Internet Mathematics Vol. 2, No. 3: 333-358

Towards Scaling Fully Personalized PageRank: Algorithms, Lower Bounds, and Experiments

Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós

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 online queries only for a restricted choice of pages. In this paper we achieve full personalization by a novel algorithm that precomputes a compact database; using this database, it can serve online 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 has been present in the literature since the beginning of the success story of Google's PageRank [Brin and Page 98, Page et al. 98] and other hyperlink-based quality measures [Kleinberg 99, Borodin et al. 01]. Topic sensitivity is either achieved by precomputing modified measures over the entire web [Haveliwala 02] or by ranking the neighborhood of pages containing the query word [Kleinberg 99]. These methods,

1542-7951/05 $0.50 per page

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 [Haveliwala 02, Jeh and Widom 03] of personalized PageRank [Page et al. 98]. Just as all hyperlink based ranking methods, PageRank is based on the assumption that *the existence of a hyperlink $u \to v$ 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 online query response service.

Earlier personalized PageRank algorithms restricted personalization to a few topics [Haveliwala 02], to a subset of popular pages [Jeh and Widom 03], or to hosts [Kamvar et al. 03a]; see [Haveliwala et al. 03] for an analytical comparison of these methods. The state of the art Hub Decomposition algorithm [Jeh and Widom 03] 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 [Netscape 05].

In contrast to earlier PPR algorithms, we achieve *full personalization*: our method enables online 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 online 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 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 web graph 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 web graph 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 the 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 [Haveliwala 02] and for popular pages with large neighborhoods by the hub skeleton algorithm [Jeh and Widom 03], 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 online personalization 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 [Page et al. 98] (Naive in Table 1) simply takes the personalization vector as input and performs power iteration at query time. This approach is clearly infeasible for online queries. One may precompute the power iterations for a well-selected set of personalization vectors as in the Topic Sensitive PageRank [Haveliwala 02]; 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 [Kamvar et al. 03a], 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 online queries.

The remarkable Hub Decomposition algorithm [Jeh and Widom 03] 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 [Jeh and Widom 03] to lower estimate the personalized PageRank vector of arbitrary pages by taking into account only the paths that go 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 [Jeh and Widom 03] 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, which 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 Neighb(k) within distance k as seen in Figure 1. The average size of the sparse vectors exceeds 1,000 after $k \geq 4$ iterations, and on average 24% of all vertices are reached within $k = 15$ steps.[1] 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.

The basic dynamic programming method can be extended with a pruning strategy that eliminates some of the nonzero entries from the approximation vectors. However, it is nontrivial 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; we refer the reader to our follow-up work [Sarlós et al. 05] for further details. Certain pruning strategies may also increase the precomputation time. For example, a major drawback of selecting the top ranks after each iteration is that it 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, [Palmer et al. 02] applies probabilistic counting to estimate the neighborhood function of the Internet graph, [Cohen 97] estimates the size of transitive closure for massive graphs occurring in databases, and [Fogaras and Rácz 04, Fogaras and Rácz 05] approximates link-based similarity scores by fingerprints. Apart from graph algorithms, [Broder 97] estimates the resemblance and containment of textual documents with fingerprinting.

[1] The neighborhood function was computed by combining the size estimation method of [Cohen 97] with our external memory algorithm discussed in [Fogaras and Rácz 05].

Method	Personalization	Limits of scalability	Postitive aspects	Negative aspects
Naive [Page et al. 98]	any page	power iteration in query time		infeasible to serve online personalization
Topic-Sensitive PageRank [Haveliwala 02]	restricted to linear combination of t topics, e.g., $t = 16$	$t \cdot V$ disk space required	distributed computing	
BlockRank [Kamvar et al. 03a]	restricted to personalize on hosts	power iteration in query time	reduced number of power iterations, distributed computing	infeasible to serve online personalization
Hub Decomposition [Jeh and Widom 03]	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 [Jeh and Widom 03]	any page	$V \cdot \text{Neighb}(k)$ disk space required for k iterations, where 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.

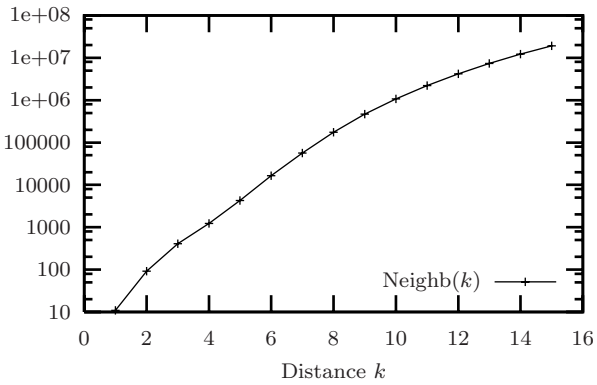

Figure 1. The neighborhood function measured on the Stanford WebBase graph of 80M pages.

Random walks were used earlier to compute various web statistics, mostly focused on sampling the web (uniformly or according to static PR) [Henzinger et al. 00, Rusmevichientong et al. 01, Bar-Yossef et al. 00, Henzinger et al. 99a] but also for calculating page decay [Bar-Yossef et al. 04] and similarity values [Fogaras and Rácz 04, Fogaras and Rácz 05].

The lower bounds of Section 4 show that precise PPR requires a significantly larger database than Monte Carlo estimation does. Analogous results with similar communication complexity arguments were proved in [Henzinger et al. 99b] for the space complexity of several data stream graph algorithms.

1.2. Preliminaries

In this section we introduce notation and 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 web graph with normalized rows and $c \in (0, 1)$ the *teleportation probability*. In addition, let $\vec{r}$ be the so-called *preference vector* inducing a probability distribution over $\mathcal{V}$. *PageRank vector* $\vec{p}$ is defined as the solution of the following equation [Page et al. 98]:

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

If $\vec{r}$ is uniform over $\mathcal{V}$, then $\vec{p}$ is referred to as the *global PageRank vector*. For nonuniform $\vec{r}$ the solution $\vec{p}$ will be referred to as the *personalized PageRank vector of* $\vec{r}$, denoted by $\mathrm{PPV}(\vec{r})$. In the special case when for some page u the

uth coordinate of $\vec{r}$ is 1 and all other coordinates are 0, the PPV will be referred to as the *individual PageRank vector* of u denoted by $\mathrm{PPV}(u)$. We will also refer to this vector as the personalized PageRank vector of u. Furthermore, the vth coordinate of $\mathrm{PPV}(u)$ will be denoted by $\mathrm{PPV}(u, v)$.

Theorem 1.1. (Linearity.) [Haveliwala 02] *For any preference vectors $\vec{r}_1$, $\vec{r}_2$ and positive constants α_1, α_2 with $\alpha_1 + \alpha_2 = 1$, the following equality holds:*

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

Linearity is a fundamental tool for scalable online 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 $\mathrm{PPV}(u)$ for all $u \in \mathcal{V}$, and the individual PPVs can be combined online for any small subset of pages. Therefore, in the rest of this paper, we investigate algorithms to make all individual PPVs available online.

The following statement will play a central role in our PPV estimations. The theorem provides an alternate probabilistic characterization of individual PageRank scores.[2]

Theorem 1.2. [Jeh and Widom 03, Fogaras 03] *Suppose that a number L is chosen at random with probability $\Pr\{L = i\} = c(1-c)^i$ for $i = 0, 1, 2, \ldots$. Consider a random walk starting from some page u and taking L steps. Then, for the vth coordinate $\mathrm{PPV}(u, v)$ of vector $\mathrm{PPV}(u)$,*

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

2. Personalized PageRank Algorithm

In this section we will present a new Monte Carlo algorithm to compute approximate values of personalized PageRank utilizing the probabilistic characterization of PPR from Section 1. We will compute approximations of each of the PageRank vectors personalized on a single page; therefore, by the linearity theorem we achieve full personalization.

[2]Notice that this characterization slightly differs from the random surfer formulation of PageRank [Page et al. 98].

Definition 2.1. (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 2.2. (Fingerprint.) A *fingerprint* of a vertex u is the ending vertex of a fingerprint path of u.

By Theorem 1.2 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 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 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 [Boldi and Vigna 04] 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 the case of the *external memory model* and a *distributed system* with tens to thousands of medium capacity computers [Dean and Ghemawat 04]. Both algorithms use techniques similar to those of the respective I/O efficient algorithms computing PageRank [Chen et al. 02].

Since 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.

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] := ∅
while Paths ≠ ∅ 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)
```

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 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 that 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 that 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.[3]

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

[3]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, 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 (PathStart, PathEnd).

```
for u with part(u) = current computer do
  for i := 1 to N do
    insert pair (u,u) into InQueue /*start N fingerprint paths from node u: initially,
    PathStart=PathEnd= u*/
while at least one queue is not empty do /*some of the fingerprints are still being calcu-
lated*/
  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 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 := 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.
```

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, 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, 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 the current computer's graph or part(v) has to be computable from v, for example, by renumbering the vertices according to the partition. For the sake of simplicity, the output queue management is omitted from the pseudocode 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 the case of the web graph, we can significantly reduce the 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 [Kamvar et al. 03a, Eiron and McCurley 03].

We can further extend this heuristic 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 such a partition is being used, 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 $\mathrm{PPV}(u)$, 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 [Jeh and Widom 03]:

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

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 $\mathrm{PPV}(u)$ 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 we want to have even more samples. We mention that such query-time iterations are analogous to the basic dynamic programming algorithm of [Jeh and Widom 03]. 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 [Jeh and Widom 03].

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

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

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

Theorem 3.1. *For any vertices u, v, w consider $\mathrm{PPV}(u)$ and assume that*

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

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

Theorem 3.2. *For any $\epsilon, \delta > 0$ there exists an N_0 such that for any $N \ge N_0$ number of fingerprints, for any graph, and any vertices u, v, w such that $\mathrm{PPV}(u,v) - \mathrm{PPV}(u,w) > \delta$, the inequality $\Pr\{\widehat{\mathrm{PPV}}(u,v) < \widehat{\mathrm{PPV}}(u,w)\} < \epsilon$ 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 $\mathbb{E}Z = \mathrm{PPV}(u, v) - \mathrm{PPV}(u, w) > 0$. 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 $\mathrm{Var}(Z) = \mathrm{PPV}(u, v) + \mathrm{PPV}(u, w) - (\mathrm{PPV}(u, v) - \mathrm{PPV}(u, w))^2 \leq \mathrm{PPV}(u, v) + \mathrm{PPV}(u, w)$:

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

From the above inequality both theorems follow. □

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 [Lempel and Moran 05] 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 web graph. The theorems show that for achieving full personalization the web-search community should either utilize some specific properties of the web graph 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 the worst case, and if it personalizes 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 two-phase model, we will consider the following types of queries:

(1) *Exact:* Calculate $\mathrm{PPV}(u, v)$, the vth element of the personalized PageRank vector of u.

(2) *Approximate:* Estimate $\mathrm{PPV}(u, v)$ with a $\widehat{\mathrm{PPV}}(u, v)$ such that for fixed $\epsilon, \delta > 0$
$$\Pr\{|\widehat{\mathrm{PPV}}(u, v) - \mathrm{PPV}(u, v)| < \delta\} \geq 1 - \epsilon.$$

(3) *Positivity:* Decide whether $\mathrm{PPV}(u, v)$ 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 $|\mathrm{PPV}(u, v) - \mathrm{PPV}(u, w)| > \delta$ holds.

Our tool towards the lower bounds will be the asymmetric communication complexity game *bit-vector probing* [Henzinger et al. 99b]: there are two players A and B. Player A has an m-bit vector x, player B has a number $y \in \{1, 2, \ldots, m\}$, and their task is to compute the function $f(x, y) = x_y$, i.e., the output is the yth bit of the input vector. To compute the proper output, they have to communicate, and communication is restricted in the direction $A \to B$. The *one-way communication complexity* [Kushilevitz and Nisan 97] of this function is the required bits of transfer in the worst case for the best protocol.

Theorem 4.1. [Henzinger et al. 99b] *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 the 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 that there are V vertices in total. Notice that in the case of full personalization $H = V$ holds.

Theorem 4.2. *Any algorithm solving the positivity problem (3) must use an index database of size $\Omega((1-2\epsilon)HV)$ bits in the 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 yth 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 $u_1, \ldots, u_H$ denote the vertices whose personalization is calculated. Add $v_1, v_2, \ldots, v_n$ 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 $\mathrm{PPV}(u_i, v_j)$ is positive iff edge (u_i, v_j) 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 $H = V/2$. □

Corollary 4.3. *Any algorithm solving the exact* PPV *problem (1) must have an index database of size $\Omega(H \cdot V)$ bits in the worst case.*

Theorem 4.4. *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 the 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 4.2 for the approximation problem. We have to achieve that when a bit is set in the input graph, then the queried $\mathrm{PPV}(u_i, v_j)$ 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 $\mathrm{PPV}(u_i, v_j) = \frac{1-c}{k}$.

For this to be over 2δ, we can have at most $n = \frac{1-c}{2\delta}$ possible $v_1, \ldots, v_n$ vertices. With $\frac{1+\gamma}{2} = 1 - \epsilon$, the first statement of the theorem follows.

For the second statement the original construction suffices. □

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

Proof. We will modify the graph of Theorem 4.2 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 $w_1, \ldots, w_n$ 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 $\mathrm{PPV}(u_i)$. 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 $\mathrm{PPV}(u_i, v_j) > \mathrm{PPV}(u_i, w_j)$ will yield the required output for the bit-vector probing problem. □

Corollary 4.6. *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 the worst case. If $V = H + O(\frac{1}{\delta})$, then the index database needs $\Omega((1-2\epsilon)HV)$ bits in the worst case.*

Proof. Modifying the proof of Theorem 4.5 according to the proof of Theorem 4.4 yields the necessary results. □

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 [Jeh and Widom 03]. Our evaluation is based on the web graph of 80 million pages crawled in 2001 by the Stanford WebBase Project [Hirai et al. 00]. We also validated the tendencies presented on a 31 million page web graph of

the .de domain created using the Polybot crawler [Suel and Shkapenyuk 02] in April 2004.

In the experiments we personalize on a single page u chosen uniformly at random from all vertices with nonzero out-degree. The experiments were carried out with 1,000 independently-chosen personalization nodes, 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 $\mathrm{PPV}(u)$ and approximate $\widehat{\mathrm{PPV}}(u)$ vectors. The length k of the compared top lists is in the range 10 to 1,000.

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 [Kamvar et al. 03b], the distortion of PageRank encodings [Haveliwala 03], and the quality of rank-aggregation methods [Fagin et al. 03b, Fagin et al. 04, Fagin et al. 03a, Dwork et al. 01].

In our scenario the exact PPR scores provide the ground truth ranking and the following three methods evaluate the similarity of the 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 $\widehat{\mathrm{PPV}}(u)$ 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* [Singitham et al. 04] 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):

$$\mathrm{RAG}(k,u) = \frac{\sum_{v \in \widehat{T_k^u}} \mathrm{PPV}(u,v)}{\sum_{v \in T_k^u} \mathrm{PPV}(u,v)}.$$

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*):

$$\mathrm{Prec}(k,u) = \frac{|\widehat{T_k^u} \cap T_k^u|}{k}.$$

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 [Kendall 55] 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, and *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 the 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 [Boldi and Vigna 04]) to store the Stanford WebBase graph in 1.6 GB of main memory. The computation of 1,000 approximated personalized PageRank vectors took 1.1 seconds (for $N = 1,000$ fingerprints truncated at length $L = 12$). The exact PPR values were calculated using the algorithm by Jeh and Widom [Jeh and Widom 03] with a precision of 10^{-8} in L_1 norm. The default parameters were the number of fingerprints $N = 1,000$ with one level of recursive evaluation (see Section 2.3) and the maximal path length $L = 12$.

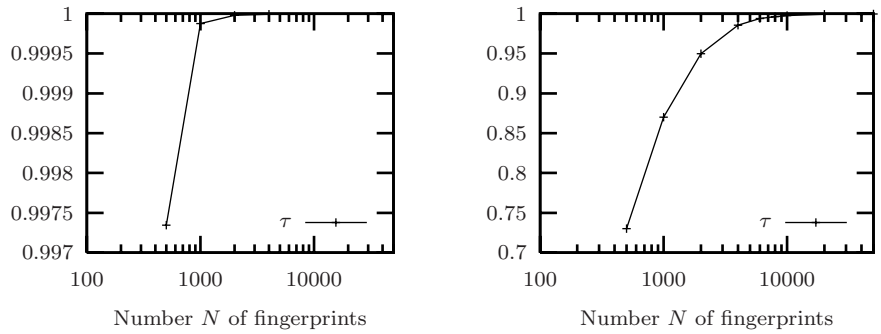

Figure 2. Effect of the number of fingerprints on Kendall's τ restricted to pairs with a PPR difference of at least $\delta = 0.01$ (left) and $\delta = 0.001$ (right).

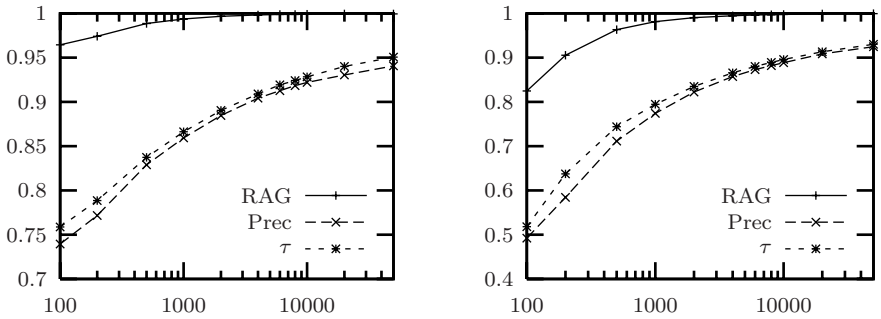

Figure 3. Effect of the number of fingerprints on various measures of goodness with (left) or without (right) recursive evaluation.

In our first experiments, depicted in Figure 2, we demonstrate the exponential convergence of Theorems 3.1 and 3.2. 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.

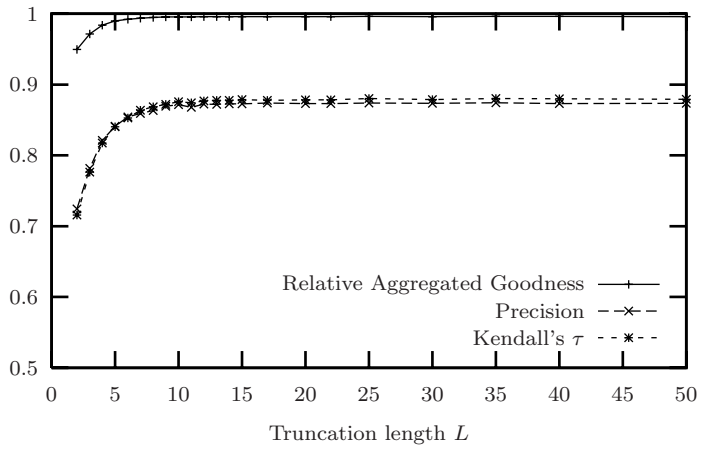

Figure 4. Effect of the path length/truncation on various measures of goodness.

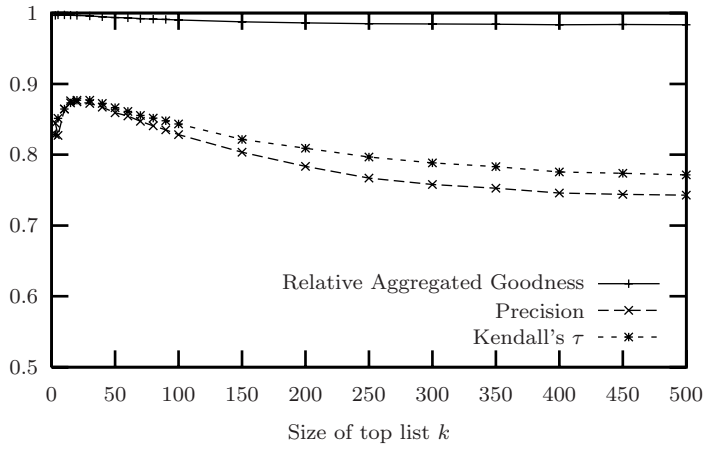

Figure 5. Effect of the size k of the top set taken on various measures of goodness.

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, Figures 5 and 6 (Figure 6 for N=10,000 fingerprints) indicate that as the top-list size k increases, the task of approximating the top-k set becomes

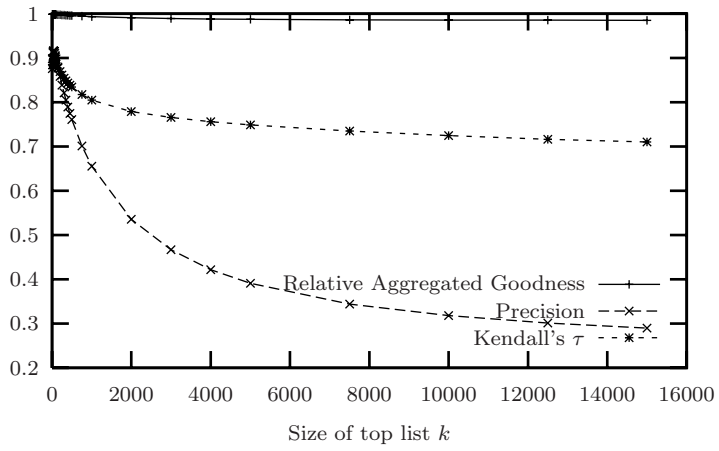

Figure 6. Effect of the size k of the top set taken on various measures of goodness.

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 (especially Monte Carlo) methods.

6. Conclusions and Open Problems

In this paper we introduced a new algorithm for calculating personalized PageRank 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 zero exponentially with increasing 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 $N = 1,000$ fingerprints suffices for proper precision approximation.

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

Acknowledgements. 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 [Chirita et al. 04] of the Jeh-Widom Personalized PageRank algorithm [Jeh and Widom 03] and Boldi et al. [Boldi et al. 04] for their fast Kendall's τ code. We would like to thank the Stanford WebBase project [Hirai et al. 00] and Torsten Suel [Suel and Shkapenyuk 02] 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.

Our 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.

References

[Bar-Yossef et al. 00] 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*, pp. 535–544. San Francisco: Morgan Kaufmann Publishers Inc., 2000.

[Bar-Yossef et al. 04] 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 International Conference on World Wide Web*, pp. 328–337. New York: ACM Press, 2004.

[Boldi and Vigna 04] P. Boldi and S. Vigna. "The Webgraph Framework I: Compression Techniques." In *Proceedings of the 13th International Conference on World Wide Web*, pp. 595–602. New York: ACM Press, 2004.

[Boldi et al. 04] P. Boldi, M. Santini, and S. Vigna. "Do Your Worst to Make the Best: Paradoxical Effects in PageRank Incremental Computations." In *Algorithms and Models for the Web-Graph: Third International Workshop, WAW 2004, Rome, Italy, October 16, 2004, Proceedings*, pp. 168–180, Lecture Notes in Computer Science 3243. Berlin: Springer, 2004.

[Borodin et al. 01] A. Borodin, G. O. Roberts, J. S. Rosenthal, and P. Tsaparas. "Finding Authorities and Hubs from Link Structures on the World Wide Web." In *Proceedings of the 10th International Conference on World Wide Web*, pp. 415–429. New York: ACM Press, 2001.

[Brin and Page 98] S. Brin and L. Page. "The Anatomy of a Large-Scale Hypertextual Web Search Engine." *Computer Networks and ISDN Systems* 30:1–7 (1998), 107–117.

[Broder 97] A. Z. Broder. "On the Resemblance and Containment of Documents." In *Proceedings of the Compression and Complexity of Sequences 1997*, pp. 21–29. Los Alamitos, CA: IEEE Computer Society, 1997.

[Chen et al. 02] Y.-Y. Chen, Q. Gan, and T. Suel. "I/O-Efficient Techniques for Computing PageRank." In *Proceedings of the 11th International Conference on Information and Knowledge Management*, pp. 549–557. New York: ACM Press, 2002.

[Chirita et al. 04] Paul-Alexandru Chirita, D. Olmedilla, and W. Nejdl. "PROS: A Personalized Ranking Platform for Web Search." In *Adaptive Hypermedia and*

Adaptive Web-Based Systems: Third International Conference, AH 2004, Eindhoven, The Netherlands, August 23–26, 2004, Proceedings, pp. 34–43, Lecture Notes in Computer Science 3137. Berlin: Springer, 2004.

[Cohen 97] E. Cohen. "Size-Estimation Framework with Applications to Transitive Closure and Reachability." *J. Comput. Syst. Sci.* 55:3 (1997), 441–453.

[Dean and Ghemawat 04] 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)*, pp. 137–150. San Francisco: USENIX Association, 2004.

[Dwork et al. 01] C. Dwork, S. R. Kumar, M. Naor, and D. Sivakumar. "Rank Aggregation Methods for the Web." In *Proceedings of the 10th International Conference on World Wide Web*, pp. 613–622. New York: ACM Press, 2001.

[Eiron and McCurley 03] N. Eiron and K. S. McCurley. "Locality, Hierarchy, and Bidirectionality in the Web." Presented at Second Workshop on Algorithms and Models for the Web-Graph (WAW 2003). Available from World Wide Web (http://mccurley.org/papers/graph.pdf), 2003.

[Fagin et al. 03a] R. Fagin, R. Kumar, K. McCurley, J. Novak, D. Sivakumar, J. Tomlin, and D. Williamson. "Searching the Workplace Web." In *Proceedings of the 12th International Conference on World Wide Web Conference*, pp. 366–375. New York: ACM Press, 2003.

[Fagin et al. 03b] R. Fagin, R. Kumar, and D. Sivakumar. "Comparing Top k Lists." In *Proceedings of the Fourteenth ACM-SIAM Symposium on Discrete Algorithms*, pp. 26–36, Philadelphia: SIAM, 2003. Full version in *SIAM Journal on Discrete Mathematics* 17:1 (2003), 134–160

[Fagin et al. 04] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee. "Comparing and Aggregating Rankings with Ties." In *Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pp. 47–58. New York: ACM Press, 2004.

[Fogaras 03] D. Fogaras. "Where to Start Browsing the Web?" In *Innovative Internet Community Systems: Third International Workshop IICS 2003, Leipzig, Germany, June 19–21, 2003, Revised Papers* , pp. 65–79, Lecture Notes in Computer Science 2877. Berlin: Springer, 2003.

[Fogaras and Rácz 04] D. Fogaras and B. Rácz. "A Scalable Randomized Method to Compute Link-Based Similarity Rank on the Web Graph." In *Current Trends in Database Technology—EDBT 2004 Workshops: EDBT 2004 Workshops PhD, Data X, PIM, P2P&DB, and ClustWeb, Heraklion, Crete, Greece, March 14–18, 2004, Revised Selected Papers,* edited by W. Lindner et al., pp. 557–565, Lecture Notes in Computer Science 3268. Berlin: Springer, 2004.

[Fogaras and Rácz 05] D. Fogaras and B. Rácz. "Scaling Link-Based Similarity Search." In *Proceedings of the 14th International Conferece on World Wide Web*, pp. 641–650. New York: ACM Press, 2005.

[Haveliwala 02] T. H. Haveliwala. "Topic-Sensitive PageRank." In *Proceedings of the 11th International Conference on World Wide Web*, pp. 517–526. New York: ACM Press, 2002.

[Haveliwala 03] T. H. Haveliwala. "Efficient Encodings for Document Ranking Vectors." In *Proceedings of the International Conference on Internet Computing, IC '03, Las Vegas, Nevada, USA, June 23–26, 2003*, Vol. 1, pp. 3–9. Las Vegas: CSREA Press, 2003.

[Haveliwala et al. 03] T. H. Haveliwala, S. Kamvar, and G. Jeh. "An Analytical Comparison of Approaches to Personalizing PageRank." Technical report, Stanford University, 2003.

[Henzinger et al. 99a] M. R. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork. "Measuring Index Quality Using Random Walks on the Web." *Computer Networks* 31:11–16 (1999), 1291–1303.

[Henzinger et al. 99b] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. "Computing on Data Streams." In *External Memory Algorithms*, edited by J. M. Abello and J. S. Vitter, pp. 107–118. Providence, RI: American Mathematical Society, 1999.

[Henzinger et al. 00] M. R. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork. "On Near-Uniform URL Sampling." In *Proceedings of the Ninth International World Wide Web Conference*, pp. 295–308. Amsterdam: North-Holland Publishing Co., 2000.

[Hirai et al. 00] J. Hirai, S. Raghavan, H. Garcia-Molina, and A. Paepcke. "WebBase: A Repository of Web Pages." In *Proceedings of the Ninth International World Wide Web Conference*, pp. 277–293. Amsterdam: North-Holland Publishing Co., 2000.

[Jeh and Widom 03] G. Jeh and J. Widom. "Scaling Personalized Web Search." In *Proceedings of the 12th International Conference World Wide Web*, pp. 271–279. New York: ACM Press, 2003.

[Kamvar et al. 03a] 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.

[Kamvar et al. 03b] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub. "Extrapolation Methods for Accelerating PageRank Computations." In *Proceedings of the 12th International Conference on World Wide Web*, pp. 261–270. New York: ACM Press, 2003.

[Kendall 55] M. G. Kendall. *Rank Correlation Methods*. New York: Hafner Publishing Co., 1955.

[Kleinberg 99] J. Kleinberg. "Authoritative Sources in a Hyperlinked Environment." *Journal of the ACM* 46:5 (1999), 604–632.

[Kushilevitz and Nisan 97] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge, UK: Cambridge University Press, 1997.

[Lempel and Moran 05] R. Lempel and S. Moran. "Rank Stability and Rank Similarity of Link-Based Web Ranking Algorithms in Authority-Connected Graphs." *Information Retrieval* 8:2 (2005), 245–264.

[Netscape 05] Netscape Communication Corporation. "dmoz Open Directory Project." Available from World Wide Web (http://www.dmoz.org), 2005.

[Page et al. 98] 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.

[Palmer et al. 02] C. R. Palmer, P. B. Gibbons, and C. Faloutsos. "ANF: A Fast and Scalable Tool for Data Mining in Massive Graphs." In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 81–90. New York: ACM Press, 2002.

[Rusmevichientong et al. 01] P. Rusmevichientong, D. M. Pennock, S. Lawrence, and C. L. Giles. "Methods for Sampling Pages Uniformly from the World Wide Web." In *Using Uncertainty Within Computation: Papers from the AAAI Fall Symposium*, pp. 121–128. Menlo Park, CA: AAAI Press, 2001.

[Sarlós et al. 05] T. Sarlós, András A. Benczúr, K. Csalogány, D. Fogaras, and B. Rácz. "To Randomize or Not To Randomize: Space Optimal Summaries for Hyperlink Analysis." Technical Report, MTA SZTAKI, November 2005. Available from World Wide Web (http://www.ilab.sztaki.hu/websearch/Publications/).

[Singitham et al. 04] P. K. C. Singitham, M. S. Mahabhashyam, and P. Raghavan. "Efficiency-Quality Tradeoffs for Vector Score Aggregation." In *Proceedings of the Thirtieth International Conference on Very Large Data Bases*, pp. 624–635. San Francisco: Morgan Kaufmann, 2004.

[Suel and Shkapenyuk 02] T. Suel and V. Shkapenyuk. "Design and Implementation of a High-Performance Distributed Web Crawler." In *Proceedings of the 18th IEEE International Conference on Data Engineering*, pp. 357–368. Los Alamitos, CA: IEEE Computer Society, 2002.

Dániel Fogaras, Informatics Laboratory, MTA SZTAKI (Computer and Automation Research Institute of the Hungarian Academy of Sciences), Lágymányosi u. 11., Budapest, H-1111, Hungary (fd+im@cs.bme.hu)

Balázs Rácz, Informatics Laboratory, MTA SZTAKI (Computer and Automation Research Institute of the Hungarian Academy of Sciences), Lágymányosi u. 11., Budapest, H-1111, Hungary (bracz+p91@math.bme.hu)

Károly Csalogány, Informatics Laboratory, MTA SZTAKI (Computer and Automation Research Institute of the Hungarian Academy of Sciences), Lágymányosi u. 11., Budapest, H-1111, Hungary (cskaresz@ilab.sztaki.hu)

Tamás Sarlós, Informatics Laboratory, MTA SZTAKI (Computer and Automation Research Institute of the Hungarian Academy of Sciences), Lágymányosi u. 11., Budapest, H-1111, Hungary (stamas@ilab.sztaki.hu)

Received November 1, 2004; accepted August 31, 2005.