To Randomize or Not To Randomize: Space Optimal Summaries for Hyperlink Analysis[*]

Tamás Sarlós[1,2] András A. Benczúr[1,2] Károly Csalogány[1,2] Dániel Fogaras[1,3] Balázs Rácz[1,3]

[1] Computer and Automation Research Institute, Hungarian Academy of Sciences (MTA SZTAKI)
[2] Eötvös University, Budapest [3] Budapest University of Technology and Economics
{stamas,benczur,cskaresz,fd,bracz+ps72}@ilab.sztaki.hu

ABSTRACT

Personalized PageRank expresses link-based page quality around user selected pages. The only previous personalized PageRank algorithm that can serve on-line queries for an unrestricted choice of pages on large graphs is our Monte Carlo algorithm [WAW 2004]. In this paper we achieve unrestricted personalization by combining rounding and randomized sketching techniques in the dynamic programming algorithm of Jeh and Widom [WWW 2003]. We evaluate the precision of approximation experimentally on large scale real-world data and find significant improvement over previous results. As a key theoretical contribution we show that our algorithms use an optimal amount of space by also improving earlier asymptotic worst-case lower bounds. Our lower bounds and algorithms apply to SimRank as well; of independent interest is the reduction of the SimRank computation to personalized PageRank.

Categories and Subject Descriptors

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

General Terms

Algorithms, Theory, Experimentation

Keywords

link-analysis, similarity search, scalability, data streams

1. Introduction

The idea of using hyperlink mining algorithms in Web search engines appears since the beginning of the success of Google's PageRank [24]. Hyperlink based methods are based on the assumption that *a hyperlink $u \to v$ implies that page u votes for v as a quality page.* In this paper we address the computational issues [13, 17, 11, 12] of personalized PageRank [24] and SimRank [16].

Personalized PageRank (PPR) [24] enters user preferences by assigning more importance to the neighborhood of pages at the user's selection. Jeh and Widom [16] introduced *SimRank*, the multi-step link-based similarity function with the recursive idea that *two pages are similar if pointed to by similar pages*. Notice that both measures are hard to compute over massive graphs: naive personalization would require on the fly power iteration over the entire graph for a user query; naive SimRank computation would require power iteration over all pairs of vertices.

We give algorithms with provable performance guarantees based on computation with sketches [7] as well as simple deterministic summaries; see Table 1 for a comparison of our methods with previous approaches. We may personalize to any single page from which arbitrary page set personalization follows by linearity [13]. Similarly, by our SimRank algorithm we may compute the similarity of any two pages or the similarity top list of any single page. Motivated by search engine applications, we give two-phase algorithms that first compute a compact database from which value or top list queries can be answered with a low number of accesses. Our key results are summarized as follows:

- We give practical methods for serving unrestricted on-line personalized PageRank (Section 2.1) as well as SimRank queries with space a reasonable constant per vertex (Section 3). The methods are based on deterministic rounding.
- We give a theoretically optimal algorithm for personalized PageRank value queries (Section 2.2) based on randomized sketching. Given an additive error ϵ and the probability δ of an incorrect result, we improve the disk usage bound from $n \log n \log(1/\delta)/\epsilon^2$ [11, 12] to $n \log(1/\delta)/\epsilon$.
- We give theoretically optimal algorithms for SimRank value and top list queries (Section 3.1) by a nontrivial reduction of SimRank to personalized PageRank.
- We improve the communication complexity based lower bounds of [11, 12] for the size of the database (Section 4); our bounds are matched by our algorithms. Our sketch-based algorithms use optimal space; surprisingly for top list queries deterministic rounding is already optimal in itself.
- In Section 5 we experimentally analyze the precision of approximation over the Stanford WebBase graph and conclude that our summaries provide better approximation for the top personalized PageRank scores than previous methods.

[*] This work was supported by the Mobile Innovation Center, Hungary and Inter-University Center for Telecommunication and Informatics (ETIK). This is an abbreviated version of the full paper available at http://www.ilab.sztaki.hu/websearch/Publications/.

WWW 2006, May 23–26, 2006, Edinburgh, Scotland.
ACM 1-59593-323-9/06/0005.

	Storage requirement	Personalization
Topic sensitive [13]	$O(t \cdot n)$ words	t topics $\delta = \epsilon = 0$
Hub decomposition [17]	$\Omega(h^2), O(h \cdot n)$ words	h pages $\delta = \epsilon = 0$
Monte Carlo [11]	$O(n \cdot \log n \cdot \frac{1}{\epsilon^2} \cdot \log \frac{1}{\delta})$ bits	arbitrary
Rounding	$O(n \cdot \log n \cdot \frac{1}{\epsilon})$ bits*	arbitrary $\delta = 0$
Sketching	$O(n \cdot \log \frac{1}{\delta} \cdot \frac{1}{\epsilon})$ bits**	arbitrary

$*$ optimal for top queries

$**$ optimal for value queries

Table 1: Comparison of personalized PageRank algorithms for graphs of n vertices, additive error ϵ and error probability δ.

1.1 Related Results

The scalable computation of personalized PageRank was addressed by several papers [13, 18, 17] that gradually increase the choice for personalization. By Haveliwala's method [13] we may personalize to the combination of 16 topics extracted from the Open Directory Project. The BlockRank algorithm of Kamvar et al. [18] speeds up personalization to the combination of hosts. The state of the art Hub Decomposition algorithm of Jeh and Widom [17] computed and encoded personalization vectors for approximately 100K personalization pages.

To the best of our knowledge, the only scalable personalized PageRank algorithm that supports the unrestricted choice of the teleportation vector is the Monte Carlo method of [11]. This algorithm samples the personalized PageRank distribution of each page simultaneously during the precomputation phase, and estimates the personalized PageRank scores from the samples at query time. The drawback of the sampling approach is that approximate scores are returned, where the error of approximation depends on the random choice. In addition the bounds involve the unknown variance, which can in theory be as large as $\Omega(1)$, and hence we need $\Theta(\frac{1}{\epsilon^2}\log 1/\delta)$ random samples. Indeed a matching sampling complexity lower bound for telling binomial distributions with means $1/2 \pm \epsilon$ apart [1] indicates that one can not reduce the number of samples when approximating personalized PageRank. Similar findings of the superiority of summarization or sketching over sampling is described in [5]. The algorithms presented in Section 2 outperform the Monte Carlo method by significantly reducing the error.

We also address the computational issues of SimRank, a link-based similarity function introduced by Jeh and Widom [16]. The power iteration SimRank algorithm of [16] is not scalable since it iterates on a quadratic number of values, one for each pair of Web pages; in [16] experiments on graphs with no more than 300K vertices are reported. Analogously to personalized PageRank, the scalable computation of SimRank was first achieved by sampling [12]. Our new SimRank approximation algorithms presented in Section 3 improve the precision of computation.

The key idea of our algorithms is that we use lossy representation of large vectors either by *rounding* or *sketching*. Sketches are compact randomized data structures that enable approximate computation in low dimension. To be more precise, we adapt the *Count-Min Sketch* of Cormode and Muthukrishnan [7], which was primarily introduced for data stream computation. We use sketches for small space computation; in the same spirit Palmer et al. [25] apply probabilistic counting sketches to approximate the sizes of neighborhoods of vertices in large graphs. Further sketching techniques for data streams are surveyed in [23]. Lastly we mention that Count-Min Sketch and the historically first sketch, the Bloom filter [2] stem from the same idea; we refer to the detailed survey [4] for further variations and applications.

Surprisingly, it turns out that sketches do not help if the top t highest ranked or most similar nodes are queried; the deterministic version of our algorithms show the same performance as the randomized without even allowing a small probability of returning a value beyond the error bound. Here the novelty is the optimal performance of the deterministic method; the top t problem is known to cause difficulties in sketch-based methods and always increases sketch sizes by a factor of $\Omega(\log n)$. By using $\Omega(\log n)$ times larger space we may use a binary search structure or we may use p sketches accessed $n^{1/p}$ times per query [7]. Note that $n^{1/p}$ queries require an error probability of $O(\delta/n^p)$ that again increase sketch sizes by a factor of $\Omega(\log n)$.

In Section 4 we show that our algorithms build optimal sized databases. To obtain lower bounds on the database size, we apply communication complexity techniques that are commonly used for space lower bounds [21]. Our reductions are somewhat analogous to those applied by Henzinger et al. [14] for space lower bounds on stream graph computation.

1.2 Preliminaries

We briefly introduce notation, and recall definitions and basic facts about PageRank, SimRank and the Count-Min sketch.

Personalized PageRank

Let us consider the web as a graph. Let n denote the number of vertices and m the number edges. Let $d^+(v)$ and $d^-(v)$ denote the number of edges leaving and entering v, respectively. Details of handling nodes with $d^+(v) = 0$ and $d^-(v) = 0$ are omitted.

In [24] the *PageRank* vector $p = (p(1), \ldots, p(n))$ is defined as the solution of the following equation $p(u) = c \cdot r(u) + (1 - c) \cdot \sum_{v:(vu) \in E} p(v)/d^+(v)$, where $r = (r(1), \ldots, r(n))$ is the teleportation vector and c is the teleportation probability with a typical value of $c \approx 0.15$. If r is uniform, i.e. $r(u) = 1/n$ for all u, then p is the PageRank. For non-uniform r the solution p is called *personalized PageRank*; we denote it by PPR_r. Since PPR_r is linear in r [13, 17], it can be computed by linear combination of personalization to single points v, i.e. to vectors $r = \chi_v$ consisting of all 0 except for node v where $\chi_v(v) = 1$. Let $\mathrm{PPR}_v = \mathrm{PPR}_{\chi_v}$.

An alternative characterization of $\mathrm{PPR}_u(v)$ [10, 17] is based on the probability that a length k random walk starting at node u ends in node v. We obtain $\mathrm{PPR}_u(v)$ by choosing k random according to the geometric distribution:

$$\mathrm{PPR}_u^{[k]}(v) = \sum_{v_0 = u, v_1, \ldots, v_k = v} 1/(d^+(v_0) \cdots d^+(v_{k-1})); \tag{1}$$

the summation is along walks starting at u and ending in v. Thus

$$\mathrm{PPR}_u(v) = \sum_{k'=0}^{\infty} c(1-c)^{k'} \mathrm{PPR}_u^{[k']}(v). \tag{2}$$

Similarly we get $\mathrm{PPR}_u^{(k)}$ if we sum up only to k instead of ∞. An equivalent reformulation of the path summing formula (2) is the Decomposition Theorem proved by Jeh and Widom [17]:

$$\mathrm{PPR}_u = c\chi_u + (1-c) \cdot \sum_{v:(uv) \in E} \mathrm{PPR}_v / d^+(u). \tag{3}$$

The Decomposition Theorem immediately gives rise to the Dynamic Programming approach [17] to compute personalized Page-

Rank that performs iterations for $k = 1, 2, \ldots$ with $\mathrm{PPR}_u^{(0)} = c \cdot \chi_u$:

$$\mathrm{PPR}_u^{(k)} = c\chi_u + (1-c) \cdot \sum_{v:(uv)\in E} \mathrm{PPR}_v^{(k-1)}/d^+(u). \quad (4)$$

SimRank

Jeh and Widom [16] define SimRank by the following equation very similar to the PageRank power iteration such that $\mathrm{Sim}^{(0)}(u_1, u_2) = \chi_{u_1}(u_2)$ and

$$\mathrm{Sim}^{(k)}(u_1,u_2) = \begin{cases} (1-c) \cdot \frac{\sum \mathrm{Sim}^{(k-1)}(v_1,v_2)}{d^-(u_1)\cdot d^-(u_2)} & \text{if } u_1 \neq u_2 \\ 1 & \text{if } u_1 = u_2. \end{cases} \quad (5)$$

where summation is for $v_1, v_2 : (v_1u_1), (v_2u_2) \in E$.

Count-Min Sketch

The *Count-Min Sketch* [7] is a compact randomized approximate representation of non-negative vector $x = (x_1, \ldots, x_n)$ such that a single value x_j can be queried with a fixed additive error $\epsilon > 0$ and a probability $\delta > 0$ of returning a value out of this bound. The representation is a table of *depth* $d = \ln 1/\delta$ and *width* $w = e/\epsilon$. One row C of the table is computed with a random hash function $h : \{1, \ldots, n\} \to \{1, \ldots, w\}$. The i^{th} entry of the row C is defined as $C_i = \sum_{j:h(j)=i} x_j$. Then the Count-Min sketch table of x consists of d such rows with hash functions chosen uniformly at random from a pairwise-independent family.

THEOREM 1 (CORMODE, MUTHUKRISHNAN [7]). *Let* $\hat{x}_j = \min_C\{C_{h(j)}\}$ *where the minimum is taken over the* d *rows of the table. Then* $\hat{x}_j \geq x_j$ *and Prob*$(\hat{x}_j > x_j + \epsilon||x||_1) \leq \delta$ *hold.*

Count-Min sketches are based on the principle that any randomized approximate computation with one sided error and bias ϵ' can be turned into an algorithm that has guaranteed error at most $e \cdot \epsilon'$ with probability $1 - \delta$ by running $\ln 1/\delta$ parallel copies and taking the minimum. The proof simply follows from Markov's inequality and is described for the special cases of sketch value and inner product in the proofs of Theorems 1 and 2 of [7], respectively.

2. Personalized PageRank

We give two efficient realizations of the dynamic programming algorithm of Jeh and Widom [17]. Our algorithms are based on the idea that if we use an approximation for the partial values in certain iteration, the error will not aggregate when summing over out-edges, instead the error of previous iterations will decay with the power of $1 - c$. Our first algorithm in Section 2.1 uses certain deterministic rounding optimized for smallest runtime for a given error, while our second algorithm in Section 2.2 is based on Count-Min sketches [7].

The original implementation of dynamic programming [17] relies on the observation that in the first k iterations of dynamic programming only vertices within distance k have non-zero value. However, the rapid expansion of the k-neighborhoods increases disk requirement close to n^2 after a few iterations, which limits the usability of this approach[1]. Furthermore, an external memory implementation would require significant additional disk space.

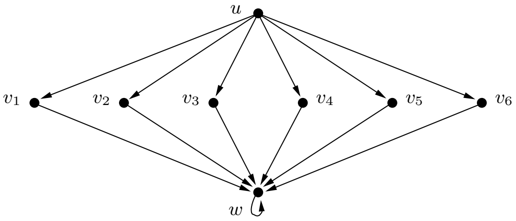

Figure 1: A simple example showing the superiority of dynamic programming over power iterations for small space computations.

We may justify why dynamic programming is the right choice for small-space computation by comparing dynamic programming to power iteration over the graph of Fig. 1. When computing $\mathrm{PPR}_u(w)$, power iteration moves top-down, starting at u, stepping into its neighbors $v_1, v_2, \ldots$ and finally adding up all their values at w. Hence when approximating, we accumulate all error when entering the large in-degree node w and in particular we must compute $\mathrm{PPR}_u(v_i)$ values fairly exact. Dynamic programming, in contrast, moves bottom up by computing the trivial PPR_w vector, then all the PPR_{v_i}, then finally averages all of them into PPR_u. Because of averaging we do not amplify error at large in-degrees; even better by looking at (4) we notice that the effect of earlier steps diminishes exponentially in $(1 - c)$. In particular even if there are edges entering u from further nodes, we may safely discard all the small $\mathrm{PPR}_u(v_i)$ values for further computations, thus saving space over power iteration where we require the majority of these values in order to compute $\mathrm{PPR}_u(w)$ with little error.

We measure the performance of our algorithms in the sense of intermediate disk space usage. Notice that our algorithms are *two-phase* in that they preprocess the graph to a compact database from which value and top list queries can be served real-time; preprocessing space and time is hence crucial for a search engine application. Surprisingly, in this sense rounding in itself yields an optimal algorithm for top list queries as shown by giving a matching lower bound in Section 4. The sketching algorithm further improves space usage by a factor of $\log n$ and is hence optimal for single value queries. For finding top lists, however, we need additional techniques such as binary searching as in [7] that loose the $\log n$ factor gain and use asymptotically the same amount of space as the deterministic algorithm. Since the deterministic rounding involves no probability of giving an incorrect answer, that algorithm is superior for top list queries.

The key to the efficiency of our algorithms is the use of small size approximate $\widehat{\mathrm{PPR}}_u^{(k)}(v)$ values obtained either by rounding and handling sparse vectors or by computing over sketches. In order to perform the update step of Algorithm 1 we must access all $\widehat{\mathrm{PPR}}_v$ vectors; the algorithm proceeds as if we were multiplying the weighted adjacency matrix $A_{uv} = 1/d^+(u)$ for $(uv) \in E$ with the vector $\{\widehat{\mathrm{PPR}}_u(w) : u \in V\}$ parallel for all values of w. We may use (semi)external memory algorithms [27]; efficiency will depend on the size of the description of the vectors.

The original algorithm of Jeh and Widom defined by equation (4) uses two vectors in the implementation. We remark that a single vector suffices since by using updated values within an iteration we only speed convergence up. A similar argument is given by McSherry [22] for the power iteration, however there the resulting sequential update procedure still requires two vectors.

[1] Over the publicly available Stanford WebBase graph [15] we computed an average over 1000 non-zeroes after 4 iterations; on average 24% of all vertices are reached within $k = 15$ steps.

Algorithm 1 PageRank by rounded Dynamic Programming

```
for each node $u$ do
    Initialize $\widehat{\mathrm{PPR}}_u$ by $\psi_0(c\chi_u)$
for $k := 1, \dots, k_{\max} = 2\log_{1-c}\epsilon$ do
    $\epsilon_k = \epsilon \cdot (1-c)^{-\frac{k_{\max}-k}{2}}$
    Define function $\psi_k$ that rounds down to multiple of $\epsilon_k$
    for each node $u$ do
        $\widehat{\mathrm{PPR}}_u \leftarrow \psi_k\Big(c\chi_u + (1-c) \cdot \sum_{v:(uv)\in E} \widehat{\mathrm{PPR}}_v / d^+(u)\Big)$
```

2.1 Rounding

In Algorithm 1 we compute the steps of the dynamic programming personalized PageRank algorithm (4) by rounding all values down to a multiple of the prescribed error value ϵ. As the sum of $\mathrm{PPR}_u(v)$ for all v equals one, the rounded non-zeroes can be stored in small space since there may be at most $1/\epsilon$ of them.

We improve on the trivial observation that there are at most $1/\epsilon$ rounded non-zero values in two ways as described in the next two theorems. First, we observe that the effect of early iterations decays as the power of $(1-c)$ in the iterations, allowing us to similarly increase the approximation error ϵ_k for early iterations k. We prove correctness in Theorem 2; later in Theorem 4 it turns out that this choice also weakens the dependency of the running time on the number of iterations. Second, we show that the size of the non-zeroes can be efficiently bit-encoded in small space; while this observation is less relevant for a practical implementation, this is key in giving an algorithm that matches the lower bound of Section 4.

THEOREM 2. *Algorithm 1 returns values between PPR$_u(v) - 2\epsilon/c$ and PPR$_u(v)$.*

PROOF. By induction on iteration k of Algorithm 1 we show a bound that is tighter for $k = k_{\max}$ than that of the Theorem:

$$|\mathrm{PPR}_u(w) - \widehat{\mathrm{PPR}}_u(w)| < \frac{1}{1-\sqrt{1-c}}\epsilon_k.$$

By the choice of ϵ and $k_{\max}$ we have $\epsilon_0 = 1$, thus the $k = 0$ case is immediate since $0 \le \mathrm{PPR}_u(v) \le 1$.

Since we use a single vector in the implementation, we may update a value by values that have themselves already been updated in iteration k. Nevertheless since $\epsilon_k = \sqrt{1-c} \cdot \epsilon_{k-1}$ and hence decreases in k, values that have earlier been updated in the current iteration in fact incur an error smaller than required on the right hand side of the update step of Algorithm 1. In order to distinguish values before and after a single step of the update, let us use $\widehat{\mathrm{PPR}}'$ to denote values on the right hand side. To prove, notice that by the Decomposition Theorem (3)

$$\mathrm{PPR}_u(w) - c\chi_u - (1-c) \cdot \sum_{v:(uv)\in E} \widehat{\mathrm{PPR}}'_v(w)/d^+(u)$$
$$= (1-c) \cdot \sum_{v:(uv)\in E} (\mathrm{PPR}_v(w) - \widehat{\mathrm{PPR}}'_v(w))/d^+(u)$$

As ψ_k introduces at most ϵ_k error, by the triangle inequality

$$|\mathrm{PPR}_u(w) - \widehat{\mathrm{PPR}}_u(w)|$$
$$\le \epsilon_k + (1-c) \cdot \sum_{v:(uv)\in E} |\mathrm{PPR}_v(w) - \widehat{\mathrm{PPR}}'_v(w)|/d^+(u).$$

Using the inductive hypothesis this leads us to

$$|\mathrm{PPR}_u(w) - \widehat{\mathrm{PPR}}_u(w)| < \epsilon_k + \frac{1-c}{1-\sqrt{1-c}} \cdot \epsilon_{k-1}$$
$$= \epsilon_k + \frac{1-c}{1-\sqrt{1-c}} \cdot \frac{\epsilon_k}{\sqrt{1-c}} = \frac{1}{1-\sqrt{1-c}} \cdot \epsilon_k,$$

completing the proof. □

Next we show that multiples of ϵ that sum up to 1 can be stored in $1/\epsilon$ bit space. For the exact result we need to select careful but simple encoding methods given in the trivial lemma below.

LEMMA 3. *Let z non-negative values be given, each a multiple of ϵ that sum up to at most 1. If we unary encode the values as multiples of ϵ and use a termination symbol, the encoding uses space $\frac{1}{\epsilon} + z$ bits. If we combine the same encoding with sparse vector storage by recording the position of non-zeroes in $\log z$ space each, we may encode the sequence by $\frac{1}{\epsilon} \cdot (1 + \log z)$ bits.* □

THEOREM 4. *Algorithm 1 runs in time of $O(m \log n/(c \cdot \epsilon))$ bit operations*[2] *and builds a database of size $n \cdot \frac{2}{\epsilon} \cdot \log n$ bits. In order to return the approximate value of PPR$_u(v)$ and the largest t elements of PPR$_u(.)$, we may binary search and sequentially access $\frac{2}{\epsilon} \cdot \log n$ bits, respectively.*

PROOF. We determine the running time by noticing that in each iteration k for each edge we perform addition with the sparse vector $\mathrm{PPR}^{(k)}$. We may use a linked list of non-zeroes for performing the addition, thus requiring $O(\log n)$ bit operations for each non-zero of the vector. Since in iteration k we store all values of a vector with norm at most one rounded down to a multiple of ϵ_k, we require $2/\epsilon_k$ space to store at most $1/\epsilon_k$ non-zeroes of $\mathrm{PPR}^{(k)}$ by Lemma 3. By $\sum_{k\le k_{\max}} \sqrt{1-c}^{k_{\max}-k} < \frac{1}{1-\sqrt{1-c}} < 2/c$ the total running time becomes $O(m \log n/(\epsilon \cdot c))$. □

2.2 Sketching

Next we give a sketch version of Algorithm 1 that improves the space requirement of the rounding based version by a factor of $\log n$, thus matches the lower bound of Section 4 for value queries. First we give a basic algorithm that uses uniform error bound ϵ in all iterations and is not optimized for storage size in bits. Then we show how to gradually decrease approximation error to speed up earlier iterations with less effect on final error; finally we obtain the space optimal algorithm by the bit encoding of Lemma 3.

The key idea is that we replace each $\mathrm{PPR}_u^{(k)}$ vector with its constant size Count-Min sketch in the dynamic programming iteration (4). Let S denote the sketching operator that replaces a vector by the $d \times w$ table as in Section 1.2 and let us perform the iterations of (4) with $\mathrm{SPPR}_u^{(k)}$ and $S(c \cdot \chi_u)$. Since the sketching operator is trivially linear, in iteration k we obtain the sketch of the next temporary vector $\mathrm{SPPR}_u^{(k)}$ from the sketches $\mathrm{SPPR}_v^{(k-1)}$.

To illustrate the main ingredients, we give the simplest form of a sketch-based algorithm with error, space and time analysis. Let us perform the iterations of (4) with e/ϵ wide and $\ln\frac{1}{\delta}$ deep sketches $k^*_{\max} = \log_{1-c}\epsilon$ times; then by Theorem 1 and the linearity of sketching we can estimate $\mathrm{PPR}_u(v)$ for all u, v from $\mathrm{SPPR}_u^{(k^*_{\max})}(v)$ with additive error 2ϵ and error probability δ. The personalized PageRank database consists of sketch tables $\mathrm{SPPR}_u^{(k^*_{\max})}$ for all u. The data occupies $\Theta(\frac{n}{\epsilon}\log\frac{1}{\delta})$ machine words, since we have to store n tables of reals. An update for node u takes $O(\frac{d^+(u)}{\epsilon}\ln\frac{1}{\delta})$ time by averaging $d^+(u)$ tables of $\frac{e}{\epsilon} \times \ln\frac{1}{\delta}$ size and adding $S(c\chi_u)$, each in $O(\ln\frac{1}{\delta})$ time. Altogether the required $k^*_{\max}$ iterations run in $O(\frac{m}{\epsilon}\ln\frac{1}{\delta}\log_{1-c}\epsilon)$ time.

[2] We measure time in bit operations in order to better compare with bit optimal sketches as in Theorem 5.

Next we weaken the dependence of the running time on the number of iterations by gradually decreasing error ϵ_k as in Section 2.1. When decreasing the error in sketches, we face the problem of increasing hash table sizes as the iterations proceed. Since there is no way to efficiently rehash data into larger tables, we approximate personalized PageRank slightly differently by representing the end distribution of length k walks, $\mathrm{PPR}_u^{[k]}$, with their rounded sketches $\widehat{\mathrm{SPPR}}_u^{[k]}$ in the path-summing formula (2):

$$\widehat{\mathrm{PPR}}_u(v) = \min_{i=1\ldots\ln\frac{1}{\delta}} \Big(\sum_{k=0}^{k_{\max}} c(1-c)^k \widehat{\mathrm{SPPR}}_u^{[k]}(i, h_k^i(v)) \Big) \quad (6)$$

where h_k^i denotes the i-th hash function of the k-th iteration. By (6) we need to calculate $\widehat{\mathrm{SPPR}}_u^{[k]}$ efficiently in small space. Notice that unlike in the dynamic programming where we gradually increase the precision of $\widehat{\mathrm{PPR}}_u^{(k)}$ as k grows, we may compute $\widehat{\mathrm{SPPR}}_u^{[k]}$ less precise with growing k since its values are scaled down by $(1-c)^k$. Hence we obtain our algorithm by using $e/\bar{\epsilon}_k = e/\epsilon_{k_{\max}-k}$ wide hash tables in the k-th iteration and replacing the last line of Algorithm 1 with

$$\widehat{\mathrm{SPPR}}_u^{[k]} \leftarrow \psi_{k_{\max}-k}\Big(\phi_{k_{\max}-k}\Big(\sum_{v:(uv)\in E} \widehat{\mathrm{SPPR}}_v^{[k-1]}/d^+(u)\Big)\Big) \quad (7)$$

where ϕ_k is the recoding function shrinking hash tables from width e/ϵ_{k+1} to e/ϵ_k. To be more precise, we round the width of each sketch up to the nearest power of two; thus we maintain the error bound, increase space usage by less than a factor of two, and use the recoding function that halves the table when necessary.

THEOREM 5. *Let us run the sketch version of the dynamic programming Algorithm 1 with sketching to width $e/\bar{\epsilon}_k$ and rounding all table entries to multiples of $\bar{\epsilon}_k$ in iteration k. The algorithm runs in time $O(m \ln \frac{1}{\delta}/(c\cdot\epsilon))$; builds a database of size $O(n \ln \frac{1}{\delta}/(c\cdot\epsilon))$ bits; and returns a value $\widehat{PPR}_u(v) \geq PPR_u(v) - \epsilon(1+2/c)$ such that Prob$(\widehat{PPR_u}(v) > PPR_u(v) + 2\epsilon/c) \leq \delta$.*

PROOF. As $\sum_{k=0}^{k_{\max}} 1/\bar{\epsilon}_k < 2/(\epsilon c)$ still holds, along the same lines as Theorem 4 we immediately get the running time; space follows by Lemma 3.

We err for three reasons: we do not run the iteration infinitely; in iteration k we round values down by at most $\bar{\epsilon}_k$, causing a deterministic negative error; and finally the Count-Min Sketch uses hashing, causing a random positive error. For bounding these errors, imagine running iteration (7) without the rounding function ψ_k but still with $e/\bar{\epsilon}_k$ wide and $\ln 1/\delta$ deep sketches and denote its results by $\mathrm{SPPR}_u^{[k]}$ and define

$$\widetilde{\mathrm{PPR}}_u(v) = \min_{i=1\ldots\ln\frac{1}{\delta}} \Big(\sum_{k=0}^{k_{\max}} c(1-c)^k \mathrm{SPPR}_u^{[k]}(i, h_k^i(v)) \Big).$$

First we upper bound $\widetilde{\mathrm{PPR}}$ to obtain the last claim of the Theorem. Since $\widehat{\mathrm{PPR}}_u(v) \leq \widetilde{\mathrm{PPR}}_u(v)$, we need Prob$(\widetilde{\mathrm{PPR}}_u(v) > \mathrm{PPR}_u(v) + 2\epsilon/c) \leq \delta$. By the Count-Min principle it is sufficient to show that for a fixed row i^*, the expected overestimation of $\sum_{k=0}^{k_{\max}} c(1-c)^k \mathrm{SPPR}_u^{[k]}(i^*, h_k^{i^*}(v))$ is not greater than $2\epsilon/c$. Since the bias of each sketch row $\mathrm{SPPR}_u^{[k]}(i^*, .)$ is $\bar{\epsilon}_k$, the bias of their $c(1-c)^k$ exponentially weighted sum is bounded by $2\epsilon/c$.

Finally we lower bound $\widehat{\mathrm{PPR}}$; the bound is deterministic. The $\bar{\epsilon}_k$ loss due to rounding down in iteration k affects all subsequent iterations, and hence

$$\widehat{\mathrm{PPR}}_u(v) \geq \widetilde{\mathrm{PPR}}_u(v) - \sum_{k=0}^{\infty} \bar{\epsilon}_k \cdot (1-c)^k \geq \widetilde{\mathrm{PPR}}_u(v) - 2\epsilon/c.$$

And since we sum up to $k_{\max}$ instead of infinity, $\widetilde{\mathrm{PPR}}_u(v)$ underestimates $\mathrm{PPR}_u(v)$ by at most $\epsilon^2 \leq \epsilon$, proving the Theorem. □

3. SimRank

In this section first we give a simpler algorithm for serving SimRank value and top-list queries that combines rounding with the empirical fact that there are relatively few large values in the similarity matrix. Then in Section 3.1 we give an algorithm for SimRank values that uses optimal storage in the sense of the lower bounds of Section 4. Of independent interest is the main component of the algorithm that reduces SimRank to the computation of values similar to personalized PageRank.

SimRank and personalized PageRank are similar in that they both fill an $n \times n$ matrix when the exact values are computed. Another similarity is that practical queries may ask for the maximal elements within a row. Unlike personalized PageRank however, when rows can be easily sketched and iteratively computed over approximate values, the $n \times n$ matrix structure is lost within the iterations for $\mathrm{Sim}(v_1, v_2)$ as we may have to access values of arbitrary $\mathrm{Sim}(u_1, u_2)$. Even worse $\sum_{u,v} \mathrm{PPR}_u(v) = n$ while

$$M = \sum_{u_1,u_2} \mathrm{Sim}(u_1, u_2)$$

can in theory be as large as $\Omega(n^2)$; an $O(n)$-size sketch may hence store relevant information about personalized PageRank but could not even contain values below 1 for SimRank. An example of a sparse graph with $M = \Omega(n^2)$ is an n-node star where $\mathrm{Sim}(u, v) = 1 - c$ for all pairs other than the center.

In practice M is expected be a reasonable constant times n. Hence first we present a simple direct algorithm that finds the largest values within the entire $\mathrm{Sim}(u_1, u_2)$ table. In order to give a rounded implementation of the iterative SimRank equation (5), we need to give an efficient algorithm to compute a single iteration. The naive implementation requires $\Omega(1)$ time for each edge pair with a common source vertex that may add up to $\Omega(n^2)$. Instead for $u_1 \neq u_2$ we will compute the next iteration with the help of an intermediate step when edges out of only one of the two vertices are considered:

$$\mathrm{ASim}^{(k)}(u_1, v_2) = \frac{\sqrt{1-c}}{d^-(u_1)} \sum_{v_1:(v_1u_1)\in E} \mathrm{Sim}^{(k-1)}(v_1, v_2) \quad (8)$$

$$\mathrm{Sim}^{(k)}(u_1, u_2) = \frac{\sqrt{1-c}}{d^-(u_2)} \sum_{v_2:(v_2u_2)\in E} \mathrm{ASim}^{(k)}(u_1, v_2) \quad (9)$$

Along the same line as the proof of Theorems 2 we prove that (i) by rounding values in iterations (8–9) we approximate values with small additive error; (ii) the output of the algorithm occupies small space; and (iii) approximate top lists can be efficiently answered from the output. The proof is omitted due to space limitations. We remark here that (8–9) can be implemented by 4 external memory sorts per iteration, in two of which the internal space usage can in theory grow arbitrary large even compared to M. This is due to the fact that we may round only once after each iteration; hence if for some large out-degree node v a value $\mathrm{Sim}^{(k-1)}(v, v_2)$ is above the rounding threshold or $\mathrm{ASim}^{(k)}(u_1, v)$ becomes positive, then we have to temporarily store positive values for all out-neighbors, most of which will be discarded when rounding.

THEOREM 6. *Let us iterate (8–9) $k_{\max} = 4 \log_{1-c} \epsilon$ times by rounding values in iteration k down to multiples of $\epsilon_k = \epsilon \cdot (1-c)^{-\frac{k_{\max}-k}{4}}$ for (9) and $\epsilon_{k-1} + \epsilon_k$ for (8).*

(i) *The algorithm returns approximate values for $u \neq v$ with $\mathit{Sim}(u,v) \geq \widehat{\mathit{Sim}}(u,v) \geq \mathit{Sim}(u,v) - 4\epsilon/c$.*

(ii) *The space used by the $\widehat{\mathit{Sim}}(u,v)$ values is $O(M \cdot \frac{2}{\epsilon} \cdot \log n)$ bits where $M = \sum_{u,v} \mathit{Sim}(u,v)$.*

(iii) *Top list queries can be answered after positive $\widehat{\mathit{Sim}}(u,\cdot)$ values are sorted for each u in $O(M \cdot \frac{2}{\epsilon} \cdot \log^2 n)$ time.*

3.1 Reduction of SimRank to PPR

Now we describe a SimRank algorithm that uses a database of size matching the corresponding lower bound of Section 4 by taking advantage of the fact that large values of similarity appear in blocks of the $n \times n$ similarity table. The blocking nature can be captured by observing the similarity of Sim_{v_1,v_2} to the product $\mathrm{PPR}_{v_1} \cdot \mathrm{PPR}_{v_2}$ of vectors PPR_{v_1} and PPR_{v_2}.

We use the independent result of [10, 17, 16] that PageRank type values can be expressed by summing over endpoints of walks as in equation (1). First we express SimRank by walk pair sums, then we show how SimRank can be reduced to personalized PageRank by considering pairs of walks as products. Finally we give sketching and rounding algorithms for value and top queries based on this reduction.

In order to capture pairs of walks of equal length we define “reversed” PPR by using walks of length exactly k by modifying (1):

$$\mathrm{RP}_v^{[k]}(u) = \sum_{v_0=v,v_1,\ldots,v_k=u} 1/(d^-(v_1)\cdots d^-(v_k)) \qquad (10)$$

where $v_0, \ldots, v_k$ is a walk from v to u on the *transposed graph*. Similarly [16] shows that $\mathrm{Sim}^{(k)}_{v_1,v_2}$ equals the total weight of pairs

$$\begin{aligned} v_1 = w_0, w_1, \ldots, w_{k'-1}, w_{k'} &= u \\ v_2 = w'_0, w'_1, \ldots, w'_{k'-1}, w'_{k'} &= u \end{aligned}$$

with length $k' \leq k$ that both end at u and one of them comes from v_1 while the other one from v_2. The weight of the pair of walks is the *expected* $(1-c)$ *meeting distance* as defined in [16]:

$$(1-c)^{k'} / (d^-(w_1)\cdots d^-(w_{k'}) \cdot d^-(w'_1)\cdots d^-(w'_{k'})) \qquad (11)$$

Importantly the walks satisfy two properties: they have (i) equal length and (ii) no common vertex at the same distance from start, i.e. $w_i \neq w'_i$ for $i \neq k'$. Except for the last two requirements $\mathrm{Sim}^{(k)}_{v_1,v_2}$ has a form similar to the inner product of PPR_{v_1} and PPR_{v_2} on the reversed graph by (1).

Next we formalize the relation and give an efficient algorithm that reduces SimRank to PPR on the reversed graph. As a “step 0 try” we consider

$$\mathrm{Sim}^{(0)}_{v_1,v_2} = \sum_{k>0} (1-c)^k \sum_u \mathrm{RP}_{v_1}^{[k]}(u) \mathrm{RP}_{v_2}^{[k]}(u) \qquad (12)$$

with form similar to SimRank with exact length k walks except that these walks may have common vertices unlike stated in (ii).

In order to exclude pairs of walks that meet before ending, we use the principle of inclusion and exclusion. We count pairs of walks that have at least t meeting points after start as follows. Since after their first meeting point v the walks proceed as if computing the similarity of v to itself, we introduce a self-similarity measure by counting weighted pairs of walks that start at v and terminate at the same vertex u by extending (12):

$$\mathrm{SSim}^{(0)}(v) = \sum_u \sum_{k>0} (1-c)^k \mathrm{RP}_v^{[k]}(u) \mathrm{RP}_v^{[k]}(u) \qquad (13)$$

Algorithm 2 SimRank by reduction to personalized PageRank. The algorithm approximates $\widehat{\mathrm{SSim}}$ values; approximate $\widehat{\mathrm{Sim}}_{v_1,v_2}$ values are served by evaluating (17) at query time.

```
for $k := 0, \ldots, k_{\max} = 2 \log_{1-c} \epsilon$ do
    $\epsilon_k = \epsilon \cdot (1-c)^{-\frac{k_{\max}-k}{2}}$
    Define function $\psi_k$ that rounds down to multiples of $\epsilon_k$
for each node $u$ do
    Initialize $\widehat{\mathrm{RP}}_u^{[0]}$ by $\psi_{k_{\max}}(\chi_u)$ and $\widehat{\mathrm{SSim}}^{(-1)}(u) = \widehat{\mathrm{SSim}}(u)$ by 1
for $k := 1, \ldots, k_{\max} = 2 \log_{1-c} \epsilon$ do
    for each node $u$ do
        $\widehat{\mathrm{RP}}_u^{[k]} \leftarrow \psi_{k_{\max}-k}\Big(\sum_{v:(vu)\in E} \widehat{\mathrm{RP}}_v^{[k-1]}/d^-(u)\Big)$
for $t := 0, \ldots, t_{\max} = \log_{\frac{1-c}{c}} \epsilon$ do
    for each node $v$ do
        $\widehat{\mathrm{SSim}}^{(t)}(v) = \sum_u \sum_{k>0} (1-c)^k \widehat{\mathrm{RP}}_v^{[k]}(u) \widehat{\mathrm{RP}}_v^{[k]}(u) \widehat{\mathrm{SSim}}^{(t-1)}(u)$
        $\widehat{\mathrm{SSim}}(v) = \widehat{\mathrm{SSim}}(v) - (-1)^t \widehat{\mathrm{SSim}}^{(t)}(v)$
for each node $v$ do
    Round $\widehat{\mathrm{SSim}}(v)$ to multiples of $\epsilon$.
```

Now we may recursively define a value that counts all pairs of walks with at least $t+1$ inner points where the walks meet; the values below in fact count each pair $\binom{s}{t+1}$ times that meet at exactly s inner points. First we define self-similarity as

$$\mathrm{SSim}^{(t+1)}(v) = \sum_u \sum_{k>0} (1-c)^k \mathrm{RP}_v^{[k]}(u) \mathrm{RP}_v^{[k]}(u) \cdot \mathrm{SSim}^{(t)}(u) \qquad (14)$$

and then similarity with at least $t+1$ inner meeting points as

$$\mathrm{Sim}^{(t+1)}_{v_1,v_2} = \sum_u \sum_{k>0} (1-c)^k \mathrm{RP}_{v_1}^{[k]}(u) \mathrm{RP}_{v_2}^{[k]}(u) \cdot \mathrm{SSim}^{(t)}(u). \qquad (15)$$

By the principle of inclusion and exclusion

$$\mathrm{Sim}(v_1, v_2) = \mathrm{Sim}^{(0)}_{v_1,v_2} - \mathrm{Sim}^{(1)}_{v_1,v_2} + \mathrm{Sim}^{(2)}_{v_1,v_2} - \ldots \qquad (16)$$

where $\mathrm{Sim}^{(0)}$ is defined in (12) as the weighted number of walk pairs that are unrestricted in the number of meeting times. Induction on t shows that $\mathrm{Sim}^{(t)}_{v_1,v_2} \leq \left(\frac{1-c}{c}\right)^{t+1}$, thus the infinite series (16) is (absolute) convergent if $\frac{1-c}{c} < 1$. By changing the order of summation (16) becomes

$$\begin{aligned} \mathrm{Sim}(v_1, v_2) &= \sum_{k>0} (1-c)^k \sum_u \mathrm{RP}_{v_1}^{[k]}(u) \mathrm{RP}_{v_2}^{[k]}(u) \cdot \mathrm{SSim}(u), \text{ where} \\ \mathrm{SSim}(u) &= 1 - \sum_{t\geq 0} (-1)^t \mathrm{SSim}^{(t)}(u). \qquad (17) \end{aligned}$$

The proof of the main theorems below are omitted due to space limitations.

THEOREM 7. *If $c > 1/2$ Algorithm 2 uses $n \cdot \frac{4}{c\epsilon} \cdot \log n$ bits to store $\widehat{RP}^{[k]}$ and $\widehat{SSim}^{(t)}$ values; the $\log n$ factor can be replaced by $e \log 1/\delta$ by sketching $\widehat{RP}^{[k]}$ in width $e/\epsilon_{k_{\max}-k}$. The algorithm computes these values in $O((m+n \log_{\frac{1-c}{c}} \epsilon)/(c\epsilon))$ time measured in RAM operations and approximates $\widehat{Sim}(v_1, v_2)$ in $O(\log n/(c\epsilon))$ bit operations time. For given v_1 the set S of non-negative $\widehat{Sim}(v_1, v_2)$ values can be computed in bit operations time $O(|S| \log n/(c\epsilon))$.*

THEOREM 8. *If $c > 1/2$ the above algorithm gives $|\widehat{Sim}_{v_1,v_2} - Sim_{v_1,v_2}| = O(\epsilon/(c-1/2)^2)$; when sketching RP, this holds with probability at least $1-\delta$.*

4. Lower bounds

In this section we will prove lower bounds on the database size of approximate PPR algorithms that achieve personalization over a subset of H vertices. 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 a database; in the second phase the algorithm gets a query and has to answer by accessing the database, i.e. the algorithm cannot access the graph during query-time. A $b(H)$ *worst case lower bound* on the database size holds, if for any two-phase algorithm there exists a personalization input such that a database of size $b(H)$ bits is built in the first phase.

We will consider the following queries for $0 \le \epsilon, \delta, \phi \le 1$:

- ϵ–δ *value approximation*: given the vertices u, v approximate $\mathrm{PPR}_u(v)$ with $\widehat{\mathrm{PPR}}_u(v)$ such that

$$\mathrm{Prob}\{|\mathrm{PPR}_u(v) - \widehat{\mathrm{PPR}}_u(v)| \le \epsilon\} \ge 1 - \delta.$$

- $\phi - \epsilon - \delta$ *top query*: given the vertex u, with probability $1 - \delta$ compute the set of vertices W which have personalized PPR values according to vertex u greater than ϕ. Precisely we require the following:

$$\begin{aligned} \forall w \in V \quad &: \quad \mathrm{PPR}_u(w) \ge \phi \Rightarrow w \in W \\ \forall w \in W \quad &: \quad \mathrm{PPR}_u(w) \ge \phi - \epsilon \end{aligned}$$

As Theorem 6 of [11] shows, any two-phase PPR algorithm solving the exact ($\epsilon = 0$) PPR value problem requires an $\Omega((1 - 2\delta)|H| \cdot n)$ bit database. Our tool towards the lower bounds will be the asymmetric communication complexity game *bit-vector probing* [14]: there are two players A and B; player A has a vector x of m bits; player B has a number $y \in \{1, 2, \ldots, m\}$; and they have to compute the function $f(x, y) = x_y$, i.e., the output is the y^{th} bit of the input vector x. To compute the proper output they have to communicate, and communication is restricted in the direction $A \to B$. The *one-way communication complexity* [21] of this function is the number of transferred bits in the worst case by the best protocol.

THEOREM 9 ([14]). *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.*

Now we are ready to state and prove our lower bounds, which match the performance of the algorithms presented in Sections 2 and 3.1, hence showing that they are space optimal.

THEOREM 10. *Any two-phase* ϵ–δ *PPR value approximation algorithm with* $\epsilon, \delta > 0$ *builds a database of* $\Omega(\frac{1}{\epsilon} \cdot \log \frac{1}{\delta} \cdot |H|)$ *bits in worst case, when the graph has at least* $|H| + \frac{1-c}{8\epsilon\delta}$ *nodes.*

PROOF. We prove the theorem by reducing the bit vector probing problem to the ϵ–δ approximation. Given a vector x of $m = \Omega(\frac{1}{\epsilon} \cdot \log \frac{1}{\delta} \cdot |H|)$ bits, player A will construct a graph and compute a PPR database with the first phase of the ϵ–δ approximation algorithm. Then A transmits this database to B. Player B will perform a sequence of second-phase queries such that the required bit x_y will be computed with error probability $\frac{1}{4}$. The above outlined protocol solves the bit vector probing with error probability $\frac{1}{4}$. Thus the database size that is equal to the number of transmitted bits is $\Omega(m) = \Omega(\frac{1}{\epsilon} \cdot \log \frac{1}{\delta} \cdot |H|)$ in worst case by Theorem 9. It remains to show the details of the graph construction on A's side and the query algorithm on B's side.

Given a vector x of $m = \frac{1-c}{2\epsilon} \cdot \log \frac{1}{4\delta} \cdot |H|$ bits, A constructs the "bipartite" graph with vertex set $\{u_i : i = 1, \ldots, |H|\} \cup \{v_{j,k} : j = 1, \ldots, \frac{1-c}{2\epsilon}, k = 1, \ldots, \frac{1}{4\delta}\}$. For the edge set, x is partitioned into $\frac{1-c}{2\epsilon} \cdot |H|$ blocks, where each block $b_{i,j}$ contains $\log \frac{1}{4\delta}$ bits for $i = 1, \ldots, |H|$, $j = 1, \ldots, \frac{1-c}{2\epsilon}$. Notice that each $b_{i,j}$ can be regarded as a binary encoded number with $0 \le b_{i,j} < \frac{1}{4\delta}$. To encode x into the graph, A adds an edge $(u_i, v_{j,k})$ iff $b_{i,j} = k$, and also attaches a self-loop to each $v_{j,k}$. Thus the $\frac{1-c}{2\epsilon}$ edges outgoing from u_i represent the blocks $b_{i,1}, \ldots, b_{i,(1-c)/2\epsilon}$.

After constructing the graph A computes an ϵ–δ approximation PPR database with personalization available on $u_1, \ldots, u_{|H|}$, and sends the database to B, who computes the y^{th} bit x_y as follows. Since B knows which of the blocks contains x_y it is enough to compute $b_{i,j}$ for suitably chosen i, j. The key property of the graph construction is that $\mathrm{PPR}_{u_i}(v_{j,k}) = \frac{1-c}{d^+(u_i)} = 2\epsilon$ iff $b_{i,j} = k$ otherwise $\mathrm{PPR}_{u_i}(v_{j,k}) = 0$. Thus B computes $\widehat{\mathrm{PPR}}_{u_i}(v_{j,k})$ for $k = 1, \ldots, \frac{1}{4\delta}$ by the second phase of the ϵ–δ approximation algorithm. If all $\widehat{\mathrm{PPR}}_{u_i}(v_{j,k})$ are computed with $|\mathrm{PPR}_{u_i}(v_{j,k}) - \widehat{\mathrm{PPR}}_{u_i}(v_{j,k})| \le \epsilon$, then $b_{i,j}$ containing x_y will be calculated correctly. By the union bound the probability of miscalculating any of $\widehat{\mathrm{PPR}}_{u_i}(v_{j,k})$ is at most $\frac{1}{4\delta} \cdot \delta = \frac{1}{4}$. □

THEOREM 11. *Any two-phase PPR algorithm solving the top query problem with parameters* $\epsilon > 0$, $\delta \ge 0$ *builds a database of* $\Omega(\frac{1-2\delta}{\epsilon}|H| \log n)$ *bits in worst case, when the graph has* $n \ge 2|H| + \left(\frac{1-c}{2\epsilon}\right)^2$ *nodes.*

PROOF. We will proceed according to the proof of Theorem 10.

Let $\phi = 2\epsilon$ and $k = \lfloor \frac{1-c}{2\epsilon} \rfloor$ and the graph have nodes $\{u_i : i = 1, \ldots, |H|\} \cup \{v_j : j = 1, \ldots, n^*\}$ (with $n^* = n - |H|$). By the assumptions on the vertex count, $n^* = \Omega(n)$ and $\sqrt{n^*} \ge k$.

Let the size of the bit-vector probing problem's input be $m = |H| \cdot k \cdot \log n^*/2$. Assign each of the $k \cdot \log n^*/2$ sized blocks to a vertex u_i and fix a code which encodes these bits into k-sized subsets of the vertices $\{v_j\}$. This is possible, as the number of subsets is $\binom{n^*}{k} > (\frac{n^*}{k})^k \ge \sqrt{n^*}^k$. These mappings are known to both parties A and B. Note that due to the constraints on $n^*, k, |H|$ and n we have $k \cdot \log n^*/2 = \Omega(\frac{1}{\epsilon} \log n)$.

Given an input bit-vector of A, for each vertex u_i take its block of bits and compute the corresponding subset of vertices $\{v_j\}$ according to the fixed code. Let u_i have an arc into these vertices. Let all vertices v_j have a self-loop. Now A runs the first phase of the PPR algorithm and transfers the resulting database to B.

Given a bit index y, player B determines its block, and issues a top query on the representative vertex, u_i. As each of the out-neighbors w of u_i has $\mathrm{PPR}_{u_i}(w) = \frac{1-c}{d^+(u_i)} = \frac{1-c}{k} \ge \phi$, and all other nodes w' have $\mathrm{PPR}_{u_i}(w') = 0$, the resulting set will be the set of out-neighbors of u_i, with probability $1 - \delta$. Applying the inverse of the subset encoding, we get the bits of the original input vector, thus the correct answer to the bit-vector probing problem. Setting $\frac{1+\gamma}{2} = 1 - \delta$ we get that the number of bits transmitted, thus the size of the database was at least $\Omega(\gamma|H| \cdot k \cdot \log n^*/2) = \Omega(\frac{1-2\delta}{\epsilon}|H| \log n)$. □

We remark that using the graph construction in the full version of [12] it is straightforward to modify Theorems 10 and 11 to obtain the same space lower bounds for SimRank as well. Moreover, it is easy to see that Theorem 11 holds for the analogous problem of approximately reporting the top $t = 1/\phi$ number vertices with the highest PageRank or SimRank scores respectively.

With the graph construction of Theorem 10 at hand, it is possible to bypass the bit-vector probing problem and reduce the approximate personalized PageRank value query to similar lower bounds for the Bloom filter [4] or to a weaker form for the Count-Min sketch [8]. However, to the best of our knowledge, we are unaware of previous results similar to Theorem 11 for the top query.

5. Experiments

This section presents our personalized PageRank experiments on 80M pages of the 2001 Stanford WebBase crawl [15]. The following questions are addressed by our experiments:

- How does the rounding error affect the quality and running time of the rounded dynamic programming algorithm? We confirm that the maximum error drops linearly in the rounding error ϵ, and find that running times increase sublinearly with $1/\epsilon$, making rounded dynamic programming particularly scalable.
- How do the qualities of the various algorithms relate to each other? We conclude that rounded dynamic programming outperforms all the other methods by a large margin.

5.1 Measuring the Quality of Approximate PageRank Scores

We compare our approximate PPR scores to exact PPR scores computed by the personalized PageRank algorithm of Jeh and Widom [17] with a precision of 10^{-8} in L_1 norm. In the experiments we set teleportation constant c to its usual value 0.15, and personalize on a single page u chosen uniformly at random from all vertices. The experiments were carried out with 1000 independently chosen personalization node u, and the results were averaged.

To compare the exact and approximate PPR scores personalized to page u, we measure the difference between top score lists of exact PPR_u and approximate $\widehat{\mathrm{PPR}}_u$ vectors. The length t of the compared top lists is in the range 5 to 1000, which is the usual maximum length of the results returned by search engines.

The comparison of top lists is key in measuring the goodness of a ranking method [9, and the references therein] or the distortion of a PageRank encoding [13]. Let T_t^u denote the set of pages having the t highest personalized PageRank values in the vector PPR_u personalized to a single page u. We approximate this set by $\widehat{T_t^u}$, the set of pages having the t highest approximated scores in vector $\widehat{\mathrm{PPR}}_u$. We will apply the following three measures to compare the exact and approximate rankings of T_t^u and $\widehat{T_t^u}$. The first two measures will determine the overall quality of the approximated top-t set $\widehat{T_t^u}$, as they will be insensitive to the ranking of the elements within $\widehat{T_t^u}$.

Relative aggregated goodness [26] measures how well the approximate top-t set performs in finding a set of pages with high total personalized PageRank. It calculates the sum of exact PPR values in the approximate set compared to the maximum value achievable (by using the exact top-t set T_t^u):

$$\mathrm{RAG}(t,u) = \sum_{v\in\widehat{T_t^u}} \mathrm{PPR}_u(v) / \sum_{v\in T_t^u} \mathrm{PPR}_u(v)$$

We also measure the *precision* of returning the top-t set (note that as the sizes of the sets are fixed, precision coincides with *recall*). If all exact PPR scores were different we could simply define precision as $\frac{|\widehat{T_t^u}\cap T_t^u|}{t}$. Treating nodes with equal exact PPR scores in a more liberal way we define precision as

$$\mathrm{Prec}(t,u) = |v \in \widehat{T_t^u} : \mathrm{PPR}_u(v) \geq \min_{w\in T_t^u} \mathrm{PPR}_u(w)|/t$$

The third measure, *Kendall's* τ compares the exact ranking with the approximate ranking in the top-t set. Note that the exact PPR list of nodes with a small neighborhood or the tail of approximate PPR ranking may contain a large number of ties (nodes with equal 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 [19]. Ignoring ties for the ease of presentation, the rank correlation Kendall's τ compares the number of pairs ordered the same way in both rankings to the number of reversed pairs; its range is $[-1,+1]$, where -1 expresses complete disagreement, $+1$ represents a perfect agreement. To restrict the computation to the top t elements, we took the union of the exact and approximated top-t sets $T_t^u \cup \widehat{T_t^u}$. For each ordering, all nodes that were outside the orderings' top-t set were considered to be tied and ranked strictly smaller than any node contained in its top-t set.

5.2 Breadth First Search Heuristic

One of our baselines in our experiments is a heuristically modified power iteration algorithm. While the example of Figure 1 shows that we may get large error even by discarding very small intermediate values, a heuristic that delays the expansion of nodes with small current PageRank values [17, 22, 6] still achieves good results on real world data.

When personalizing to node u, let us start from χ_u and keep a dedicated list of the non-zero entries, which we expand breadth first. This allows us to perform one iteration quickly as long as these lists are not too long; we cease the expansion if we have reached the value S. Moreover we skip the expansion originating from a node if its current PageRank divided by the outdegree is below a given threshold E. Finally we never let the number of iteration exceed the predetermined value K. We experimented with a variant of McSherry's state of the art update iteration [22], as well as a scheme to reuse the previous node's result, but neither of them produced better approximation within the same running time, hence we do not report these results.

5.3 Results

We conducted our experiments on a single AMD Opteron 2.0 GHz machine with 4 GB of RAM under Linux OS. We used a semi-external memory implementation for rounded dynamic programming, partitioning the intermediate $\widehat{\mathrm{PPR}}_u^{(k)}(w)$ vectors along the coordinate w. Using a single vector allowed us to halve the memory requirements by storing the intermediate results in a FIFO like large array, moving the $\widehat{\mathrm{PPR}}_u^{(k)}$ being updated from the head of the queue to its tail. We stored the PageRank values as multiples of the rounding error ϵ using a simple but fast variable length byte-level encoding. We did not partition the vector $\widehat{\mathrm{PPR}}_u^{(k)}(w)$ into predefined subsets of w; instead as the algorithm ran out of memory, it split the current set of w and checkpointed one half to disk. Once the calculation in the first half was finished, it resumed in the second half, resulting in a Depth First Search like traversal of subsets of w. Since dynamic programming accesses the edges of the graph sequentially, we could overlay the preloading of the next batch of edges with the calculations in the current batch using either asynchronous I/O or a preloader thread. This way we got the graph I/O almost for free. It is conceivable that reiterations [22] or the compression of vertex identifiers [3] could further speed up the computation. For implementations on a larger scale one may use external memory sorting with the two vector dynamic programming variant. Or, in a distributed environment, we may partition the graph vertices among the cluster nodes, run a semi-external memory implementation on each node and exchange the intermediate results over the network as required. To minimize network load, a partition that respects the locality of links (e.g. hostname based) is advisable.

In our first experiment we demonstrate the convergence of rounded dynamic programming measured by the maximum error as the number of iterations increases whilst keeping ϵ fixed at a modest 10^{-4} in all iterations. On Figure 2, left, it can be clearly seen that the underlying exact dynamic programming converges in far fewer iterations than its worst case bound of $\log_{1-c} \epsilon = 57$ and that the maximum error due to rounding is roughly one half of the bound $\epsilon/c \approx 7 \cdot 10^{-4}$ provable for the simplified algorithm that rounds to fixed ϵ in all iterations. On Figure 2, right we see the goodness of the approximate ranking improving the same way as maximum error drops. This is theoretically clear since at maximum one-sided error of ϵ' we never swap pairs in the approximate rankings with exact PageRank values at least ϵ' apart. In the chart we set the top set size t to 200.

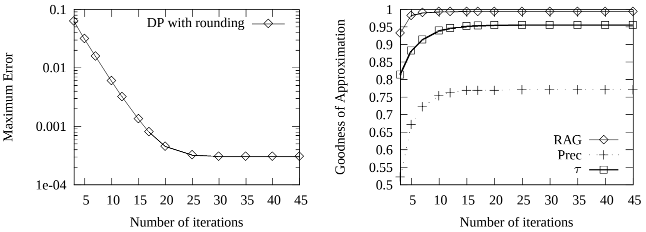

Figure 2: Effect of the number of iterations $k_{\max}$ with $\epsilon = 10^{-4}$ rounding error.

Based on the experiences of Figure 2 we set the number of iterations $k_{\max}$ to a conservative 35 and investigated the effects of the error ϵ, using rounding to varying ϵ_k as in Algorithm 1. The straight line on Figure 3, left clearly indicates that rounding errors do not accumulate and the maximum error decreases linearly with the rounding error as in Theorem 2. Additionally the error is much smaller than the worst case bound[3], for example at $\epsilon = 10^{-5}$ it is $3.5 \cdot 10^{-5}$ vs. $20 \cdot 10^{-5}$. The running time scales sublinearly in $1/\epsilon$ (e.g. from 3.5 hours to 14 hours when moving from $\epsilon = 10^{-3}$ to $\epsilon = 10^{-4}$) as it is governed by the *actual* average number of PPR$_u$ entries being above ϵ and not the worst case upper estimate $1/\epsilon$. The same observation applies to the database size, which requires 201 GB for $\epsilon = 10^{-5}$. The right panel of Figure 3 demonstrates that the approximate ranking improves accordingly as we move to smaller and smaller ϵ (we set the top set size t to 100).

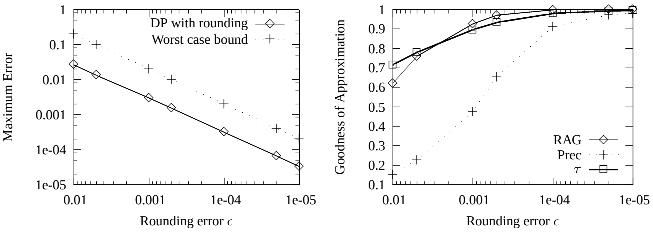

Figure 3: Effect of the rounding error ϵ with $k_{\max} = 35$ iterations.

Algorithm	Reference	Running time
Dynamic Programming with $\epsilon = 2 \cdot 10^{-5}$ and $\epsilon = 10^{-5}$ rounding to varying ϵ_k	Section 2.1	1.5 and 2.25 days
Dynamic Programming with $\epsilon = 6 \cdot 10^{-3}, \delta = 4 \cdot 10^{-3}$ sketches	Section 2.2	6 days
Monte Carlo sampling with $N = 10000$ samples and path truncation length $L = 35$	[12]	6 days
Breadth First Search heuristic with parameters $E = 10^{-5}, S = 2000, K = 35$	Section 5.2	3.5 days

Table 2: Summary of approximate personalized PageRank algorithms and heuristics.

We now turn to the experimental comparison of personalized PageRank approximation algorithms summarized in Table 2. For the BFS heuristic and the Monte Carlo method we used an elementary compression (much simpler and faster than [3]) to store the Stanford WebBase graph in 1.6 GB of main memory, hence the semi-external dynamic programming started with a handicap. Finally we mention that we did not apply rounding to the sketches.

It is possible to enhance the precision of most of the above algorithms by applying the neighbor averaging formula (4) over the approximate values once at query time. We applied this trick to all algorithms, except for the sketches (where it does not make a difference), which resulted in slightly higher absolute numbers for larger top lists at a price of a tiny drop for very short top lists, but did not affect the shape or the ordering of the curves. Due to the lack of space, we report neighbor averaged results only.

All three measures of Figure 4 indicate that as the top list size increases, the task of approximating the top-t set becomes more and more difficult. This is mainly due to the fact that among lower ranked pages the personalized PageRank difference is smaller and hence harder to capture using approximation methods. As expected, RAG scores are the highest, and the strictest Kendall's τ scores are the smallest with precision values being somewhat higher than Kendall's τ.

Secondly, the ordering of the algorithms with respect to the quality of their approximate top lists is fairly consistent among Relative Aggregated Goodness, Precision and Kendall's τ. Rounded dynamic programming performs definitely the best, moreover its running times are the lowest among the candidates. The runner up Breadth-First Search heuristic and Monte Carlo sampling seem to perform similarly with respect to Relative Aggregated Goodness with Monte Carlo sampling underperforming in the precision and Kendall's τ measures. Although sketching is quite accurate for single value queries with an average additive error of $5 \cdot 10^{-4}$, its top list curve drops sharply at size t around 50. This is due to the fact that $\delta \gg 1/n$ and hence a large number of significantly overestimated vertices crowd out the true top list. Based on these findings, for practical applications we recommend rounded dynamic programming with neighbor averaging , which achieves solid precision and rank correlation (at or above 0.95) over the top $200-300$ pages with reasonable resource consumption.

Lastly we remark that preliminary SimRank experiments over the same WebBase graph indicate that Algorithm 2 outperforms iterations (8–9) when using an equal amount of computation. Additionally SimRank scores computed by Algorithm 2 achieve marginally weaker agreement (Kruskal-Goodman $\Gamma = 0.33$) with the Open Directory Project (http://dmoz.org) as Monte Carlo sampling ($\Gamma = 0.34$) [12] but with higher recall.

[3] For the ease of presentation, we have ignored the issue of dangling nodes, ie. nodes with zero out-degree, throughout this paper. In the presence of dangling nodes the bound of Theorem 2 becomes $3\epsilon/c$.

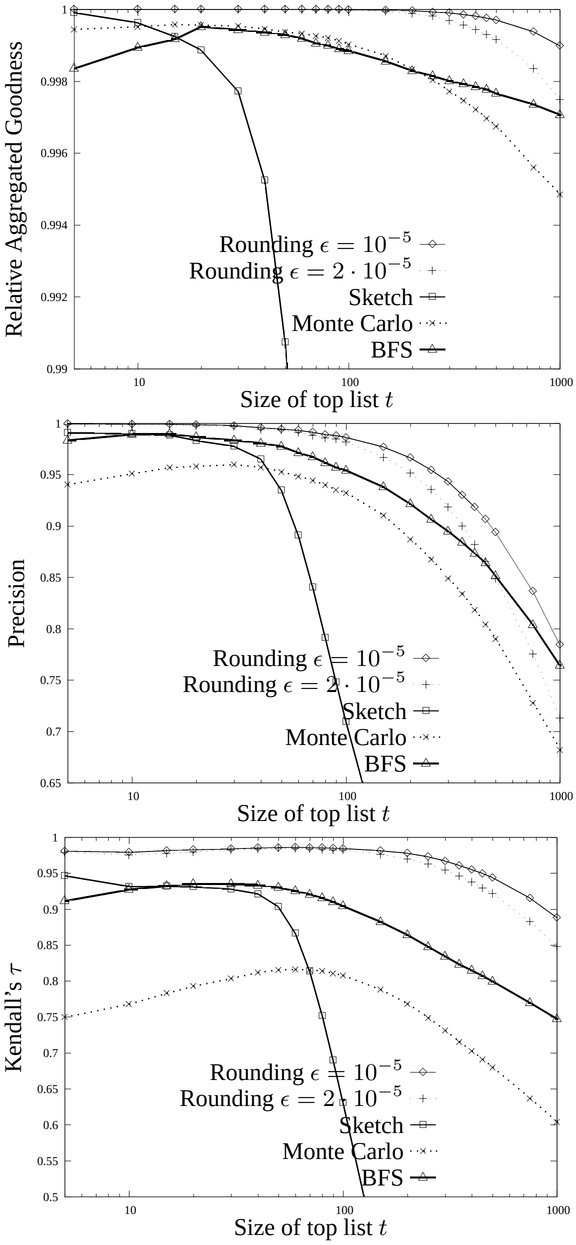

Figure 4: Empirical comparison of approximate personalized PageRank algorithms with three measures of goodness and varying top set size t.

6. Conclusions and Future work

We presented algorithms for the personalized PageRank and SimRank problems that give provable guarantees of approximation and build space optimal data structures to answer arbitrary on-line user queries. Experiments over 80M pages showed that for the personalized PageRank problem rounded dynamic programming performs remarkably well in practice both in terms of precomputation time, database size and the quality of approximation.

An interesting open question is whether our techniques can be extended to approximate other properties of Markov chains over massive graphs, e.g. the hitting time. Another important area of future work is to thoroughly evaluate the quality of the SimRank approximation algorithms. Finally we leave the existence of theoretically optimal algorithms for SimRank value and top list queries with parameter $c \leq 1/2$ open.

7. REFERENCES

[1] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Sampling algorithms: Lower bounds and applications. *Proc. of 33rd STOC*, 2001.

[2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[3] P. Boldi and S. Vigna. The webgraph framework I: Compression techniques. *Proc. of 13th WWW*, pp. 595–602, 2004.

[4] A. Z. Broder and M. Mitzenmacher. Network applications of Bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2005.

[5] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Proc. of 29th ICALP*, pp. 693–703, 2002.

[6] Y.-Y. Chen, Q. Gan, and T. Suel. Local methods for estimating PageRank values. *Proc. of 12th CIKM*, pp. 381–389, 2004.

[7] G. Cormode and S. Muthukrishnan. An improved data stream summary: The Count-Min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[8] G. Cormode and S. Muthukrishnan. Summarizing and mining skewed data streams. *Proc. of 5th SIAM Intl. Conf. on Data Mining*, 2005.

[9] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee. Comparing and aggregating rankings with ties. *Proc. of 23rd PODS*, 2004.

[10] D. Fogaras. Where to start browsing the web? *Proc. of 3rd I2CS*, Springer LNCS vol. 2877, pp. 65–79, 2003.

[11] D. Fogaras and B. Rácz. Towards scaling fully personalized PageRank. *Proc. of 3rd WAW*, pp. 105–117, 2004. Full version to appear in Internet Mathematics.

[12] D. Fogaras and B. Rácz. Scaling link-based similarity search. *Proc. of 14th WWW*, pp. 641–650, 2005. Full version available at `www.ilab.sztaki.hu/websearch/Publications/`.

[13] T. H. Haveliwala. Topic-sensitive PageRank: A context-sensitive ranking algorithm for web search. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):784–796, 2003.

[14] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. In *External Memory Algorithms, DIMACS Book Series vol. 50.*, pp. 107–118. American Mathematical Society, 1999.

[15] J. Hirai, S. Raghavan, H. Garcia-Molina, and A. Paepcke. WebBase: A repository of web pages. *Proc. of 9th WWW*, pp. 277–293, 2000.

[16] G. Jeh and J. Widom. SimRank: A measure of structural-context similarity. *Proc. of 8th SIGKDD*, pp. 538–543, 2002.

[17] G. Jeh and J. Widom. Scaling personalized web search. *Proc. of 12th WWW*, pp. 271–279, 2003.

[18] S. Kamvar, T. H. Haveliwala, C. Manning, and G. Golub. Exploiting the block structure of the web for computing PageRank. Technical Report 2003-17, Stanford University, 2003.

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

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

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

[22] F. McSherry. A uniform approach to accelerated PageRank computation. *Proc. of 14th WWW*, pp. 575–582, 2005.

[23] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Comp. Sci.*, 1(2), 2005.

[24] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford University, 1998.

[25] C. R. Palmer, P. B. Gibbons, and C. Faloutsos. ANF: A fast and scalable tool for data mining in massive graphs. *Proc. of 8th SIGKDD*, pp. 81–90, 2002.

[26] P. K. C. Singitham, M. S. Mahabhashyam, and P. Raghavan. Efficiency-quality tradeoffs for vector score aggregation. *Proc. of 30th VLDB*, pp. 624–635, 2004.

[27] J. S. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33(2):209–271, 2001.