

# Towards Scaling Fully Personalized PageRank <sup>\*</sup>

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

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

<sup>2</sup> Budapest University of Technology and Economics  
fd@cs.bme.hu, bracz+p31@math.bme.hu

**Abstract** Personalized PageRank expresses backlink-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 page selection. In this paper we achieve full personalization by a novel algorithm that computes a compact database of simulated random walks; this database can serve arbitrary personal choices of small subsets of web pages. 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 exact personalized PageRank values can only be obtained from a database of quadratic size.

## 1 Introduction

The idea of topic sensitive or personalized ranking appears since the beginning of the success story of Google's PageRank [5,23] and other hyperlink-based centrality measures [20,4]. Topic sensitivity is either achieved by precomputing modified measures over the entire Web [13] or by ranking the neighborhood of pages containing the query word [20]. 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 of personalized PageRank [13,18]. 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) [23] 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.

We introduce a novel scalable Monte Carlo algorithm for PPR that precomputes a compact database. As described in Section 2, the database contains simulated random walks, and PPR is estimated on-line with a limited number of database accesses. Earlier algorithms [14] restricted personalization to a few topics, a subset of popular pages or to hosts; our algorithm on the other hand enables personalization for *any* small set of pages. Query time is linear in the number of pages with non-zero personalization. Similar statement holds for the previous approaches, too.

---

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

The price that we pay for full personalization is that our algorithm is randomized and less precise; the formal analysis of the error probability is discussed in Section 3. In Section 4 it is verified that we have to pay the price of approximation by showing that full personalization requires a database of  $\Omega(V^2)$  bits over a graph with  $V$  vertices.

Though this approximation approach might fail or need longer query time in certain cases (for example for pages with large neighborhoods), the available personalization algorithms can be combined to resolve these issues. For example we can precompute personalization vectors for topics (topic-sensitive PR), popular pages with large neighborhoods (use [18]), some often requested combination of popular pages (sub-topics), and use our algorithm for those many pages not covered so far. This combination gives adequate precision for most queries with large flexibility for personalization.

**Related results.** The possibility of personalization was first mentioned in [5,23] together with PageRank. Linearity of PPR [13] implies that if PPR is precomputed for some preference vectors, then PPR can be calculated on-line for any linear combination of the preference vectors by combining the precomputed PPRs. Thus personalization was achieved for any combination of 16 basic topics in [13]; an experimental implementation is already available at [12]. The methods of [18] precompute PPR for at most 100.000 individual pages, and then for any subset of the individual pages personalization is available by linearity. Furthermore, the algorithm of [19] personalizes PageRank over hosts rather than single web pages. Instead of user preferences, [25] tunes PageRank automatically using the query keywords.

To the best of our knowledge randomized algorithms are not very common in the link-mining community. A remarkable exception [24] applies probabilistic counting to estimate the neighborhood function of web pages. Besides link-mining the paper [8] estimates the size of transitive closure for massive graphs occurring in databases. For text-mining algorithms [6] estimates the resemblance and containment of documents with a sampling technique.

Random walks were used before to compute various web statistics, mostly focused on sampling the web (uniformly or according to static PR) [16,26,1,15], but also for calculating page decay [2] and similarity values [11].

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 [17] for the space complexity of several data stream graph algorithms.

**Preliminaries.** In this section we briefly 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 webgraph with normalized rows and  $c \in (0, 1)$  the *teleportation constant*. 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 [23]

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

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* denoted by  $\text{PPV}(\mathbf{r})$ . The special case when for some page  $u$  the  $u^{\text{th}}$  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  $\text{PPV}(u)$ . Furthermore the  $v^{\text{th}}$  coordinate of  $\text{PPV}(u)$  will be denoted by  $\text{PPV}(u, v)$ .

**Theorem 1 (Linearity, [13]).** *For any preference vectors  $\mathbf{r}_1, \mathbf{r}_2$ , and positive constants  $\alpha_1, \alpha_2$  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 last statement of the introduction will play a central role in our PPV estimations. The theorem provides an alternate probabilistic characterization of individual PageRank scores.<sup>3</sup>

**Theorem 2 ([18,10]).** *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  $\text{PPV}(u, v) = \Pr\{\text{the random walk ends at page } v\}$ .*

## 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.

**Definition 1 (Fingerprint).** *A fingerprint 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 every step the walk ends with probability  $c$ , and takes a further step with probability  $1 - c$ .*

By Theorem 2 the ending vertex of a fingerprint, as a random variable, has the distribution of the personalized PageRank vector of  $u$ . 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. The ending vertices of the fingerprints will constitute the *index database*, and the output ranking will be computed at query time from the fingerprints of positive personalization weights using the linearity theorem.

To increase the precision of the approximation of  $\text{PPV}(u)$  we will use the fingerprints of  $u$ 's neighbors in the calculation, as described in Section 2.3.

<sup>3</sup> Notice that this characterization slightly differs from the random surfer formulation [23] of PageRank.

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 page, and we will count the database scans and total I/O size as the efficiency measure of our algorithms. The assumption is made, since even with the latest compression techniques [3] it does not seem plausible to store the entire web graph in main memory. 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 (external memory algorithm) and a distributed system with tens to thousands of medium capacity computers. Both algorithms use similar techniques to the respective I/O efficient algorithms computing PageRank [7].

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 due to the different approach.

## 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, as we are interested in adding the ending vertex to the index of the starting vertex. 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 we can generate the random out-neighbor (the next vertex) of each partial fingerprint. For each partial fingerprint 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 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

---

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

---

$V$  is the number of vertices in the web graph,  $N$  is the required number of fingerprints for each vertex. For a vertex  $v$ ,  $\text{OutEdges}[v]$  is the set of links on page  $v$ . The teleportation constant of PPR is  $c$ .

```
for  $u := 1$  to  $V$  do
  for  $i := 1$  to  $N$  do
    Paths[( $u - 1$ )  $\cdot N + i$ ].PathStart:= $u$ 
    Paths[( $u - 1$ )  $\cdot N + i$ ].PathEnd:= $u$ 
    Fingerprint[ $u$ ]:=∅
  while Paths  $\neq$  ∅ do
    sort Paths by PathEnd /*use an external memory sort*/
    for  $j := 1$  to Paths.length do /*simultaneous scan of OutEdges and Paths*/
       $l := \text{random}(\text{OutEdges}[\text{Paths}[j].\text{PathEnd}].\text{length})$  /*choose an edge at random*/
      Paths[ $j$ ].PathEnd:=OutEdges[Paths[ $j$ ].PathEnd][ $l$ ] /*prolong the path*/
      if random() $<c$  then /*this fingerprint ends here*/
        Fingerprint[Paths[ $j$ ].PathStart].push(Paths[ $j$ ].PathEnd)
        Paths.delete( $j$ )
```

---

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.<sup>4</sup>

An other 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 (Path-Start, PathEnd) pairs: one (large) input queue, and for each computer one small output queue<sup>5</sup>.

---

<sup>4</sup> 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.

<sup>5</sup> Preferably the size of a network packet.

---

**Algorithm 2** Indexing (distributed computing method)

---

The algorithm of one participating computer. Each computer is assumed to have a part of the OutEdges[] arrays, in 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: (PathStart, PathEnd).

```
for  $v$  s.t.  $\text{part}(v) = \text{current computer}$  do
  InQueue.push( $(v, v)$ )
while at least one queue is not empty do /*some of the fingerprints are still being calculated*/
   $p := \text{InQueue.get()}$  /*If empty, flush output queues and block until a packet arrives.*/
   $l := \text{random}(\text{OutEdges}[p.\text{PathEnd}].\text{length})$  /*choose an edge at random*/
   $q.\text{PathEnd} := \text{OutEdges}[q.\text{PathEnd}][l]$  /*prolong the path*/
  if  $\text{random}() < c$  then /*teleport: this fingerprint ends here*/
    Fingerprint[ $q.\text{PathStart}$ ].push( $q.\text{PathEnd}$ )
  else
     $o := \text{part}(q.\text{PathEnd})$ 
    OutQueue[ $o$ ].push( $p$ )
    if OutQueue[ $o$ ].length  $\geq$  reasonable packet size then
      transmit OutQueue[ $o$ ] to the InQueue of computer  $o$ .
  transmit the finished fingerprints to the proper computers for collecting and sorting.
```

---

The computation starts with each computer filling their own input queue with  $N$  copies of the initial partial fingerprints  $(v, v)$ , for each vertex  $v$  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.<sup>6</sup> If the 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.

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 total 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. As the web graph tends to have many 'local' links, with a suitable partition of vertices<sup>7</sup> the network transfer will be considerably less. Also note that this amount of transfer is distributed almost uniformly between all pairs of network nodes<sup>8</sup>, so the effective switching capacity of the network is challenged, not the capacity of the individual network links.

---

<sup>6</sup> 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.

<sup>7</sup> It should be enough to have each domain on a single computer, as the majority of the links are intra-domain links [19,9].

<sup>8</sup> Also depending on the actual partition; as a heuristics one should use a partition that distributes the global PageRank uniformly across computers: the expected value of the total InQueue hits of a computer is proportional to the the total PageRank score of vertices belonging to that computer.

### 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 [18]:

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

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 fingerprint endings for  $u$ , the set of out-edges of  $u$ , and the fingerprint endings for the vertices linked by  $u$ .<sup>9</sup> From this set of fingerprints we use the above equation to approximate  $\text{PPV}(u)$  using a higher amount of samples, thus achieving higher precision. This is a tradeoff between query time (database accesses) and precision: with  $k$  database accesses we can approximate the vector from  $kN$  samples.

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  $(1 - c)^L$  to the approximation and drop all fingerprints longer than  $L$ . This would also solve the problem of the approximated individual PPV vectors having many zeroes (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  $\widehat{\text{PPV}}(u)$  converges to the actual personalized PageRank vector  $\text{PPV}(u)$ . To show that the rate of convergence is exponential, recall that each fingerprint of  $u$

---

<sup>9</sup> We can iterate this recursion if we want to have even more samples.

ends at  $v$  with probability  $\text{PPV}(u, v)$ , where  $\text{PPV}(u, v)$  denotes the  $v^{\text{th}}$  coordinate of  $\text{PPV}(u)$ . Therefore  $N \cdot \widehat{\text{PPV}}(u, v)$ , the number of fingerprints of  $u$  that ends at  $v$ , has binomial distribution with parameters  $N$  and  $\text{PPV}(u, v)$ . Then Chernoff's inequality yields the following bound on the error of over-estimating  $\text{PPV}(u, v)$  and the same bound holds for under-estimation:

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

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  $\text{PPV}$  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  $\text{Var}(Z) = \text{PPV}(u, v) + \text{PPV}(u, w) - (\text{PPV}(u, v) - \text{PPV}(u, w))^2 \leq \text{PPV}(u, v) + \text{PPV}(u, w)$ :

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

From the above inequality both theorems follow.  $\square$

The first theorem shows that even a modest amount of fingerprints are enough to make distinction 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 deep problem of PageRank itself, as [21] showed that PageRank is unstable around the low ranked pages, in the sense that with little 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  $\epsilon$  and  $\delta$ .

## 4 Lower Bounds for PPR Database Size

In this section we will prove several lower bounds on the complexity of personalized PageRank. In particular, we will prove that exact computation the necessary index database size of a fully personalized PageRank must be at least  $\Omega(V^2)$  bits, and if personalizing only for  $H$  pages, the database size is at least  $\Omega(H \cdot V)$ . In the approximate problem 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 edge set of 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 only on the index database. In this model we will lower bound the index database size. We will consider the following types of queries:

- (1) Exact: Calculate  $\text{PPV}(u, v)$ , the  $v^{\text{th}}$  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  $\epsilon$ .
- (4) Comparison: Decide in which order  $v$  and  $w$  are in the personalized rank of  $u$  with error probability at most  $\epsilon$ .
- (5)  $\epsilon$ - $\delta$  comparison: For a fixed  $\epsilon > 0, \delta > 0$  decide the comparison problem with error probability at most  $\epsilon$ , 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* [17]: there are two players  $A$  and  $B$ ,  $A$  has an  $m$ -bit vector  $x$ ,  $B$  has an index  $y \in \{1, 2, \dots, m\}$ , and they have to compute the function  $f(x, y) = x_y$ , i.e. the output is the  $y^{\text{th}}$  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* [22] of this function is the required bits of transfer in the worst case for the best protocol.

**Theorem 5 ([17]).** *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  $\gamma m$  bits.*

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. In the case of full personalization  $H = V$ .

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

*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 index on this graph, and transmit this index to  $B$ . Then Player  $B$  will use the positivity query algorithm for some vertices (depending on

the requested index  $y$ ) such that the answer to the positivity query will be the  $y^{\text{th}}$  bit of the input vector  $x$ . Thus if the algorithm solves the PPV indexing and positivity query with error probability  $\epsilon$ , 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  $\gamma m$ .

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. 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 index  $1 \leq y \leq m$  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 indexed, 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 sized  $\Omega(H \cdot V)$  bits.*

**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. For smaller  $\delta$  the index database requires  $\Omega((1-2\epsilon)HV)$  bits.*

*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  $2\delta$ , so that the approximation will decide the positivity problem, too. If the  $u_i$  vertex in the input graph of our construction has  $k$  edges connected to it, then each of those  $v_j$  end-vertices will have exactly  $\frac{1-c}{k}$  weight in  $\text{PPV}(u_i)$ . For this to be over  $2\delta$  we can have at most  $n = \frac{1-c}{2\delta}$  possible  $v_1, \dots, v_n$  vertices. With  $\frac{1+\gamma}{2} = 1 - \epsilon$  the theorem follows.

For small  $\delta$  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  $\epsilon, \delta$  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.*

*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 in the graph construction  $w_1, \dots, w_n$ , 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 put into, 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  $\epsilon$ - $\delta$  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. For smaller  $\delta$  the index database needs  $\Omega((1 - 2\epsilon)HV)$  bits.

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

## 5 Conclusions

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 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.

## Acknowledgement

We wish to acknowledge András Benczúr, Katalin Friedl and Tamás Sarlós for several discussions and comments on this research. Furthermore we are grateful to Glen Jeh for encouraging us to consider Monte-Carlo PPR algorithms.

## References

1. 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.

8. E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55(3):441–453, 1997.
9. 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.
10. 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.
11. D. Fogaras and B. Rácz. A scalable randomized method to compute link-based similarity rank on the web graph. In *Clustering Information over the Web, International Workshop in conjunction with EDBT04*, 2004.
12. P. Google. <http://labs.google.com/personalized>.
13. T. H. Haveliwala. Topic-sensitive PageRank. In *Proceedings of the 11th World Wide Web Conference (WWW)*, Honolulu, Hawaii, 2002.
14. T. H. Haveliwala, S. Kamvar, and G. Jeh. An analytical comparison of approaches to personalizing PageRank. Technical report, Stanford University, 2003.
15. 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.
16. 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.
17. M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. *External memory algorithms*, pages 107–118, 1999.
18. 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.
19. 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.
20. J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.
21. 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.
22. N. Nisan and E. Kushilevitz. *Communication Complexity*. Cambridge University Press, 1997.
23. 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.
24. 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.
25. M. Richardson and P. Domingos. The Intelligent Surfer: Probabilistic combination of link and content information in PageRank. *Advances in Neural Information Processing Systems*, 14:1441–1448, 2002.
26. 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.