16$	$t \cdot V$ disk space required	distributed computing	
BlockRank [27]	restricted to personalize on hosts	power iteration in query-time	reduced number of power iterations, distributed computing	infeasible to serve on-line personalization
Hub Decomposition [26]	restricted to personalize on the top H ranked pages, practically $H \leq 100K$	H^2 disk space required, H partial vectors aggregated in query time	compact encoding of H personalized PR vectors	
Basic Dynamic Programming [26]	any page	$V \cdot \text{Neighb}(k)$ disk space required for k iterations, where $\text{Neighb}(k)$ grows fast in k		infeasible to perform more than $k = 3, 4$ iterations within reasonable disk size
Fingerprint (this paper)	any page	no limitation	linear-size ($N \cdot V$) disk required, distributed computation	lower precision approximation

Table 1. Analytical comparison of personalized PageRank algorithms. V denotes the number of all pages.

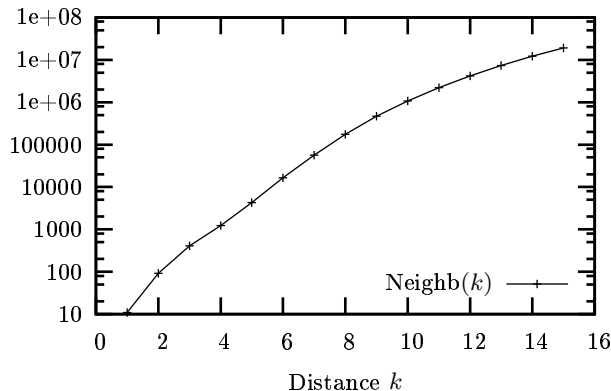


Figure 1. The neighborhood function measured on the Stanford WebBase graph of 80M pages.

We believe that the basic Dynamic Programming could be extended with some pruning strategy that eliminates some of the non-zero entries from the approximation vectors. However, it seems difficult to upper bound the error caused by the pruning steps, since the small error caused by a pruning step is distributed to many other approximation vectors in subsequent steps. Another drawback of the pruning strategy is that selecting the top ranks after each iteration requires extra computational efforts such as keeping the intermediate results in priority queues. In contrast, our fingerprint based method tends to eliminate low ranks inherently, and the amount of error caused by the limited storage capacity can be upper bounded formally.

Now, we briefly review some algorithms that solve the scalability issue by fingerprinting or sampling for applications that are different from personalized web search. For example, [34] applies probabilistic counting to estimate the neighborhood function of the Internet graph, [10] estimates the size of transitive closure for massive graphs occurring in databases, and [18,17] approximates link-based similarity scores by fingerprints. Apart from graph algorithms, [8] estimates the resemblance and containment of textual documents with fingerprinting.

Random walks were used before to compute various web statistics, mostly focused on sampling the web (uniformly or according to static PR) [23,36,2,22], but also for calculating page decay [3] and similarity values [18,17].

The lower bounds of Section 4 show that precise PPR requires significantly larger database than Monte Carlo estimation does. Analogous results with similar communication complexity arguments were proved in [24] for the space complexity of several data stream graph algorithms.

1.2 Preliminaries.

In this section we introduce notation, recall definitions and basic facts about PageRank. Let \mathcal{V} denote the set of web pages, and $V = |\mathcal{V}|$ the number of pages.

The directed graph with vertex set \mathcal{V} and edges corresponding to the hyperlinks will be referred to as the *web graph*. Let A denote the adjacency matrix of the webgraph with normalized rows and $c \in (0, 1)$ the *teleportation probability*. In addition, let \mathbf{r} be the so called *preference vector* inducing a probability distribution over \mathcal{V} . *PageRank* vector \mathbf{p} is defined as the solution of the following equation [33]

$$\mathbf{p} = (1 - c) \cdot \mathbf{p}A + c \cdot \mathbf{r} .$$

If \mathbf{r} is uniform over \mathcal{V} , then \mathbf{p} is referred to as the *global PageRank vector*. For non-uniform \mathbf{r} the solution \mathbf{p} will be referred to as *personalized PageRank vector of \mathbf{r}* denoted by $\text{PPV}(\mathbf{r})$. The special case when for some page u the u^{th} coordinate of \mathbf{r} is 1 and all other coordinates are 0, the PPV will be referred to as the *individual PageRank vector of u* denoted by $\text{PPV}(u)$. We will also refer to this vector as the *personalized PageRank vector of u* . Furthermore the v^{th} coordinate of $\text{PPV}(u)$ will be denoted by $\text{PPV}(u, v)$.

Theorem 1 (Linearity, [19]). *For any preference vectors $\mathbf{r}_1, \mathbf{r}_2$, and positive constants α_1, α_2 with $\alpha_1 + \alpha_2 = 1$ the following equality holds:*

$$\text{PPV}(\alpha_1 \cdot \mathbf{r}_1 + \alpha_2 \cdot \mathbf{r}_2) = \alpha_1 \cdot \text{PPV}(\mathbf{r}_1) + \alpha_2 \cdot \text{PPV}(\mathbf{r}_2).$$

Linearity is a fundamental tool for scalable on-line personalization, since if PPV is available for some preference vectors, then PPV can be easily computed for any combination of the preference vectors. Particularly, for full personalization it suffices to compute individual $\text{PPV}(u)$ for all $u \in \mathcal{V}$, and the individual PPVs can be combined on-line for any small subset of pages. Therefore in the rest of this paper we investigate algorithms to make all individual PPVs available on-line.

The following statement will play a central role in our PPV estimations. The theorem provides an alternate probabilistic characterization of individual PageRank scores.⁵

Theorem 2 ([26,16]). *Suppose that a number L is chosen at random with probability $\Pr\{L = i\} = c(1-c)^i$ for $i = 0, 1, 2, \dots$. Consider a random walk starting from some page u and taking L steps. Then for the v^{th} coordinate $\text{PPV}(u, v)$ of vector $\text{PPV}(u)$*

$$\text{PPV}(u, v) = \Pr\{\text{the random walk ends at page } v\}.$$

2 Personalized PageRank algorithm

In this section we will present a new Monte-Carlo algorithm to compute approximate values of personalized PageRank utilizing the above probabilistic characterization of PPR. We will compute approximations of each of the PageRank vectors personalized on a single page, therefore by the linearity theorem we achieve full personalization.

⁵ Notice that this characterization slightly differs from the random surfer formulation [33] of PageRank.

Definition 1 (Fingerprint path). A fingerprint path of a vertex u is a random walk starting from u ; the length of the walk is of geometric distribution of parameter c , i.e., after each step the walk takes a further step with probability $1 - c$ and ends with probability c .

Definition 2 (Fingerprint). A fingerprint of a vertex u is the ending vertex of a fingerprint path of u .

By Theorem 2 the fingerprint of page u , as a random variable, has the distribution of the personalized PageRank vector of u . For each page u we will calculate N independent fingerprints by simulating N independent random walks starting from u and approximate $\text{PPV}(u)$ with the empirical distribution of the ending vertices of these random walks. These fingerprints will constitute the *index database*, thus the size of the database is $N \cdot V$. The output ranking will be computed at query time from the fingerprints of pages with positive personalization weights using the linearity theorem.

To increase the precision of the approximation of $\text{PPV}(u)$ we will use the fingerprints that were generated for the neighbors of u , as described in Section 2.3.

The challenging problem is how to scale the indexing, i.e., how to generate N independent random walks for each vertex of the web graph. We assume that the edge set can only be accessed as a data stream, sorted by the source pages, and we will count the database scans and total I/O size as the efficiency measure of our algorithms. Though with the latest compression techniques [5] the entire web graph may fit into main memory, we still have a significant computational overhead for decompression in case of random access. Under such assumption it is infeasible to generate the random walks one-by-one, as it would require random access to the edge-structure.

We will consider two computational environments here: a single computer with constant random access memory in case of the *external memory model*, and a *distributed system* with tens to thousands of medium capacity computers [11]. Both algorithms use similar techniques to the respective I/O efficient algorithms computing PageRank [9].

As the task is to generate N independent fingerprints, the single computer solution can be trivially parallelized to make use of a large cluster of machines, too. (Commercial web search engines have up to thousands of machines at their disposal.) Also, the distributed algorithm can be emulated on a single machine, which may be more efficient than the external memory approach depending on the graph structure.

2.1 External memory indexing

We will incrementally generate the entire set of random walks simultaneously. Assume that the first k vertices of all the random walks of length at least k are already generated. At any time it is enough to store the starting and the current vertices of the fingerprint path, as we will eventually drop all the nodes on the path except the starting and the ending nodes. Sort these pairs by the

Algorithm 1 Indexing (external memory method)

N is the required number of fingerprints for each vertex. The array Paths holds pairs of vertices (u, v) for each partial fingerprint in the calculation, interpreted as (PathStart, PathEnd). The teleportation probability of PPR is c . The array Fingerprint[u] stores the fingerprints computed for a vertex u .

```
for each web page  $u$  do
  for  $i := 1$  to  $N$  do
    append the pair  $(u, u)$  to array Paths /*start  $N$  fingerprint paths from node  $u$ :
    initially PathStart=PathEnd=  $u$ */
  Fingerprint[ $u$ ] :=  $\emptyset$ 
while Paths  $\neq \emptyset$  do
  sort Paths by PathEnd /*use an external memory sort*/
  for all  $(u, v)$  in Paths do /*simultaneous scan of the edge set and Paths*/
     $w :=$  a random out-neighbor of  $v$ 
    if random() $<c$  then /*with probability  $c$  this fingerprint path ends here*/
      add  $w$  to Fingerprint[ $u$ ]
      delete the current element  $(u, v)$  from Paths
    else /*with probability  $1 - c$  the path continues*/
      update the current element  $(u, v)$  of Paths to  $(u, w)$ 
```

ending vertices. Then by simultaneously scanning through the edge set and this sorted set we can have access to the neighborhoods of the current ending vertices. Thus each partial fingerprint path can be extended by a next vertex chosen from the out-neighbors of the ending vertex uniformly at random. For each partial fingerprint path we also toss a biased coin to determine if it has reached its final length with probability c or has to advance to the next round with probability $1 - c$. This algorithm is formalized as Algorithm 1.

The number of I/O operations the external memory sorting takes is $D \log_M D$, where D is the database size and M is the available main memory. Thus the expected I/O requirement of the sorting parts can be upper bounded by

$$\sum_{k=0}^{\infty} (1-c)^k NV \log_M((1-c)^k NV) = \frac{1}{c} NV \log_M(NV) - \Theta(NV)$$

using the fact that after k rounds the expected size of the Paths array is $(1 - c)^k NV$. Recall that V and N denote the numbers of vertices and fingerprints, respectively.

We need a sort on the whole index database to avoid random-access writes to the Fingerprint arrays. Also, upon updating the PathEnd variables we do not write the unsorted Paths array to disk, but pass it directly to the next sorting stage. Thus the total I/O is at most $\frac{1}{c} NV \log_M NV$ plus the necessary edge-scans.

Unfortunately this algorithm apparently requires as many edge-scans as the length of the longest fingerprint path, which can be very large: $\Pr\{\text{the longest fingerprint is shorter, than } L\} = (1 - (1 - c)^L)^{N \cdot V}$. Thus instead of scanning the edges in the final stages of the algorithm, we will change strategy when the

Paths array has become sufficiently small. Assume a partial fingerprint path has its current vertex at v . Then upon this condition the distribution of the end of this path is identical to the distribution of the end of any fingerprint of v . Thus to finish the partial fingerprint we can retrieve an already finished fingerprint of v . Although this decreases the number of available fingerprints for v , this results in only a very slight loss of precision.⁶

Another approach to this problem is to truncate the paths at a given length L and approximate the ending distribution with the static PageRank vector, as described in Section 2.3.

2.2 Distributed index computing

In the distributed computing model we will invert the previous approach, and instead of sorting the path ends to match the edge set we will partition the edge set of the graph in such a way that each participating computer can hold its part of the edges in main memory. So at any time if a partial fingerprint with current ending vertex v requires a random out-edge of v , it can ask the respective computer to generate one. This will require no disk access, only network transfer.

More precisely, each participating computer will have several queues holding (PathStart, PathEnd) pairs: one large input queue, and for each computer one small output queue preferably with the size of a network packet.

The computation starts with each computer filling their own input queue with N copies of the initial partial fingerprints (u, u) , for each vertex u belonging to the respective computer in the vertex partition.

Then in the input queue processing loop a participating computer takes the next input pair, generates a random out-edge from PathEnd, decides whether the fingerprint ends there, and if it does not, then places the pair in the output queue determined by the next vertex just generated. If an output queue reaches the size of a network packet's size, then it is flushed and transferred to the input queue of the destination computer. Notice that either we have to store the partition index for those v vertices that have edges pointing to in the current computer's graph, or $\text{part}(v)$ has to be computable from v , for example by renumbering the vertices according to the partition. For sake of simplicity the output queue management is omitted from the pseudo-code shown as Algorithm 2.

The total size of all the input and output queues equals the size of the Paths array in the previous approach after the respective number of iterations. The expected network transfer can be upper bounded by $\sum_{n=0}^{\infty} (1-c)^n NV = \frac{1}{c} NV$, if every fingerprint path needs to change computer in each step.

In case of the webgraph we can significantly reduce the above amount of network transfer with a suitable partition of the vertices. The key idea is to keep *each domain on a single computer*, since the majority of the links are intra-domain links as reported in [27,13].

⁶ Furthermore, we can be prepared for this event: the distribution of these v vertices will be close to the static PageRank vector, thus we can start with generating somewhat more fingerprints for the vertices with high PR values.

Algorithm 2 Indexing (distributed computing method)

The algorithm of one participating computer. Each computer is responsible for a part of the vertex set, keeping the out-edges of those vertices in main memory. For a vertex v , $\text{part}(v)$ is the index of the computer that has the out-edges of v . The queues hold pairs of vertices (u, v) , interpreted as $(\text{PathStart}, \text{PathEnd})$.

```
for  $u$  with  $\text{part}(u) = \text{current computer}$  do
  for  $i := 1$  to  $N$  do
    insert pair  $(u, u)$  into InQueue /*start  $N$  fingerprint paths from node  $u$ : initially  $\text{PathStart} = \text{PathEnd} = u$ */
  while at least one queue is not empty do /*some of the fingerprints are still being calculated*/
    get an element  $(u, v)$  from InQueue /*if empty, wait until an element arrives.*/
     $w :=$  a random out-neighbor of  $v$  /*prolong the path; we have the out-edges of  $v$  in memory*/
    if  $\text{random}() < c$  then /*with probability  $c$  this fingerprint path ends here*/
      add  $w$  to the fingerprints of  $u$ 
    else /*with probability  $1 - c$  the path continues*/
       $o := \text{part}(w)$  /*the index of the computer responsible for continuing the path*/
      insert pair  $(u, w)$  into the InQueue of computer  $o$ 
  transmit the finished fingerprints to the proper computers for collecting and sorting.
```

We can further extend the above heuristical partition to balance the computational and network load among the participating computers in the network. One should use a partition of the pages such that the *amount of global PageRank is distributed uniformly across the computers*. The reason is that the expected value of the total InQueue hits of a computer is proportional to the total PageRank score of vertices belonging to that computer. Thus when using such a partition, the total switching capacity of the network is challenged, not the capacity of the individual network links.

2.3 Query

The basic query algorithm is as follows: to calculate $\text{PPV}(u)$ we load the ending vertices of the fingerprints for u from the index database, calculate the empirical distribution over the vertices, multiply it with $1 - c$, and add c weight to vertex u . This requires one database access (disk seek).

To reach a precision beyond the number of fingerprints saved in the database we can use the *recursive property* of PPV, which is also referred to as the decomposition theorem in [26]:

$$\text{PPV}(u) = c \mathbb{1}_u + (1 - c) \frac{1}{|O(u)|} \sum_{v \in O(u)} \text{PPV}(v)$$

where $\mathbb{1}_u$ denotes the measure concentrated at vertex u (i.e., the unit vector of u), and $O(u)$ is the set of out-neighbors of u .

This gives us the following algorithm: upon a query u we load the fingerprints for u , the set of out-neighbors $O(u)$, and the fingerprints for the vertices of $O(u)$. From this set of fingerprints we use the above equation to approximate $PPV(u)$ using a higher amount of samples, thus achieving higher precision. This is a tradeoff between query time (database accesses) and precision: with $|O(u)|$ database accesses we can approximate the vector from $|O(u)| \cdot N$ samples. We can iterate this recursion, if want to have even more samples. We mention that such query time iterations are analogous to the basic dynamic programming algorithm of [26]. The main difference is that in our case the iterations are used to increase the number of fingerprints rather than the maximal length of the paths taken into account as in [26].

The increased precision is essential in approximating the PPV of a page with large neighborhood, as from N samples at most N pages will have positive approximated PPR values. Fortunately, this set is likely to contain the pages with highest PPR scores. Using the samples of the neighboring vertices will give more adequate result, as it will be formally analyzed in the next section.

We could also use the expander property of the web graph: after not so many random steps the distribution of the current vertex will be close to the static PageRank vector. Instead of allowing very long fingerprint paths we could combine the PR vector with coefficient $(1 - c)^{L+1}$ to the approximation and drop all fingerprints longer than L . This would also solve the problem of the approximated individual PPR vectors having many zeros (in those vertices that have no fingerprints ending there). The indexing algorithms would benefit from this truncation, too.

There is a further interesting consequence of the recursive property. If it is known in advance that we want to personalize over a fixed (maybe large) set of pages, we can introduce an artificial node into the graph with the respective set of neighbors to generate fingerprints for that combination.

3 How Many Fingerprints are Needed?

In this section we will discuss the convergence of our estimates, and analyze the required amount of fingerprints for proper precision.

It is clear by the law of large numbers that as the number of fingerprints $N \rightarrow \infty$, the estimate $\widehat{PPV}(u)$ converges to the actual personalized PageRank vector $PPV(u)$. To show that the rate of convergence is exponential, recall that each fingerprint of u ends at v with probability $PPV(u, v)$, where $PPV(u, v)$ denotes the v^{th} coordinate of $PPV(u)$. Therefore $N \cdot \widehat{PPV}(u, v)$, the number of fingerprints of u that ends at v , has binomial distribution with parameters N and $PPV(u, v)$. Then Chernoff's inequality yields the following bound on the error of over-estimating $PPV(u, v)$ and the same bound holds for under-estimation:

$$\begin{aligned} \Pr\{\widehat{PPV}(u, v) > (1 + \delta) PPV(u, v)\} &= \Pr\{N \widehat{PPV}(u, v) > N(1 + \delta) PPV(u, v)\} \\ &\leq e^{-N \cdot PPV(u, v) \cdot \delta^2 / 4}. \end{aligned}$$

Actually, for applications the exact values are not necessary. We only need that the ordering defined by the approximation match fairly closely the ordering defined by the personalized PageRank values. In this sense we have exponential convergence too:

Theorem 3. *For any vertices u, v, w consider $\text{PPV}(u)$ and assume that:*

$$\text{PPV}(u, v) > \text{PPV}(u, w)$$

Then the probability of interchanging v and w in the approximate ranking tends to 0 exponentially in the number of fingerprints used.

Theorem 4. *For any $\epsilon, \delta > 0$ there exists an N_0 such that for any $N \geq N_0$ number of fingerprints, for any graph and any vertices u, v, w such that $\text{PPV}(u, v) - \text{PPV}(u, w) > \delta$, the inequality $\Pr\{\widehat{\text{PPV}}(u, v) < \widehat{\text{PPV}}(u, w)\} < \epsilon$ holds.*

Proof. We prove both theorems together. Consider a fingerprint of u and let Z be the following random variable: $Z = 1$, if the fingerprint ends in v , $Z = -1$ if the fingerprint ends in w , and $Z = 0$ otherwise. Then $\mathbb{E} Z = \text{PPV}(u, v) - \text{PPV}(u, w) > 0$. Estimating the PPV values from N fingerprints the event of interchanging v and w in the rankings is equivalent to taking N independent Z_i variables and having $\sum_{i=1}^N Z_i < 0$. This can be upper bounded using Bernstein's inequality and the fact that $\text{Var}(Z) = \text{PPV}(u, v) + \text{PPV}(u, w) - (\text{PPV}(u, v) - \text{PPV}(u, w))^2 \leq \text{PPV}(u, v) + \text{PPV}(u, w)$:

$$\begin{aligned} \Pr\left\{\frac{1}{N} \sum_{i=1}^N Z_i < 0\right\} &\leq e^{-N \frac{(\mathbb{E} Z)^2}{2 \text{Var}(Z) + 4/3 \mathbb{E} Z}} \\ &\leq e^{-N \frac{(\text{PPV}(u, v) - \text{PPV}(u, w))^2}{10/3 \text{PPV}(u, v) + 2/3 \text{PPV}(u, w)}} \\ &\leq e^{-0.3N(\text{PPV}(u, v) - \text{PPV}(u, w))^2} \end{aligned}$$

From the above inequality both theorems follow. \square

The first theorem shows that even a modest amount of fingerprints are enough to distinguish between the high, medium and low ranked pages according to the personalized PageRank scores. However, the order of the low ranked pages will usually not follow the PPR closely. This is not surprising, and actually a significant problem of PageRank itself, as [32] showed that PageRank is unstable around the low ranked pages, in the sense that with small perturbation of the graph a very low ranked page can jump in the ranking order somewhere to the middle.

The second statement has an important theoretical consequence. When we investigate the asymptotic growth of database size as a function of the graph size, the number of fingerprints remains constant for fixed ϵ and δ .

4 Worst Case Lower Bounds for PPR Database Size

In this section we will prove several worst case lower bounds on the complexity of personalized PageRank problem. The lower bounds suggest that the exact

computation and storage of all personalized PageRank vectors is infeasible for massive graphs. Notice that the theorems cannot be applied to one specific input such as the webgraph. The theorems show that for achieving full personalization the web-search community should either utilize some specific properties of the webgraph or relax the exact problem to an approximate one as in our scenario.

In particular, we will prove that the necessary index database size of a fully personalized PageRank algorithm computing exact scores must be at least $\Omega(V^2)$ bits in worst case, and if personalizing only for H nodes, the size of the database is at least $\Omega(H \cdot V)$. If we allow some small error probability and approximation, then the lower bound for full personalization is linear in V , which is achieved by our algorithm of Section 2.

More precisely we will consider *two-phase algorithms*: in the first phase the algorithm has access to the graph and has to compute an index database. In the second phase the algorithm gets a query of arbitrary vertices u, v (and w), and it has to answer based on the index database, i.e., the algorithm cannot access the graph during query-time. An $f(V)$ *worst case lower bound* on the database size holds, if for any two-phase algorithm there exists a graph on V vertices such that the algorithm builds a database of size $f(V)$ in the first phase.

In the above introduced two-phase model, we will consider the following types of queries:

- (1) *Exact*: Calculate $\text{PPV}(u, v)$, the v^{th} element of the personalized PageRank vector of u .
- (2) *Approximate*: Estimate $\text{PPV}(u, v)$ with a $\widehat{\text{PPV}}(u, v)$ such, that for fixed $\epsilon, \delta > 0$

$$\Pr\{|\widehat{\text{PPV}}(u, v) - \text{PPV}(u, v)| < \delta\} \geq 1 - \epsilon$$

- (3) *Positivity*: Decide whether $\text{PPV}(u, v)$ is positive with error probability at most ϵ .
- (4) *Comparison*: Decide in which order v and w are in the personalized rank of u with error probability at most ϵ .
- (5) ϵ - δ *comparison*: For fixed $\epsilon, \delta > 0$ decide the comparison problem with error probability at most ϵ , if $|\text{PPV}(u, v) - \text{PPV}(u, w)| > \delta$ holds.

Our tool towards the lower bounds will be the asymmetric communication complexity game *bit-vector probing* [24]: there are two players A and B . Player A has an m -bit vector x , player B has a number $y \in \{1, 2, \dots, m\}$, and they have to compute the function $f(x, y) = x_y$, i.e., the output is the y^{th} bit of the input vector. To compute the proper output they have to communicate, and communication is restricted in the direction $A \rightarrow B$. The *one-way communication complexity* [31] of this function is the required bits of transfer in the worst case for the best protocol.

Theorem 5 ([24]). *Any protocol that outputs the correct answer to the bit-vector probing problem with probability at least $\frac{1+\gamma}{2}$ must transmit at least γm bits in worst case.*

Now we are ready to prove our lower bounds. In all our theorems we assume that personalization is calculated for H vertices, and there are V vertices in total. Notice that in the case of full personalization $H = V$ holds.

Theorem 6. *Any algorithm solving the positivity problem (3) must use an index database of size $\Omega((1 - 2\epsilon)HV)$ bits in worst case.*

Proof. Set $\frac{1+\gamma}{2} = 1 - \epsilon$. We give a communication protocol for the bit-vector probing problem. Given an input bit-vector x we will create a graph, that ‘codes’ the bits of this vector. Player A will create a PPV database on this graph, and transmit this database to B . Then Player B will use the positivity query algorithm for some vertices (depending on the requested number y) such that the answer to the positivity query will be the y^{th} bit of the input vector x . Thus if the algorithm solves the PPV indexing and positivity query with error probability ϵ , then this protocol solves the bit-vector probing problem with probability $\frac{1+\gamma}{2}$, so the transferred index database’s size is at least γm .

For the $H \leq V/2$ case consider the following graph: let u_1, \dots, u_H denote the vertices for whose the personalization is calculated. Add v_1, v_2, \dots, v_n more vertices to the graph, where $n = V - H$. Let the input vector’s size be $m = H \cdot n$. In our graph each vertex v_j has a loop, and for each $1 \leq i \leq H$ and $1 \leq j \leq n$ the edge (u_i, v_j) is in the graph iff bit $(i - 1)n + j$ is set in the input vector.

For any number $1 \leq y \leq m$ let $y = (i - 1)n + j$; the personalized PageRank value $\text{PPV}(u_i, v_j)$ is positive iff (u_i, v_j) edge was in the graph, thus iff bit y was set in the input vector. If $H \leq V/2$ the theorem follows since $n = V - H = \Omega(V)$ holds implying that $m = H \cdot n = \Omega(H \cdot V)$ bits are ‘coded’.

Otherwise, if $H > V/2$ the same construction proves the statement with setting $H = V/2$. \square

Corollary 1. *Any algorithm solving the exact PPV problem (1) must have an index database of size $\Omega(H \cdot V)$ bits in worst case.*

Theorem 7. *Any algorithm solving the approximation problem (2) needs an index database of $\Omega(\frac{1-2\epsilon}{\delta}H)$ bits on a graph with $V = H + \Omega(\frac{1}{\delta})$ vertices in worst case. If $V = H + O(\frac{1}{\delta})$, then the index database requires $\Omega((1 - 2\epsilon)HV)$.*

Proof. We will modify the construction of Theorem 6 for the approximation problem. We have to achieve that when a bit is set in the input graph, then the queried $\text{PPV}(u_i, v_j)$ value should be at least 2δ , so that the approximation will decide the positivity problem, too. If there are k edges incident to vertex u_i in the constructed graph, then each target vertex v_j has weight $\text{PPV}(u_i, v_j) = \frac{1-\epsilon}{k}$. For this to be over 2δ we can have at most $n = \frac{1-\epsilon}{2\delta}$ possible v_1, \dots, v_n vertices. With $\frac{1+\gamma}{2} = 1 - \epsilon$ the first statement of the theorem follows.

For the second statement the original construction suffices. \square

This radical drop in the storage complexity is not surprising, as our approximation algorithm achieves this bound (up to a logarithmic factor): for fixed ϵ, δ we can calculate the necessary number of fingerprints N , and then for each vertex in the personalization we store exactly N fingerprints, independently of the graph’s size.

Theorem 8. *Any algorithm solving the comparison problem (4) requires an index database of $\Omega((1 - 2\epsilon)HV)$ bits in worst case.*

Proof. We will modify the graph of Theorem 6 so that the existence of the specific edge can be queried using the comparison problem. To achieve this we will introduce a third set of vertices w_1, \dots, w_n in the graph construction, such that w_j is the complement of v_j : A puts the edge (u_i, w_j) in the graph iff (u_i, v_j) was not an edge, which means bit $(i - 1)n + j$ was not set in the input vector.

Then upon query for bit $y = (i - 1)n + j$, consider $\text{PPV}(u_i)$. In this vector exactly one of v_j, w_j will have positive weight (depending on the input bit x_y), thus the comparison query $\text{PPV}(u_i, v_j) > \text{PPV}(u_i, w_j)$ will yield the required output for the bit-vector probing problem. \square

Corollary 2. *Any algorithm solving the ϵ - δ comparison problem (5) needs an index database of $\Omega(\frac{1-2\epsilon}{\delta}H)$ bits on a graph with $V = H + \Omega(\frac{1}{\delta})$ vertices in worst case. If $V = H + O(\frac{1}{\delta})$, then the index database needs $\Omega((1 - 2\epsilon)HV)$ bits in worst case.*

Proof. Modifying the proof of Theorem 8 according to the proof of Theorem 7 yields the necessary results. \square

5 Experiments

In this section we present experiments that compare our approximate PPR scores to exact PPR scores computed by the personalized PageRank algorithm of Jeh and Widom [26]. Our evaluation is based on the web graph of 80 million pages crawled in 2001 by the Stanford WebBase Project [25]. We also validated the tendencies presented on a 31 million page web graph of the .de domain created using the Polybot crawler [38] in April 2004.

In the experiments we personalize on a single page u chosen uniformly at random from all vertices with non-zero outdegree. The experiments were carried out with 1000 independently chosen personalization node u , and the results were averaged.

To compare the exact and approximate PPR scores for a given personalization page u , we measure the difference between top score lists of exact $\text{PPV}(u)$ and approximate $\widehat{\text{PPV}}(u)$ vectors. The length k of the compared top lists is in the range 10 to 1000.

As our primary application area is query result ranking, we chose measures that compare the ordering returned by the approximate PPR method to the ordering specified by the exact PPR scores. In Section 5.1 we describe these measures that numerically evaluate the similarity of the top k lists. In Section 5.2 we present our experimental results.

5.1 Comparison of ranking algorithms

The problem of comparing the top k lists of different ranking algorithms has been extensively studied by the web-search community for measuring the speed

of convergence in PageRank computations [28], the distortion of PageRank encodings [20] and the quality of rank-aggregation methods [15,14,?,12].

In our scenario the exact PPR scores provide the ground truth ranking and the following three methods evaluate the similarity of approximate scores to the exact scores.

Let T_k^u denote the set of pages having the k highest personalized PageRank values in the vector $PPV(u)$ personalized to a single page u . We approximate this set by \widehat{T}_k^u , the set of pages having the k highest approximated scores in vector $\widehat{PPV}(u)$ computed by our Monte Carlo algorithm.

The first two measures determine the overall quality of the approximated top- k set \widehat{T}_k^u , so they are insensitive to the ranking of the elements within \widehat{T}_k^u . *Relative aggregated goodness* [37] measures how well the approximate top- k set performs in finding a set of pages with high aggregated personalized PageRank. Thus relative aggregated goodness calculates the sum of exact PPR values in the approximate set compared to the maximum value achievable (by using the exact top- k set T_k^u):

$$\text{RAG}(k, u) = \frac{\sum_{v \in \widehat{T}_k^u} \text{PPV}(u, v)}{\sum_{v \in T_k^u} \text{PPV}(u, v)}$$

We also measure the *precision* of returning the top- k set in the classical information retrieval terminology (note that as the sizes of the sets are fixed, precision coincides with *recall*):

$$\text{Prec}(k, u) = \frac{|\widehat{T}_k^u \cap T_k^u|}{k}$$

The third measure, *Kendall's τ* compares the exact ranking with the approximate ranking in the top- k set. Note that the tail of approximate PPR ranking contains a large number of ties (nodes with equal approximated scores) that may have a significant effect on rank comparison. Versions of Kendall's τ with different tie breaking rules appear in the literature; we use the original definition as e.g. in [29] that we review next. Consider the pairs of vertices v, w . A pair is *concordant*, if both rankings strictly order this pair and agree on the ordering; *discordant*, if both rankings strictly order but disagree on the ordering of the pair; *e-tie*, if the exact ranking does not order the pair; *a-tie*, if the approximate ranking does not order the pair. Denote the number of these pairs by C, D, U_e and U_a respectively. The total number of possible pairs is $M = \frac{n(n-1)}{2}$, where $n = |T_k^u \cup \widehat{T}_k^u|$. Then *Kendall's τ* is defined as

$$\tau(k, u) = \frac{C - D}{\sqrt{(M - U_e)(M - U_a)}}$$

The range of Kendall's τ is $[-1, 1]$, thus we linearly rescaled it onto $[0, 1]$ to fit the other measures on the diagrams. To restrict the computation to the top k elements, the following procedure was used: we took the union of the exact and

approximated top- k sets $T_k^u \cup \widehat{T}_k^u$. For the exact ordering, all nodes that were outside T_k^u were considered to be tied and ranked strictly smaller than any node in T_k^u . Similarly, for the approximate ordering, all nodes that were outside the approximate top- k set \widehat{T}_k^u were considered to be tied and ranked strictly smaller than any node in \widehat{T}_k^u .

5.2 Results

We conducted experiments on a single AMD Opteron 2.0 Ghz machine with 4 GB of RAM under Linux OS. We used an elementary compression (much simpler and faster than [5]) to store the Stanford WebBase graph in 1.6 GB of main memory. The computation of 1000 approximated personalized PageRank vectors took 1.1 seconds (for $N = 1000$ fingerprints truncated at length $L = 12$). The exact PPR values were calculated using the algorithm by Jeh and Widom [26] with a precision of 10^{-8} in L_1 norm. The default parameters were number of fingerprints $N = 1000$ with one level of recursive evaluation (see Section 2.3) and maximal path length $L = 12$.

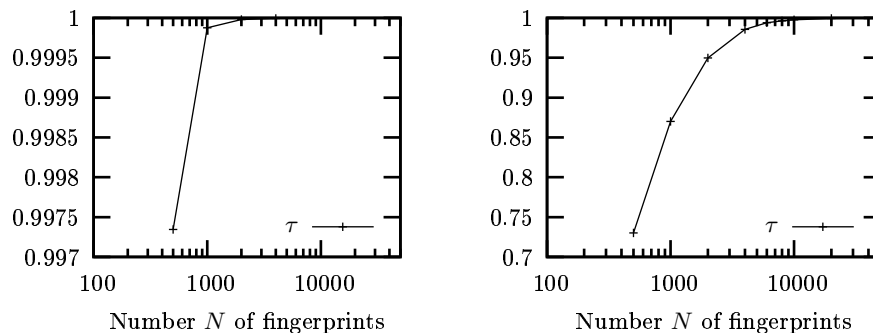


Figure 2. Effect of the number of fingerprints on Kendall's τ restricted to pairs with a PPR difference of at least $\delta = 0.01$ (left) and $\delta = 0.001$ (right).

In our first experiments depicted on Figure 2, we demonstrate the exponential convergence of Theorems 3 and 4. We calculated Kendall's τ restricted to pairs that have a difference at least δ in their exact PPR scores. We displayed the effect of the number of fingerprints on this restricted τ for $\delta = 0.01$ and $\delta = 0.001$. It can be clearly seen, that a modest amount of fingerprints suffices to properly order the pages with at least δ difference in their personalized PageRank values.

Figure 3 demonstrates the effects of the number of fingerprints and the recursive evaluation on the approximate ranking quality (without the previous restriction). The recursion was carried out for a single level of neighbors, which helped to reduce the number of fingerprints (thus the storage requirements) for the same ranking precision by an order of magnitude.

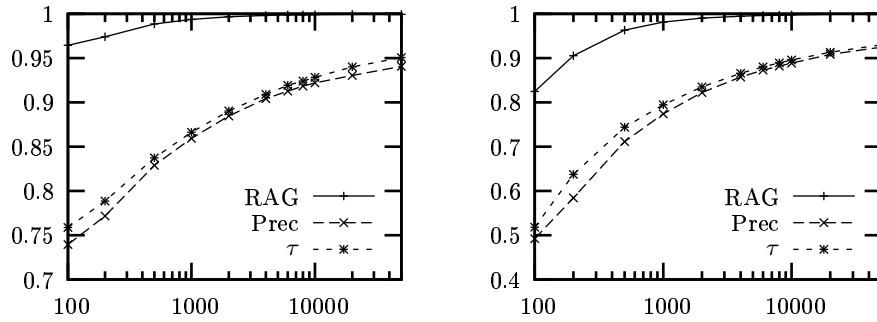


Figure 3. Effect of the number of fingerprints on various measures of goodness with (left) or without (right) recursive evaluation.

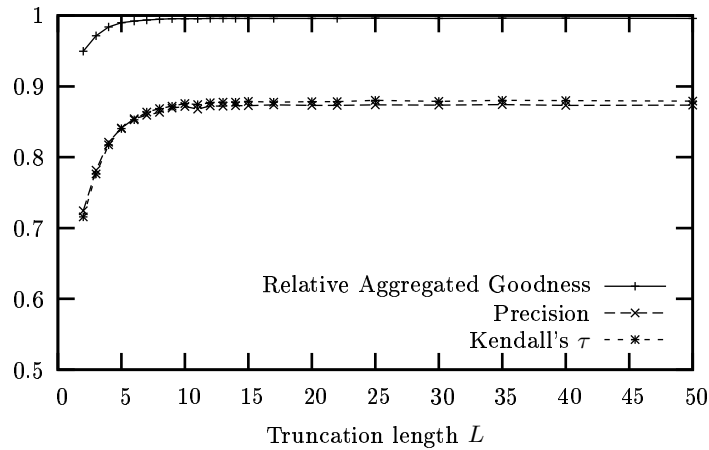


Figure 4. Effect of the path length/truncation on various measures of goodness.

Figure 4 shows the effect of truncating the fingerprints at a maximum path length. It can be seen, that paths over length 12 have small influence on the approximation quality, thus the computation costs can be reduced by truncating them.

Finally, Figure 5 and Figure 6 (Figure 6 for $N=10000$ fingerprints) indicate that as the top list size k increases, the task of approximating the top- k set becomes more and more difficult. This is mainly due to the fact that among lower ranked pages there is a smaller personalized PageRank difference, which is harder to capture using approximation methods (especially Monte Carlo methods).

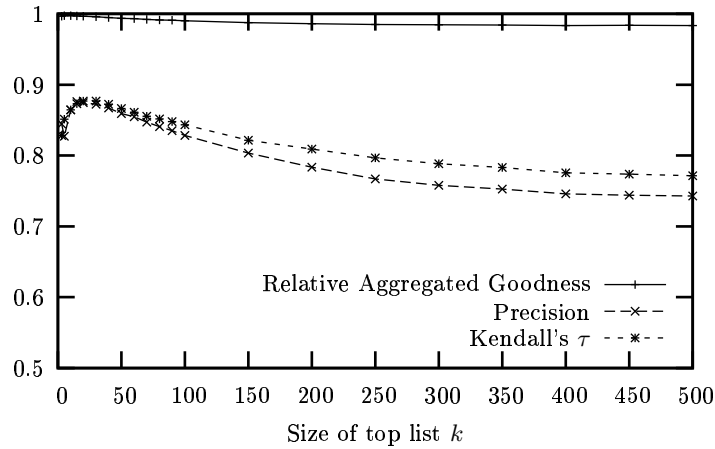


Figure 5. Effect of the size k of top set taken on various measures of goodness.

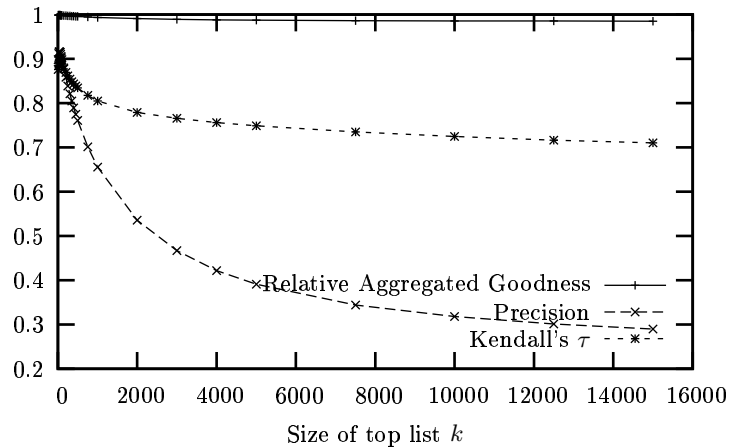


Figure 6. Effect of the size k of top set taken on various measures of goodness.

6 Conclusions and Open Problems

In this paper we introduced a new algorithm for calculating personalized Page-Rank scores. Our method is a randomized approximation algorithm based on simulated random walks on the web graph. It can provide full personalization with a linear space index, such that the error probability converges to 0 exponentially with increasing the index size. The index database can be computed even on the scale of the entire web, thus making the algorithms feasible for commercial web search engines.

We justified this relaxation of the personalized PageRank problem to approximate versions by proving quadratic lower bounds for the full personalization problems. For the estimated PPR problem our algorithm is space-optimal up to a logarithmic factor.

The experiments on 80M pages showed that using no more than $N = 1000$ fingerprints suffices for proper precision approximation.

An important future work is to combine and evaluate the available methods for computing personalized PageRank.

Acknowledgement

We wish to acknowledge András Benczúr and Katalin Friedl for several discussions and comments on this research. We thank Paul-Alexandru Chirita for giving us his implementation [35] of the Jeh-Widom Personalized PageRank algorithm [26] and Boldi et al. [4] for their fast Kendall τ code. We would like to thank the Stanford WebBase project [25] and Torsten Suel [38] for providing us with web graphs. Furthermore we are grateful to Glen Jeh for encouraging us to consider Monte-Carlo PPR algorithms, and the anonymous referee for the valuable comments on this manuscript.

References

1. Open Directory Project (ODP). <http://www.dmoz.org>.
2. Z. Bar-Yossef, A. Berg, S. Chien, J. Fakcharoenphol, and D. Weitz. Approximating aggregate queries about web pages via random walks. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 535–544. Morgan Kaufmann Publishers Inc., 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. In *Proceedings of the 13th World Wide Web Conference (WWW)*, pages 328–337. ACM Press, 2004.
4. P. Boldi, M. Santini, and S. Vigna. Do your worst to make the best: Paradoxical effects in PageRank incremental computations. In *Proceedings of the Third Workshop on Algorithms and Models for the Web-Graph (WAW 2004)*, volume 3243 of *Lecture Notes in Computer Science*, pages 118–130. Springer-Verlag, 2004.
5. P. Boldi and S. Vigna. The webgraph framework I: compression techniques. In *Proceedings of the 13th World Wide Web Conference (WWW)*, pages 595–602. ACM Press, 2004.
6. A. Borodin, G. O. Roberts, J. S. Rosenthal, and P. Tsaparas. Finding authorities and hubs from link structures on the world wide web. In *10th International World Wide Web Conference*, pages 415–429, 2001.
7. 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.
8. A. Z. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences (SEQUENCES'97)*, pages 21–29. IEEE Computer Society, 1997.
9. 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.

10. E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55(3):441–453, 1997.
11. J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 137–150, 2004.
12. C. Dwork, S. R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of the 10th International World Wide Web Conference (WWW)*, pages 613–622, Hong Kong, 2001.
13. N. Eiron and K. S. McCurley. Locality, hierarchy, and bidirectionality in the web. In *Second Workshop on Algorithms and Models for the Web-Graph (WAW 2003)*, 2003.
14. R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee. Comparing and aggregating rankings with ties. In *Proceedings of the 23rd ACM Symposium on Principles of Database Systems (PODS)*, pages 47–58, 2004.
15. R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. In *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 26–36, 2003. Full version in *SIAM Journal on Discrete Mathematics*, 17(1):134–160, 2003.
16. D. Fogaras. Where to start browsing the web? In *3rd International Workshop on Innovative Internet Community Systems I2CS, published as LNCS 2877/2003*, pages 65–79, 2003.
17. D. Fogaras and B. Rácz. Scaling link-based similarity search. To appear in the *14th World Wide Web Conference (WWW)*, 2005.
18. D. Fogaras and B. Rácz. A scalable randomized method to compute link-based similarity rank on the web graph. In *Proceedings of the Clustering Information over the Web workshop, Conference on Extending Database Technology*, 2004.
19. T. H. Haveliwala. Topic-sensitive PageRank. In *Proceedings of the 11th World Wide Web Conference (WWW)*, pages 517–526, Honolulu, Hawaii, 2002.
20. T. H. Haveliwala. Efficient encodings for document ranking vectors. In *Proceedings of the 4th International Conference on Internet Computing (IC)*, pages 3–9, Las Vegas, Nevada, USA, 2003.
21. T. H. Haveliwala, S. Kamvar, and G. Jeh. An analytical comparison of approaches to personalizing PageRank. Technical report, Stanford University, 2003.
22. M. R. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork. Measuring index quality using random walks on the Web. In *Proceedings of the 8th World Wide Web Conference, Toronto, Canada*, pages 213–225, 1999.
23. M. R. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork. On near-uniform url sampling. In *Proceedings of the 9th international World Wide Web conference on Computer networks*, pages 295–308, 2000.
24. M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. *External memory algorithms*, pages 107–118, 1999.
25. J. Hirai, S. Raghavan, H. Garcia-Molina, and A. Paepcke. Webbase: a repository of web pages. In *Proceedings of the 9th World Wide Web Conference (WWW)*, pages 277–293. North-Holland Publishing Co., 2000.
26. G. Jeh and J. Widom. Scaling personalized web search. In *Proceedings of the 12th World Wide Web Conference (WWW)*, pages 271–279. ACM Press, 2003.
27. S. Kamvar, T. H. Haveliwala, C. Manning, and G. Golub. Exploiting the block structure of the web for computing PageRank. Technical report, Stanford University, 2003.
28. S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub. Extrapolation methods for accelerating pagerank computations. In *Proceedings of the 12th World Wide Web Conference (WWW)*, pages 261–270. ACM Press, 2003.

29. M. G. Kendall. *Rank Correlation Methods*. Hafner Publishing Co., New York, 1955.
30. J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.
31. E. Kushilevitz and N. Nisan. *Communication complexity*. Cambridge University Press, 1997.
32. R. Lempel and S. Moran. Rank stability and rank similarity of link-based web ranking algorithms in authority connected graphs. In *Second Workshop on Algorithms and Models for the Web-Graph (WAW 2003)*, 2003.
33. 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.
34. 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.
35. D. O. Paul-Alexandru Chirita and W. Nejdl. PROS: A personalized ranking platform for web search. In *Proceedings of the 3rd International Conference on Adaptive Hypermedia and Adaptive Web Based Systems*, Eindhoven, Netherlands, 2004.
36. P. Rusmevichientong, D. M. Pennock, S. Lawrence, and C. L. Giles. Methods for sampling pages uniformly from the world wide web. In *AAAI Fall Symposium on Using Uncertainty Within Computation*, pages 121–128, 2001.
37. P. K. C. Singitham, M. S. Mahabhashyam, and P. Raghavan. Efficiency-quality tradeoffs for vector score aggregation. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*, pages 624–635, 2004.
38. T. Suel and V. Shkapenyuk. Design and implementation of a high-performance distributed web crawler. In *Proceedings of the 3rd IEEE International Conference on Data Engineering*, pages 357–368, February 2002.