A Sublinear Time Algorithm for PageRank Computations

Christian Borgs[1], Michael Brautbar[2], Jennifer Chayes[1], and Shang-Hua Teng[3]

[1] Microsoft Research New England, One Memorial Drive, Cambridge, MA 02142
{borgs,jchayes}@microsoft.com
[2] Computer and Information Science Department, University of Pennsylvania,
3330 Walnut Street, Philadelphia, PA 19104
brautbar@cis.upenn.edu
[3] Computer Science Department, University of Southern California,
941 Bloom Walk, Los Angeles, CA 90089
shanghua@usc.edu

Abstract. In a network, identifying all vertices whose PageRank is more than a given threshold value Δ is a basic problem that has arisen in Web and social network analyses. In this paper, we develop a nearly optimal, sublinear time, randomized algorithm for a close variant of this problem. When given a directed network $G = (V, E)$, a threshold value Δ, and a positive constant $c > 3$, with probability $1 - o(1)$, our algorithm will return a subset $S \subseteq V$ with the property that S contains all vertices of PageRank at least Δ and no vertex with PageRank less than Δ/c. The running time of our algorithm is always $\tilde{O}(\frac{n}{\Delta})$. In addition, our algorithm can be efficiently implemented in various network access models including the Jump and Crawl query model recently studied by [6], making it suitable for dealing with large social and information networks.

As part of our analysis, we show that any algorithm for solving this problem must have expected time complexity of $\Omega(\frac{n}{\Delta})$. Thus, our algorithm is optimal up to logarithmic factors. Our algorithm (for identifying vertices with significant PageRank) applies a multi-scale sampling scheme that uses a fast personalized PageRank estimator as its main subroutine. For that, we develop a new local randomized algorithm for approximating personalized PageRank which is more robust than the earlier ones developed by Jeh and Widom [9] and by Andersen, Chung, and Lang [2].

1 Introduction

A basic problem in network analysis is to identify the set of its vertices that are "significant." For example, the significant nodes in the web graph defined by a query could provide the authoritative content in web search; they could be the critical proteins in a protein interaction network; and they could be the set of people (in a social network) most effective to seed the influence for online advertising. As the networks become larger, we need more efficient algorithms to identify these "significant" nodes.

1.1   Identifying Nodes with Significant PageRanks: Our Results

The meaning of 'significant' vertices depend on the semantics of the network and the applications. In this paper, we focus on a particular measure of significance — the PageRanks of the vertices. PageRank was introduced by Page and Brin in their seminal work for ranking webpages [11]. Mathematically, the PageRank (with restart constant, also known as the teleportation constant, α) of a web-page is proportional to the the probability that the page is visited by a random surfer who explores the web using the following simple random walk: at each step, with probability $(1 - \alpha)$ go to a random webpage linked to from the current page, and with probability α, restarts the process from a randomly chosen page. For reasons to be cleared shortly, we consider a normalization of the PageRank so that the sum of the PageRank values over all vertices is equal to n, the number of vertices in the network. In other words, suppose PageRank(u) denote the PageRank of vertex u in the network $G = (V, E)$. Then,

$$\sum_{u \in V} \text{PageRank}(u) = n.$$

PageRank has been used by the Google search engine and has found applications in wide range of data analysis problems [4, 7]. In this context, the problem of identifying "significant" vertices could be illustrated by the following search problem: Let Top PageRanks denote the problem of identifying all vertices whose PageRanks in a network $G = (V, E)$ are more than a given threshold value $1 \leq \Delta \leq |V|$.

In this paper, we consider for the following close variant of Top PageRanks:

Significant PageRanks: *Given a network $G = (V, E)$, a threshold value $1 \leq \Delta \leq |V|$ and a positive constant $c > 1$, compute, with success probability $1 - o(1)$, a subset $S \subseteq V$ with the property that S contains all vertices of PageRank at least Δ and no vertex with PageRank less than Δ/c.*

We develop a nearly optimal, sublinear time randomized algorithm for Significant PageRanks for any fixed $c > 3$. The running time of our algorithm is always $\tilde{O}(\frac{n}{\Delta})$. We show that any algorithm for Significant PageRanks must have time complexity of $\Omega(\frac{n}{\Delta})$. Thus, our algorithm is optimal up to logarithmic factors. Our Significant PageRanks algorithm applies a multi-scale sampling scheme that uses a fast personalized PageRank estimator (see below) as its main subroutine.

1.2   Matrix Sampling and Personalized PageRank Approximation

While the PageRank of a vertex captures the importance of the vertex collectively assigned by all vertices in the network, as pointed out by Haveliwala [8], one can use the distributions of the following random walk to define the pairwise contributions of significances: Given a teleportation probability α and a starting

vertex u in a network $G = (V, E)$, at each step, with probability $(1 - \alpha)$ go to a random neighboring vertex, and with probability α, restarts the process from u. For $v \in V$, the probability that v is visited by this random process, denoted by $\text{PersonalizedPageRank}_u(v)$, is the u's personal PageRank contribution of significance to v. It is not hard to verify that

$$\forall u \in V, \quad \sum_{v \in V} \text{PersonalizedPageRank}_u(v) = 1; \text{ and}$$

$$\forall v \in V, \quad \text{PageRank}(v) = \sum_{u \in V} \text{PersonalizedPageRank}_u(v).$$

Personalized PageRanks has been widely used to describe personalized behavior of web-users [11] as well as for designing good network clustering techniques [2]. As a result, fast algorithms for computing or approximating personalized PageRank can be very useful. One can approximate PageRanks and personalized PageRanks by the power method [4], which involves costly matrix multiplications for large scale networks. Applying effective truncation, Jeh and Widom [9] and Andersen, Chung, and Lang [2] developed personalized PageRank approximation algorithms that can find an ϵ-additive approximation in time proportional to the product of ϵ^{-1} and the maximum in-degree in the graph.

Our sublinear-time algorithm for SIGNIFICANT PAGERANKS also requires fast subroutines for estimating personalized PageRanks. It uses a multi-scale sampling approach by selecting a set of precision parameters $\{\epsilon_1, ..., \epsilon_h\}$ where h depends on n and Δ, $\epsilon_i = 1/2^i$. Then, for each i in range $1 \leq i \leq h$, it computes the ϵ_i-precise personalized PageRanks defined by a sample of $\tilde{O}(\epsilon_i n/\Delta)$ vertices. For networks with constant maximum degrees, we can simply use the Jeh-Widom or Andersen-Chung-Lang personalized PageRank approximation algorithms in our multi-scale sampling scheme. However, for networks such as web graphs and social networks that may have nodes with large degrees, these two earlier algorithms are not robust enough for our purpose.

We develop a new local algorithm for approximating personalized PageRank that satisfies the desirable robust property that the multi-scale sample scheme requires. Given $\rho, \epsilon > 0$ and a starting vertex u in a network $G = (V, E)$, our algorithm estimates each entry in the personalized PageRank vector,

$$\text{PersonalizedPageRank}_u(.)$$

defined by u to a multiplicative factor of at most $(1+\rho)$ plus an additive precision error of at most ϵ [4]. The time complexity of our algorithm is $O(\frac{\log(|V|)\log(\epsilon^{-1})}{\epsilon\rho^2})$. Our algorithm requires a careful simulation of random walks from the starting node u to ensure that its complexity does not depends on the degree of any node.

Our algorithms can be efficiently implemented in various network querying models assuming no direct global access to the network. In particular, our algorithms can be efficiently implemented in the Jump and Crawl query model [6],

---

[4] Formally, estimated value $\hat{val}$ of val would have the property that $(1 - \rho) \cdot val - \epsilon \leq \hat{val} \leq (1 + \rho) \cdot val + \epsilon$.

making the algorithm suitable for processing large social and information networks.

In particular, our sublinear algorithm for SIGNIFICANT PAGERANKS could be used in Web search engines, which often need to build a core of web-pages to be later used for web-search. It is desirable that pages in the core have high PageRank values. These search engines usually apply crawling to discover new significant pages and add them to the core. The property that our sublinear-time algorithms have a natural implementation in the Jump and Crawl model may make them useful in a search engine for selecting pages with high PageRank values to update the current core by using them to replace the existing core pages that have relatively low PageRank values. We anticipate that our algorithm for SIGNIFICANT PAGERANKS will be useful for many other network analysis tasks.

1.3 Additional Related Work

For personalized PageRanks approximation, in addition to the work of [2, 4, 5, 9, 11], Andersen *et al* [1] developed a 'backward' version of the local algorithm of [2]. Their algorithm finds all nodes that contribute at least some fixed fraction ρ to a page's PageRank in time $O(d_{\text{max-out}}/\rho)$ where $d_{\text{max-out}}$ is the maximum out-degree in the network. This algorithm can be used to provide some reliable estimate to a node's PageRank. For example, for a given k, in time $\tilde{\Theta}(k)$ it can bound the total contribution from the k highest contributors to the node's PageRank. However, for networks with large $d_{\text{max-out}}$, its complexity may not be sublinear.

As suggested in [1], one can view the entire set of personalized PageRanks (defined by all vertices in a network) as an $|V| \times |V|$ matrix, which is referred to as the *PageRank matrix* of the graph. In the PageRank matrix, each row represents the personalized PageRanks from a particular vertex, and each column represents the contributions to its PageRank from all vertices in the network. Note that the sum of each row is 1 and the sum of the uth column is the PageRank of u.

In light of this, the problem of SIGNIFICANT PAGERANKS can be viewed as a matrix sparsification or matrix approximation problem. There has been a large body of work of finding a low complexity approximation to a matrix that preserves some of its properties. Perhaps the most relevant one to our goal is a low rank matrix approximation under the l_2 matrix norm.

All current methods for finding such low rank approximations runs in time at least linear in the size of the input matrix. See [10] for a survey of recent results.

Next, a linear time Monte Carlo based method to estimate PageRank of all nodes is devised in [3]. The method is based on running constant number of random walks from each of the nodes in the network.

Last, in the context of sublinear time graph algorithms, our research is related to the work of [12], in which sublinear time algorithms are presented for estimating several quantities. Our implementation of the Jump and Crawl query model can be viewed as a stringent type of the adjacency-list graph model used in [12].

1.4   Organization

Section 2 contains the needed definitions and notations. Section 3 presents our multi-scale sampling algorithm for SIGNIFICANT PAGERANKS. In section 4 we provide a lower bound construction for SIGNIFICANT PAGERANKS. In Section 5 we give a robust local algorithm for approximating personalized PageRank vectors.

2   Preliminaries

We consider a network which is defined as a directed graph $G = (V, E)$ with n nodes and m edges. Usually, a network is massive. Our algorithms access a network using a rather natural implementation of the Jump-and-Crawl query model of [6] developed for processing large social and information networks. The Jump and Crawl model is concerned with informational complexity of nodes, where each node access reveal its full list of adjacent neighbors at no extra cost. Our algorithms shall be designed to work under the following compelling implementation of the Jump-and-Crawl query model. We allow two types of queries:

- *Jump*: A call to the Jump query needs no input and returns a uniformly at random node from the network.
- *RandomCrawl*: A call to the RandomCrawl query requires a vertex v as input. RandomCrawl(v) returns a uniformly at random out-neighbor of v.

Note for example, that the random surfer procedure used in the definition of PageRank is itself a natural algorithm under our implementation of the Jump-and-Crawl query model.

We now move to define personalized PageRank as well as PageRank. Mathematically, the personalized PageRank vector of a node v is the stationary point of the following equation:

$$\text{PersonalizedPageRank}_v(\cdot) = \alpha \mathbf{1}_v + (1 - \alpha)\text{PersonalizedPageRank}_v(\cdot) \cdot D^{-1}A,$$

where α is the teleportation probability, A is the adjacency matrix of the directed network $G = (V, E)$ so $A(i, j) = 1$ iff $(i, j) \in E$. In this notation, D is a diagonal matrix with $d_{out}(v)$ at entry (v, v) and $\mathbf{1}_v$ is the indicator vector of v. We will follow the standard [4] by assuming that each node has at least one out-link [5].

Then, one can define the RageRank vector as

$$\text{PageRank}(\cdot) = \sum_{v \in V} \text{PersonalizedPageRank}_v(\cdot)$$

Note that in this definition, the sum of the all PageRank values is equal to n.

Following [1], we define a matrix PPR (short for personalized PageRank) to be the $n \times n$ matrix, whose v^{th} row is PersonalizedPageRank$_v(\cdot)$.

Unless stated otherwise, for any x, $\log(x)$ would mean $\log_2(x)$.

---

[5] Otherwise, as commonly done [4], consider that node as having out links into all nodes in the network.

3 Multi-scale Matrix Sampling and Approximation of PageRank

In this section, we present our nearly optimal, sublinear time algorithm for SIG-NIFICANT PAGERANKS. Recall that

> SIGNIFICANT PAGERANKS: *Given a network $G = (V, E)$, a threshold value $1 \leq \Delta \leq |V|$ and a positive constant $c > 1$, compute, with probability $1 - o(1)$, a subset $S \subseteq V$ with the property that S contains all vertices of PageRank at least Δ and no vertex with PageRank less than Δ/c.*

Note that the PageRank value of each vertex is at least α and at most n. Instrumental to our algorithm, we present a multi-scale algorithm for sampling the PageRank matrix PPR that achieves, for any fixed $c > 3$, the following goals: The algorithm makes $\tilde{O}(\frac{n}{\Delta})$ total queries and updates, and with high probability,

1. For each vertex with PageRank value at least Δ, the sum of the sampled entries of of the column corresponding to the vertex will provide a quality estimate to the PageRank value of that vertex.
2. The algorithm does not return any vertex whose PageRank value is less than Δ/c.

In our algorithm, we will use a new local algorithm *ApproxRow* for personalized PageRank approximation. Algorithm *ApproxRow* takes three input parameters: $v \in V$, an additive error factor $\epsilon \in (0, 1)$ and a multiplicative factor $\rho \in (0, 1)$. It returns an approximation to PersonalizedPageRank$_v(\cdot)$ such that for every PPR$(v, j) > \epsilon$, it returns a non-negative estimated value between $(1 - \rho)$PPR$(v, j) - \epsilon$ to $(1 + \rho)$PPR$(v, j) + \epsilon$. The running time of *ApproxRow* is essentially $O(\frac{\log(n)\log(\epsilon^{-1})}{\epsilon\rho^2})$. *ApproxRow* and its analysis will be presented in Section 5.

We start with some high-level idea of our multi-scale sampling algorithm. To assist our exposition, we will present our algorithm and its analysis for $c = 6$. Both are easily extended to any other constant value $c > 3$. Our algorithm will use $O(\log n)$ precision scales: $\epsilon_t = 2^{-t}$ for $0 \leq t \leq \log(\frac{4n}{\Delta})$. We conceptually divide each column of the PPR matrix into *chunks*, where the chunk corresponding to ϵ_t contains its entries with values between ϵ_t to $2\epsilon_t$. Thus, we ignore all entries in the PPR matrix column of value less than $\frac{\Delta}{4n}$, the finest scale. Note that entries with value at most $\frac{\Delta}{4n}$ can contribute to at most a quarter to the PageRank of a vertex whose PageRank value is least Δ.

If the sum of a chunk's entries is at least $\Delta/(2\log(n))$, we will refer to it as a *heavy chunk*. The central idea of our algorithm is to efficiently generate robust estimates of the sums for all heavy chunks, as we shall show that it is also sufficient to only provide estimates to heavy chunks.

As the entries in each chunk are within a factor of 2 of each other, we then reduce the task of estimating the sum in a chunk to the problem of approximately counting the size of the chunk. Then conceptually, we estimate the size of each heavy chunk at scale ϵ_t by taking $\tilde{O}(\epsilon_t 4n/\Delta)$ random entries from its column

and counting the numbers of samples in this chunk. The challenge we need to overcome is to efficiently sample all heavy chunks at a scale simultaneously.

This is where we will use our local PageRank approximation algorithm ApproxRow, which in $O(\frac{\log(n)\log(\epsilon^{-1})}{\epsilon})$ time when given a vertex v, returns robust estimates to all entries of values at least ϵ in v's row in the PPR matrix. To achieve $\tilde{O}(n/\Delta)$ queries and running time, we call ApproxRow $\tilde{\Theta}(\frac{n}{\Delta}\epsilon_t)$ times at scale ϵ_t, and we will show that it is sufficient to sample this much (or little).

In the last step of the algorithm, for each node j, we will simply sum up over all scales ϵ_t, its estimated values weighted by a normalizing factor $\frac{\Delta}{\epsilon_t 2 \log^2(n)}$. Then the algorithm will output only those j's where the sum is at least $\frac{\Delta}{4}$ and their estimated PageRank values.

A detailed pseudo-code of our algorithm, *ApproximatePageRank*, is given below.

Algorithm 1 ApproximatePageRank

Require: PageRank threshold Δ, a network $G = (V, E)$ on n nodes accessible only by Jump and RandomCrawl queries.
 // **First-Part** //
  1: Initialize a binary search tree, ChunkTree, indexed lexicographically by a two-tuple key $(nodeID, \epsilon)$.
  2: **for** $t = 0$ to $\log(\frac{n}{4\Delta})$ **do**
  3: Set the additive error $\epsilon_t = 2^{-t}$.
  4: **for** $(\frac{n}{\Delta}\epsilon_t 4 \log^2(n))$ times **do**
  5: Jump to a random node, call it v.
  6: Call list = ApproxRow$(v, \frac{\epsilon_t}{2}, \frac{1}{2})$ and update the chunk size estimate affiliated vertices in list as the following:
  7: **for** each pair $(nodeID, \epsilon_t)$ in the list **do**
  8:   **if** there exists an entry e with key $(nodeID, \epsilon_t)$ in ChunkTree **then**
  9:    Update entry e's value by adding 1 to its current value.
10:   **else**
11:    Create an entry in ChunkTree with key $(nodeID, \epsilon_t)$ and value 1.
12:   **end if**
13: **end for**
14: **end for** (at scale ϵ_t).
15: **end for** (for all scales)
 // **Second-Part** //
16: Initialize a final tree, called TreeofPageRankValues, indexed by key $(nodeID)$.
17: **for** all elements (chunks) in ChunkTree that all belong to same node i (namely, have i as the first part of their key) **do**
18: **if** chunk has value, val, at least $\frac{1}{2}\log(n)$ **then**
19: Let ϵ be the second part of the chunk's key.
20: Add $\frac{\Delta}{2\epsilon \log^2(n)}$ to the entry indexed by (i) in TreeofPageRankValues.
21: **end if**
22: Output all elements in TreeofPageRankValues with at least $\Delta/4$
23: **end for**

In the proofs for the following two theorems, we will analyze the performance of this algorithm. Note that we will ignore the dependence of the running time on α as for all standard PageRank computations, it is taken to be a fixed constant independent of input size [4].

Theorem 1 (Complexity of ApproximatePageRank). *The runtime of algorithm ApproximatePageRank is upper bounded by $\tilde{O}(n/\Delta)$.*

Proof The algorithm uses $O(\log(n/\Delta))$ scales. In *First-Part* of the algorithm, for scale ϵ_t, it makes $\frac{n}{\Delta}\epsilon_t 4\log^2(n)$ Jump queries and for each query it runs ApproxRow$(v, \epsilon_t/2, 1/2)$, where v represents the random vertex returned by the query. ApproxRow then has a runtime of $O(\frac{\log(n)\log(\epsilon_t^{-1})}{\epsilon_t})$. Thus, the total run-time complexity is $\tilde{O}(\frac{n}{\Delta})$ as the finest scale is Δ/n and there are at most $\log n$ scales. In addition to the time spent on querying the network, the algorithm takes $\Theta(\log(n))$ per step overhead for each access/update in its data structure.

In *Second-Part* of the algorithm, it makes no new queries. As there are only $\tilde{O}(n/\Delta)$ items in the data structure ChunkTree and then TreeofPageRankValues, the complexity of this summation part is $\tilde{O}(n/\Delta)$. The last step of outputting all nodes in the tree with value bigger than a threshold can easily be done in linear time in the size of the tree, which is $\tilde{O}(n/\Delta)$. $\square$

Theorem 2 (Correctness of ApproximatePageRank). *Given Δ and constant $c > 3$, ApproximatePageRank outputs, with probability $1 - o(1)$, all nodes with PageRank at least Δ but no node with PageRank smaller than[6] Δ/c.*

Proof For $v \in V$, let $(p_1^v, p_2^v, \ldots, p_n^v)$ be v's column in the PPR matrix. Let ChunkSet$(v, \epsilon) = \{i : \epsilon \leq p_i^v < 2\epsilon\}$, ChunkSize$(v, \epsilon) = |\text{ChunkSet}(v, \epsilon)|$, and ChunkSum$(v, \epsilon) = \sum_{i=1}^n \{p_i^v : \epsilon \leq p_i^v < 2\epsilon\}$.

Recall a chunk is heavy is its chunksum $\Delta/\log(n)$. We now prove that at the end of First-Part in Algorithm ApproximatePageRank, all heavy chunks are well approximated.

To focus on the essence of the proof for multi-scale matrix sampling, we first assume that all the values returned by ApproxRow are exact (with no error at all). We call this assumption, the *perfect row approximation assumption*. We will later show that when removing this assumption the approximation scheme would only be affected by a multiplicative factor of three, namely the effective value of c in Significant PageRanks would be one third its value under perfect row approximations.

Lemma 1 (key lemma). *Let $\epsilon_t = 2^{-t}$, for $1 \leq t \leq \frac{n}{4\Delta}$. The following holds with probability $1 - o(1)$:*

- *If ChunkSum$(v, \epsilon_t) \geq \frac{\Delta}{2\log(n)}$ then at the end of First-Part in the algorithm the entry in ChunkTree with key (v, ϵ_t), namely the algorithm's approximation of ChunkSize(v, ϵ_t), is at least $\log n/2$ and is between $\frac{ChunkSize(v,\epsilon_t)}{\Delta} \cdot \epsilon_t 2\log^2(n)$ to $\frac{ChunkSize(v,\epsilon_t)}{\Delta} \cdot \epsilon_t 8\log^2(n)$.*

---

[6] Again for exposition, we present our algorithm and its analysis for $c = 6$. We later show that the theorem on a slightly modified algorithm holds for any constant $c > 3$.

- *If $ChunkSum(v, \epsilon_t) \leq \frac{\Delta}{4 \log(n)}$ then at the end of First-Part in the algorithm the entry in ChunkTree with key (v, ϵ_t), namely the algorithm's approximation of $ChunkSize(v, \epsilon_t)$, is smaller than $\frac{\log(n)}{2}$.*

Proof Note that $\frac{1}{2\epsilon_t} ChunkSum(v, \epsilon_t) \leq ChunkSize(v, \epsilon_t) \leq \frac{1}{\epsilon_t} ChunkSum(v, \epsilon_t)$.

So, if $ChunkSum(v, \epsilon_t) \geq \frac{\Delta}{2 \log(n)}$ then $ChunkSize(v, \epsilon_t)/n \geq \Delta/(4\epsilon_t n \log n)$.

Thus, when sampling $4\epsilon_t n \log^2(n)/\Delta$ random rows (as in line 5 of the algorithm), the expected number of entries in the chunk that ApproxRow discovers is at least $ChunkSize(v, \epsilon_t)\epsilon_t 4 \log^2(n)/\Delta \geq \log(n)$. By a standard multiplicative Chernoff bound (see appendix), with probability $1 - o(1)$, after multiplying the count by $\Delta/(2\epsilon_t \log^2 n)$, we can approximate $ChunkSize(v, \epsilon_t)$ within a multiplicative factor of 2. Moreover, if $ChunkSum(v, \epsilon_t) \leq \frac{\Delta}{4 \log(n)}$ then, its estimated value is at most twice its value, namely smaller than $\frac{\log(n)}{2}$. □

Lemma 2. *The following holds with probability $1 - o(1)$ under the perfect row approximation assumption:*

- *If $PageRank(v) \geq \Delta$, then the algorithm will output v and will estimate its PageRank value to a value between $PageRank(v)/4$ to $2PageRank(v)$.*
- *If $PageRank(v) < \Delta/8$, then the algorithm will not output v.*
- *If $\Delta/8 \leq PageRank(v) < \Delta$, then the algorithm might output v. If v is outputted, then its estimated PageRank value is between $PageRank(v)/16$ to $2PageRank(v)$.*

Proof By lemma 1, that the sums of each heavy chunk are well estimated to within a multiplicative factor of 2.

Since there are at most $\log n$ chunks in column, the contribution from all non-heavy chunks is at most $\log n(\Delta/(2 \log n)) = \Delta/2$. Thus, if v's PageRank is at least Δ, then the contribution from its heavy chunks is at least $\Delta/2$. Consequently, and the algorithm's approximation to v's PageRank will be at least $\Delta/4$ and at most 2Δ, and this vertex will be outputted.

We can similarly establish the other two cases as stated in the lemma. □

We now turn to discuss the effect of having only approximate values computed in ApproxRow calls on the guarantees of ApproxmatePageRank.

Lemma 3. *Given parameters $0 < \epsilon, \rho < 1$, removing the perfect row approximation assumption changes the approximation constant c by at most 3 times its value as well as changes the estimated PageRank values computed by ApproximatePageRank to be at most three times their value.*

Proof The PPR matrix is effectively computed using calls to ApproxRow by the algorithm.

Given $\epsilon > 0$, consider an element $\epsilon \leq PPR(v, j) \leq 2\epsilon$ for some nodes v, j. There are two sources for having this element estimator differ from it real value. First, ApproxRow (with parameters $\epsilon = \epsilon/2$ and $\rho = 1/2$) computes approximate values so the estimated value is between $(1 - \rho - 1/4)$ its real value to $(1 + \rho + 1/4)$ (we could put the additive $\epsilon/4$ error in the multiplicative approximation

factor since $\epsilon \leq PPR(v,j) \leq 2\epsilon$). In particular, in the algorithm we pick $\rho = 1/2$. However, one could replace both ρ and ϵ with smaller values to get an approximation that gets closer to the true value: Replacing ρ by $k_1\rho$ and ϵ by $k_2\epsilon$ for any integral k_1, k_2 would increase the total runtime by only a factor of $k_1^2 k_2 \frac{\log(k_2)}{\log(\epsilon^{-1})}$. The second source why the estimator differs from its true value is double counting. An element with a true value between $\epsilon/2$ to ϵ as well as one with a true value between 2ϵ to 4ϵ could appear in a realization as an element with a value between ϵ to 2ϵ. However (by applying Chernoff bound), elements with true value smaller than $\epsilon/2$ as well as those with value bigger than 4ϵ would not appear as such. Thus due to double counting the sum of elements in each column can be at most three times its real value. If we denote the PageRank of node j by $\Delta(j)$ and the value it gets from the realized column values by $\Delta'(j)$ then,

$$(1 - k_1\rho - k_2/2)\Delta(j) \leq \Delta'(j) \leq 3(1 + k_1\rho + k_2/2)\Delta(j).$$

In particular, algorithm ApproximatePageRank uses $\rho = \frac{1}{2}$, $k_1 = 1$ and $k_2 = \frac{1}{2}$ which gives

$$\frac{1}{4}\Delta(j) \leq \Delta'(j) < 6\Delta(j).$$

$\square$

This ends the proof of Theorem 2. $\square$

4   Lower Bound Construction for PageRank Approximations

We now turn to prove a corresponding lower bound for PageRank approximations.

The lower bound construction will show that, any algorithm, making less than $\Omega(\frac{n}{\Delta})$ Jump and Crawl queries, will fail, with constant probability, to find any node with PageRank at least Δ on the graph. This holds true for any type of implementation of a Crawl query (including the RandomCrawl one). Given positive integers n and $\Delta < \frac{n}{2} - 1$, we construct an undirected graph on n nodes made of a path subgraph on $n - d - 1$ nodes and an isolated star subgraph on $d + 1$ nodes, where $d = 2\Delta$. See figure 1 for an illustration. Fix $0 < \alpha < 1$, the teleportation probability. By solving the PageRank equations it is not hard to check that each node on the path subgraph has PageRank value of 1, the hub of the subgraph has PageRank $\frac{d}{2} + \frac{1}{2(1-\alpha)}$ and each leaf of the star subgraph has PageRank of $\frac{1}{d}(d + 1 - \frac{d}{2} - \frac{1}{2(1-\alpha)}) \leq \frac{1}{2} + \frac{1}{d}$. As $\Delta = \frac{d}{2}$, the only node with PageRank at least Δ is the hub of the star subgraph. However, for any $\epsilon > 0$, in order to find any node that belongs to the star subgraph one needs to make, with probability at least $1 - \frac{1}{e} - \epsilon$, at least $\frac{n}{d} = \Omega(\frac{n}{\Delta})$ Jump queries.
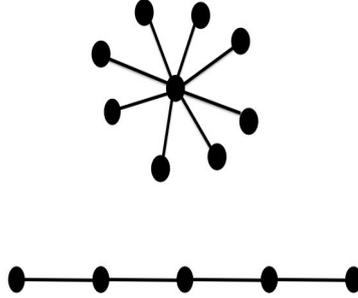
Fig. 1. An example illustrating the path-star graph of the lower bound construction for PageRank computations.

5   Local Robust Computation of Personalized PageRank

We now describe a method, *ApproxRow*, based only on local computations that approximates a node's personalized PageRank vector. The pseudo-code is given on the next page.

Theorem 3 (Complexity of ApproxRow). *For any node v and values $0 < \epsilon, \rho < 1$, the runtime of $ApproxRow(v, \epsilon, \rho)$ is upper bounded by $O(\frac{\log(n)\log(\epsilon^{-1})}{\epsilon\rho^2 \log_2(\frac{1}{1-\alpha})})$.*

Proof   The algorithm performs $\frac{1}{\epsilon\rho^2} \cdot 16\log(n)$ rounds where at each round it simulates a random walk with termination probability of α for at most *length* steps. Each step is simulated by taking a Jump ('termination' step) with probability α and taking a RandomCrawl step with probability $1 - \alpha$. Thus the total number of queries used is $\frac{16\log(n)}{\epsilon\rho^2} \cdot \log_{\frac{1}{(1-\alpha)}}(\frac{3}{\epsilon}) = O(\frac{\log(n)\log(\epsilon^{-1})}{\epsilon\rho^2 \log_2(\frac{1}{1-\alpha})})$.   □

Theorem 4 (Correctness of ApproxRow). *For any node v and values $0 < \epsilon, \rho < 1$, with probability of at least $1 - \Theta(\frac{1}{n^2})$, $ApproxRow(v, \epsilon, \rho)$ computes a list l with the following properties:*

- *Every node j that is outputted in the list l has an estimated value which is non-negative and lies between $(1 - \rho)PPR(v, j) - \frac{\epsilon}{4}$ to $(1 + \rho)PPR(v, j)$.*
- *Every node not in the list l has $PPR(v, j) \leq \epsilon/2$.*

Proof   We start with an observation. The personalized PageRank contribution from a node v to node j is exactly the probability that a random walk that starts at v, and at each time step terminates with probability α, and with probability

---

Algorithm 2 ApproxRow

Require: A node v in $G = (V, E)$, additive error parameter $0 < \epsilon < 1$, multiplicative
approximation parameter $0 < \rho < 1$, teleportation probability $0 < \alpha < 1$.
1: Initialize a binary search tree NodeCountTree where the key is a node's identity.
2: Set $length = \log_{\frac{1}{(1-\alpha)}} (\frac{4}{\epsilon})$.
3: Set $r = \frac{1}{\epsilon\rho^2} \cdot 16 \log(n)$.
4: **for** r times **do**
5:    Run one realization of a random walk with restart probability α: the walk starts
at v and at each time makes, with probability α a 'termination' step by returning
to v and terminating, and with probability $1 - \alpha$ a RandomCrawl step. The walk
is artificially stopped after $length$ steps if it has not terminated already.
6:    **if** the walk visited a node u just before making a termination step **then**
7:        Add 1 to the count stored at u's entry in NodeCountTree.
8:    **end if**
9:    Output all nodes in NodeCountTree together with their average count (over the
r rounds).
10: **end for**

$1 - \alpha$ moves to a random out-link of the node it is currently at, was at node
j one step before termination. Define $\mathbf{1}_v$ to be the indicator vector of v. The
proof of the observation follows from a series of algebraic manipulations on the
definition of the PersonalizedPageRankVector of v:

$$\text{PersonalizedPageRank}_v(\cdot) = \alpha \mathbf{1}_v + (1 - \alpha)\text{PersonalizedPageRank}_v(\cdot) \cdot D^{-1}A.$$

Solving the system gives $\text{PersonalizedPageRank}_v(\cdot) = \alpha \mathbf{1}_v (I - (1-\alpha)D^{-1}A)^{-1} = \alpha \mathbf{1}_v \sum_{i=0}^{\infty} ((1 - \alpha)D^{-1}A)^i$. This last equation makes the observation clear.

Given a node j, denote by $p_k(v, j)$ the contribution to v's Personalized PageRank vector from walks that are of length at most k. By the above observation, $p_k(v, j) = \alpha \mathbf{1}_v \sum_{i=0}^{k} ((1 - \alpha)D^{-1}A)^i$.

We ask how much is contributed to j's entry in the Personalized PageRank vector of v from walks of length bigger or equal to k. The contribution is at most $(1 - \alpha)^k$ since the walk needs to survive at least k consecutive steps. Taking $(1 - \alpha)^k \leq \frac{\epsilon}{4}$ will guarantee that at most $\frac{\epsilon}{4}$ is lost by only considering walks of length smaller than k, namely: $\text{PPR}(v, j) - \frac{\epsilon}{4} \leq p_k(v, j) \leq \text{PPR}(v, j)$.

For that it suffices to take $k = \log_{\frac{1}{(1-\alpha)}} (\frac{4}{\epsilon})$. This is exactly $length$, the length of each walk the algorithm simulates, is set to.

Next, the algorithm provide a estimate of $p_k(v, j)$ by realizing walks of length at most k. The algorithm does so by taking the average count over $\frac{1}{\epsilon\rho^2} \cdot 16 \log(n)$ trials. Denote the algorithm's output by $\hat{p}_k(v, j)$. Then, if $p_k(v, j) \geq \frac{\epsilon}{4}$, by the multiplicative Chernoff bound, $Pr(\hat{p}_k(v, j) > (1 + \rho)p_k(v, j)) \leq \exp(-2\log(n))$ and $Pr(\hat{p}_k(v, j) < (1 - \rho)p_k(v, j)) \leq \exp(-2\log(n))$.

We can conclude that $(1 - \rho)(\text{PPR}(v, j) - \frac{\epsilon}{4}) \leq \hat{p}_k(v, j) \leq (1 + \rho)\text{PPR}(v, j)$.

In particular, nodes with $\text{PPR}(v, j) > \epsilon$ will be estimated to a positive value and outputted as claimed.

Similarly, if $p_k(v,j) \leq \frac{\epsilon}{4}$ then by the multiplicative Chernoff bound, $Pr(\hat{p}_k(v,j) > \frac{\epsilon}{2}) \leq \exp(-2\log(n))$. In this case we conclude that $\hat{p}_k(v,j) \leq \frac{\epsilon}{2}$. Also, $\text{PPR}(v,j) \leq \frac{\epsilon}{4} + \frac{\epsilon}{4} \leq \frac{\epsilon}{2}$ so $|\text{PPR}(v,j) - \hat{p}_k(v,j)| \leq \frac{\epsilon}{2}$. $\qquad\square$

6 Acknowledgments

References

1. Reid Andersen, Christian Borgs, Jennifer T. Chayes, John E. Hopcroft, Vahab S. Mirrokni, and Shang-Hua Teng. Local computation of pagerank contributions. *Internet Mathematics*, 5(1):23–45, 2008.
2. Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. Local graph partitioning using pagerank vectors. In *FOCS*, pages 475–486, 2006.
3. K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova. Monte carlo methods in pagerank computation: When one iteration is sufficient. *SIAM Journal on Numerical Analysis*, 45, 2007.
4. Pavel Berkhin. Survey: A survey on pagerank computing. *Internet Mathematics*, 2(1), 2005.
5. Pavel Berkhin. Bookmark-coloring approach to personalized pagerank computing. *Internet Mathematics*, 3(1), 2006.
6. Mickey Brautbar and Michael Kearns. Local algorithms for finding interesting individuals in large networks. In *ICS*, pages 188–199, 2010.
7. Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.
8. T.H Haveliwala. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. In *Trans. Knowl. Data Eng*, volume 15(4), pages 784–796, 2003.
9. Glen Jeh and Jennifer Widom. Scaling personalized web search. In *WWW*, pages 271–279, 2003.
10. Ravindran Kannan. Spectral methods for matrices and tensors. In *STOC*, pages 1–12, 2010.
11. Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Stanford University 1998.
12. Ronitt Rubinfeld and Asaf Shapira. Sublinear time algorithms. *SIAM Journal on Discrete Math*, 25:1562–1588, 2011.

Appendix: Concentration Bounds

Lemma 4 (multiplicative Chernoff bound). *Let X_i be i.i.d. Bernoulli random variables with expectation μ each. Define $X = \sum_{i=1}^{n} X_i$. Then,*

– *For $0 < \lambda < 1, Pr[X < (1-\lambda)\mu n] < \exp(-\mu n \lambda^2/2)$.*
– *For $0 < \lambda < 1, Pr[X > (1+\lambda)\mu n] < \exp(-\mu n \lambda^2/4)$.*
– *For $\lambda \geq 1, Pr[X > (1+\lambda)\mu n] < \exp(-\mu n \lambda/2)$.*