Multi-Scale Matrix Sampling and Sublinear-Time PageRank Computation[*]

Christian Borgs[†]   Michael Brautbar[‡]   Jennifer Chayes[§]   Shang-Hua Teng[¶]

Abstract

A fundamental problem arising in many applications in Web science and social network analysis is the problem of identifying all nodes in a network whose PageRank exceeds a given threshold Δ. In this paper, we study the probabilistic version of the problem where given an arbitrary approximation factor $c > 1$, we are asked to output a set S of nodes such that with high probability, S contains all nodes of PageRank at least Δ, and no node of PageRank smaller than Δ/c. We call this problem SIGNIFICANTPAGERANKS.

We develop a nearly optimal, local algorithm for the problem with runtime complexity $\tilde{O}(n/\Delta)$ on networks with n nodes, where the tilde hides a polylogarithmic factor. We show that any algorithm for solving this problem must have runtime of $\Omega(n/\Delta)$, rendering our algorithm optimal up to logarithmic factors. Our algorithm has sublinear time complexity for applications including Web crawling and Web search that require efficient identification of nodes whose PageRanks are above a threshold $\Delta = n^\delta$, for some constant $0 < \delta < 1$.

Our algorithm comes with two main technical contributions. The first is a multi-scale sampling scheme for a basic matrix problem that could be of interest on its own. For us, it appears as an abstraction of a subproblem we need to tackle in order to solve the SIGNIFICANTPAGER-ANKS problem, but we hope that this abstraction will be useful in designing fast algorithms for identifying nodes that are significant beyond PageRank measurements.

In the abstract matrix problem it is assumed that one can access an unknown *right-stochastic matrix* by querying its rows, where the cost of a query and the accuracy of the answers depend on a precision parameter ϵ. At a cost propositional to $1/\epsilon$, the query will return a list of $O(1/\epsilon)$ entries and their indices that provide an ϵ-precision approximation of the row. Our task is to find a set that contains all columns whose sum is at least Δ, and omits any column whose sum is less than Δ/c. Our multi-scale sampling scheme solves this problem with cost $\tilde{O}(n/\Delta)$, while traditional sampling algorithms would take time $\Theta((n/\Delta)^2)$.

Our second main technical contribution is a new local algorithm for approximating personalized PageRank, which is more robust than the earlier ones developed in [2, 11] and is highly efficient particularly for networks with large in-degrees or out-degrees.

Together with our multiscale sampling scheme we are able to optimally solve the SIGNIFI-CANTPAGERANKS problem.

1 Introduction

A basic problem in network analysis is to identify the set of network nodes that are "significant."
For example, they could be the significant Web pages that provide the authoritative contents 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.

Identifying Nodes with Significant PageRanks The meanings and measures 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 *PageRank* of the vertices.

Formally, the PageRank (with restart constant, also known as the teleportation constant, α)
of a Web page is proportional to 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 Web page linked to from the current page, and with probability α, restart the process
from a uniformly chosen Web page. For the ease of presentation of our later results, we consider a
normalization of PageRank so that the sum of the PageRank values over all vertices is equal to n,
the number of vertices in the network,

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

PageRank has been used by the Google search engine and has found applications in a wide
range of data analysis problems [5,6]. In this paper, we consider the following natural problem of
finding vertices with "significant" PageRank:

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

For the corresponding algorithmic problem we assume that the network topology is described in
the *sparse* representation of an (arbitrarily ordered) adjacency list for each vertex, as is natural for
sparse graphs such as social and information networks. We are interested in developing an efficient
local algorithm [1,2,18] for the problem in the context of Web applications. The algorithm is only
allowed to randomly sample out-links of previously accessed nodes in addition to sampling nodes
uniformly at random from the network. This model is highly suitable for PageRank maintenance
in Web graphs and online information networks.

As the main contribution of this paper, we present a nearly optimal, local algorithm for SIG-
NIFICANTPAGERANKS. The running time of our algorithm is $\tilde{O}(n/\Delta)$. We also show that any
algorithm for SIGNIFICANTPAGERANKS must have query complexity as well as runtime complexity
$\Omega(n/\Delta)$. Thus, our algorithm is optimal up to a logarithmic factor. Note that when $\Delta = \Omega(n^\delta)$,
for some constant $0 < \delta < 0$, our algorithm has sublinear time complexity.

Our SIGNIFICANTPAGERANKS algorithm applies a multiscale matrix sampling scheme that uses
a fast Personalized PageRank estimator (see below) as its main subroutine.

Personalized PageRanks While the PageRank of a vertex captures the importance of the vertex as collectively assigned by all vertices in the network, one can use the distributions of the following random walks to define the pairwise contributions of significance [10]: 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 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 [15] as well as for developing good network clustering techniques [2]. As a result, fast algorithms for computing or approximating personalized PageRank are quite useful. One can approximate PageRanks and personalized PageRanks by the power method [5], which involves costly matrix-vector multiplications for large scale networks. Applying effective truncation, Jeh and Widom [11] 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.

Multi-Scale Matrix Sampling Following the matrix view of the personalized PageRank formulation of Haveliwala [10] and the subsequent approximation of algorithms [2, 11], we introduce a matrix problem whose solution would lead to fast PageRank approximation and sublinear-time algorithms for SIGNIFICANTPAGERANKS.

In the basic form of this matrix problem, we consider a blackbox model for accessing an unknown $n \times n$ *right-stochastic matrix*, in which we can only make a query of the following form: $matrixAccess(i, \epsilon)$, where $1 \leq i \leq n$ and $\epsilon \in (0, 1]$. This query will return, with high probability, a list of $O(1/\epsilon)$ entry-index pairs that provide an ϵ-precise approximation of row i in the unknown matrix: For each $1 \leq j \leq n$, if (p, j) is in the list of entry-index pairs returned by $matrixAccess(i, \epsilon)$, then $|p - m_{i,j}| \leq \epsilon$, where $m_{i,j}$ is the $(i, j)^{th}$ entry of the unknown matrix; otherwise if there is no entry containing index j, then $m_{i,j}$ is guaranteed to be at most ϵ. Further, the cost of this query is propositional to $1/\epsilon$. We will refer to this blackbox model as the *sparse and approximate row access model*, or SARA model for short.

We now define the basic form of our matrix problem:

SIGNIFICANTMATRIXCOLUMNS: *Given an $n \times n$ right-stochastic matrix M in the SARA model, a threshold Δ and a positive constant $c > 1$, return a subset of columns $S \subseteq V$ with the property that S contains all columns of sum at least Δ and no column with sum less than Δ/c.*

There is a straightforward connection between SIGNIFICANTMATRIXCOLUMNS and SIGNIFI-CANTPAGERANKS. Following [1], we define a matrix PPR (short for PersonalizedPageRank) to be the $n \times n$ matrix, whose u^{th} row is

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

Clearly PPR is a right-stochastic matrix and for $1 \leq v \leq n$, PageRank(v) is equal to the sum of the vth column in PPR. Therefore, if we can solve the SIGNIFICANTMATRIXCOLUMNS problem with cost $\tilde{O}(n/\Delta)$ and also solve the problem of computing an ϵ-additive approximation of personalized PageRank in $\tilde{O}(\log(n)/\epsilon)$ time, then we are able to solve SIGNIFICANTPAGERANKS in $\tilde{O}(n/\Delta)$ time.

In this paper, we analyze a multi-scale sampling algorithm for SIGNIFICANTMATRIXCOLUMNS. The algorithm selects a set of precision parameters $\{\epsilon_1, ..., \epsilon_h\}$ where h grows linearly with n/Δ and $\epsilon_i = i/h$. It then makes use of the sparse-and-approximate-row-access queries to obtain approximations of randomly sampled rows. For each i in range $1 \leq i \leq h$, the algorithm makes $\tilde{O}(1)$ (depending on the desired success probability) row-access queries to get a good approximation to the contribution of column elements of value of order ϵ_i. We show that with probability $1 - o(1)$, the multi-scale sampling scheme solve SIGNIFICANTMATRIXCOLUMNS with cost $\tilde{O}(n/\Delta)$.

While we could present our algorithm directly on PPR, we hope this matrix abstraction enables us to better highlight the two key algorithmic components in our fast PageRank approximation algorithm:

- multi-scale sampling, and

- robust approximation of personalized PageRanks.

Robust Approximation of PersonalizedPageRanks For networks with constant maximum degrees, we can simply use personalized PageRank approximation algorithms developed by Jeh-Widom [11] or Andersen-Chung-Lang [2] inside the multi-scale scheme to obtain an $\tilde{O}(n/\Delta)$ time algorithm for SIGNIFICANTPAGERANKS. However, for networks such as Web graphs and social networks that may have nodes with large degrees, these two approaches are not sufficient for our needs.

We develop a new local algorithm for approximating personzied PageRank that satisfies the desirable robustness property that our multiscale sample scheme requires. Given $\lambda, \epsilon > 0$ and a starting vertex u in a network $G = (V, E)$, our algorithm estimates each entry in the Pprsonalized PageRank vector defined by u,

$$\text{PersonalizedPageRank}(u, .)$$

to a $[1 - \lambda, 1 + \lambda]$ multiplicative approximation around its value plus an additive error of at most ϵ. The time complexity of our algorithm is $O\left(\frac{\log^2 n \log(\epsilon^{-1})}{\epsilon \lambda^2}\right)$. Our algorithm requires a careful simulation of random walks from the starting node u to ensure that its complexity does not depend on the degree of any node. Together with the multi-scale sampling scheme, this algorithm leads to an $\tilde{O}(n/\Delta)$ time algorithm for SIGNIFICANTPAGERANKS. We conclude our analysis by showing that our algorithm for solving SIGNIFICANTPAGERANKS is optimal up to a polylogarithmic factor.

Discussion 1. *While the main contribution of this paper is theoretical, that is, our focus is to design the first nearly optimal, local algorithm for PageRank approximation, we hope our algorithm or its refinements can be useful in practical settings for analyzing large-scale networks. For example, our sublinear algorithm for SIGNIFICANTPAGERANKS 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 insert them to the core to replace existing core pages with relatively low PageRank values. As noted already, our algorithms are local and are implementable in various*

network querying models that assume no direct global access to the network but allow one to generate random out-links of a given node as well as to uniformly at random sample nodes from the network. Such an implementation is desirable for processing large social and information networks as in the construction of the core pages for Web search. We also anticipate that our algorithm for SIGNIFICANTPAGERANKS *and the multi-scale scheme for its matrix abstraction will be useful for many other network analysis tasks.*

Related Work Our research is inspired by the body of work on local algorithms [1, 2, 18], sublinear-time algorithms [17], and property testing [9] which study algorithm design for finding relevant substructures or estimating various quantities of interest without examining the entire input. Particularly, we focus on identifying nodes with significant PageRanks and approximating personalized PageRanks without exploring the entire input network. In addition, our framework is based on a combination of uniform crawling and uniform sampling of vertices in a graph and hence it can be viewed as a sublinear algorithm (when $\Delta = n^{\Omega(1)}$) in a rather general access model as discussed in [17].

It is well-known that in a directed graph, high in-degree of a node does not imply high PageRank for that node and vice versa. In fact, even in real-world Web graphs, only weak correlations have been reported between PageRank and in-degree [16]. One therefore needs to use methods for PageRank estimation that are not solely based on finding high in-degree nodes. Indeed, over the past decade, various beautiful methods have been developed to approximate the PageRank of all nodes. The common thread is that they all run in time at least linear in the input (See [5] for a survey of results). Perhaps the closest ones to our framework are the following two Monte-Carlo based approaches. The PageRank estimation method of [3] conducts simulation of a constant number of random walks from each of the nodes in the network and therefore it requires linear time in the size of the network. A similar approach is analyzed in [4], where a small number of random walks are computed from each network node, which shows that a tight estimate for the PageRank of a node with a large enough PageRank can be computed from the summary statistics of these walks. In addition, the paper shows how these estimates can be kept up to date, with a logarithmic factor overhead, on a certain type of a dynamic graph in which a fixed set of edges is inserted in a random order.

Our scheme is suitable to handle any network with arbitrary changes in it as well, including addition or removal of edges and nodes, with the necessary computation being performed "on the spot" as needed. But in contrast to the above approaches, for $\Delta = n^{\Omega(1)}$, our construction gives a sublinear-time algorithm for identifying all nodes whose PageRanks are above threshold Δ and approximating their PageRanks.

We have benefited from the intuition of several previous works on personalized PageRank approximation. Jeh and Widom developed a method based on a deterministic simulation of random walks by pushing out units of mass across nodes [11]. Their algorithm gives an ϵ-additive approximation with runtime cost of order of $\log n/\epsilon$ times the maximum out-degree of a node in the network. Andersen, Chung, and Lang [2] provided a clever implementation of the approach of Jeh and Widom that removes the $\log n$ factor from the runtime cost, still stopping when the residual amount to push out per node is at most[1] ϵ. We note, however, that for networks with large

---

[1]Thus at termination the infinity norm of the residual vector is at most ϵ, which can easily be shown to bound from above the infinity norm of the difference between the true personalized PageRank vector and the estimation computed.

out-degrees, the complexity of this algorithm may not be sublinear.

Andersen *et al.* [1] developed a "backwards-running" version of the local algorithm of [2]. Their algorithm finds an ϵ-additive approximation to the PageRank vector with runtime proportional to $\frac{1}{\epsilon}$, times the maximum in-degree in the network, times the PageRank value. The authors show how it can be used to provide some reliable estimate to a node's PageRank: for a given k, with runtime proportional to $\tilde{\Theta}(k)$ times the maximum in-degree in the network (and no dependency on the PageRank value), it can bound the total contribution from the k highest contributors to a given node's PageRank. However, for networks with large in-degrees, its complexity may not be sublinear even for small values of k. We also note that the method does not scale well for estimating the PageRank values of multiple nodes, and needs to be run separately for each target node.

The problem of SIGNIFICANTMATRIXCOLUMNS can also be viewed as a matrix sparsification or matrix approximation problem, where the objective is to remove all columns with l_1 norm less than Δ/c while keep all columns with l_1 norm at least Δ. To achieve time-efficiency, it is essential to allow the algorithm the freedom in deciding whether to keep or delete columns whose l_1 norm is in the range $[\Delta/c, \Delta]$.

While there has been a large body of work of finding a low complexity approximation to a matrix (such as a low-rank matrix) that preserves some desirable properties, many of the techniques developed are not directly applicable to our task.

First, we would like our algorithms to work even if the graph does not have a good low rank approximation; indeed, all of our algorithms work for any input graph. Second, our requirement to approximately preserve l_1 norm only for significant columns enable us to achieve $\tilde{O}(n/\Delta)$ complexity for any stochastic matrix, whereas all low-rank matrix approximations run in time at least linear in the number of rows and columns of the matrix in order explicitly reconstruct a low-rank approximation; see [12, 13] for recent surveys on low-rank approximations.

On a high level, the problem of SIGNIFICANTMATRIXCOLUMNS may seem to share some resemblance to the *heavy-hitters* problem considered in the data streaming literature [7]. In the heavy-hitter problems, the goal is to identify all elements in a vector stream that have value bigger than the sum of all elements. The main difficulty to overcome is the sequential order by which items arrive and the small space one can use to store information about them. The main technique used to overcome these difficulties is the use of multiple hash functions which allows for concise summary of the frequent items in the stream. However, in SIGNIFICANTMATRIXCOLUMNS we are faced with a completely different type of constraints — access to only a small fraction of the input matrix (in order to achieve sublinear runtime) and having a precision-dependent cost of matrix row-approximations. As a result, hashing does not seem to be a useful avenue for this goals and one needs to develop different techniques in order to solve the problem.

Organization In Section 2, we introduce some notations that will be used in this paper. In Section 3, to better illustrate the multi-scale framework, we present a solution to a somewhat simpler abstract problem that distills the computational task we use to solve SIGNIFICANTMATRIX-COLUMNS. In particular, we consider a blackbox model accessing an unknown vector that either returns an exact answer or 0 otherwise. Like the access model in SIGNIFICANTMATRIXCOLUMNS, higher precision costs more. In Section 4, we present our multi-scale sampling algorithm for SIG-NIFICANTMATRIXCOLUMNS. In Section 5, we address the problem of finding significant columns in a PageRank matrix by giving a robust local algorithm for approximating personalized PageRank vectors. The section ends with a presentation of a tight lower bound for the cost of solving

Significant Matrix Columns over PageRank matrices.

2 Preliminaries

In this section, we introduce some basic notations that we will frequently use in the paper. For a positive integer n, $[1:n]$ denotes the set of all integers j such that $1 \leq j \leq n$. If $M \in \mathbb{R}^{n \times n}$ is an $n \times n$ real matrix, for $v \in [1:n]$, we will use $M(v, \cdot)$ and $M(\cdot, v)$ to denote v^{th} row and the vth column of M, respectively. We denote the sum of the column v in M by $\text{ColumnSum}(M, v)$. When the context is clear we shall suppress M in this notation and denote it by $\text{ColumnSum}(v)$.

Most graphs considered in this paper are directed. For a given directed graph $G = (V, E)$, we usually assume $V = [1:n]$. We use an $n \times n$ matrix $A(G)$ to denote the adjacency matrix of G. In other words, $A(i, j) = 1$ if and only $(i, j) \in E$.

The PageRank vector of a graph G is the (unique) stationary point of the following equation [10, 15]:

$$\text{PageRank}(\cdot) = \alpha \cdot \mathbf{1}^n + (1 - \alpha)\text{PageRank}(\cdot) \cdot D^{-1}A(G),$$

where $\mathbf{1}^n$ is the n-place row vector of all 1's, $0 < \alpha < 1$ is a teleportation probability constant, and D is a diagonal matrix with the out-degree of v at entry (v, v).

Similarly, the personalized PageRank vector of u in the graph G is the (unique) stationary point of the following equation [10]:

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

where $\mathbf{1}_u$ is the indicator function of u.

Note that with the above definition of PageRank, the sum of the entries of the PageRank vector is normalized to n. This normalization is more natural in the context of personalized PageRank than the traditional normalization in which the sum of all PageRank entries is 1.

For any x, $\log(x)$ means $\log_2(x)$ and $\ln(x)$ denotes the natural logarithm of x.

3 Multi-Scale Approximation of Vector Sum

Before presenting our algorithms for SignificantMatrixColumns, we give a multi-scale algorithm for a much simpler problem that, we hope, captures the essence of the general algorithm.

We consider the following blackbox model for accessing an unknown vector $\mathbf{p} = (p_1, ..., p_n) \in [0, 1]^n$: We can only access the entries of $\mathbf{p}$ by making a query of the form $vectorAccess(i, \epsilon)$. If $p_i \geq \epsilon$, the query $vectorAccess(i, \epsilon)$ returns p_i, otherwise when $p_i < \epsilon$, $vectorAccess(i, \epsilon)$ returns 0. Furthermore, $vectorAccess(i, \epsilon)$ incurs a *cost* of $1/\epsilon$. In this subsection, we consider the following abstract problem:

> VectorSum**:** *Given a blackbox model* vectorAccess() *for accessing an unknown vector* $\mathbf{p} = (p_1, ..., p_n) \in [0, 1]^n$, *a threshold* $\Delta \in [1:n]$ *and a positive constant* $c > 1$, *return* **PASS** *if* $\sum_i p_i \geq \Delta$, *return* **FAIL** *if* $\sum_i p_i < \frac{\Delta}{c}$, *and otherwise return either* **FAIL** *or* **PASS**.

To motivate our approach, before describing our multi-scale algorithm to solve this problem, let us first analyze the running time of a standard sampling algorithm. In such an algorithm, one

would take h i.i.d. samples $s_1, \ldots, s_h$ uniformly from $[1 : n]$ and query p_{s_t} at some precision ϵ to obtain an estimator

$$\frac{n}{h} \sum_{t=1}^{h} p_{s_t} \mathbf{I}[p_{s_t} \geq \epsilon]$$

for the sum $\sum_i p_i$. The error stemming from querying at precision ϵ would be of order $n\epsilon$, so we clearly will have to choose ϵ of order Δ/n or smaller not to drown our estimate in the query error, leading to a run time of order hn/Δ. The number of samples, h, on the other hand, has to be large enough to guarantee concentration, which at a minimum requires that the expectation of the sum $\sum_{t=1}^{h} p_{s_t} \mathbf{I}[p_{s_t} \geq \epsilon]$ is of order at least unity. But the expectation of this sum is upper bounded by $(h/n) \sum p_i$ which is of order $h\Delta/n$ in the most interesting case where $\sum p_i$ is roughly equal to Δ. We thus need h to be of order at least n/Δ, giving a running time of order $(n/\Delta)^2$, while we are aiming for a sublinear running time of order $\tilde{O}(n/\Delta)$.

Our algorithm is based on a different idea by querying p_t at a different precision each time, namely, by querying p_{s_t} at precision $\epsilon_t = t/h$ in the tth draw, and considering the estimator

$$\frac{n}{h} \sum_{t=1}^{h} \mathbf{I}[p_{s_t} \geq \epsilon_t] \tag{1}$$

for the sum $\sum_i p_i$. In expectation, this estimator is equal to n times

$$\frac{1}{h} \sum_{t=1}^{h} \mathbf{P}[p_{s_t} \geq \epsilon_t] = \frac{1}{h} \sum_{t=1}^{h} \mathbf{P}\left[p_{s_t} \geq \frac{t}{h}\right] \tag{2}$$

with s_t denoting an integer chosen uniformly at random from $[1 : n]$. This is a Riemann sum approximation to the well known expression

$$\mathbf{E}[p_s] = \int_0^1 dx \mathbf{P}[p_s \geq x]$$

and differs from this integral by an error $O(\frac{1}{h})$. In the most interesting case where $\sum_i p_i$ is of order Δ, concentration again requires h to be of order at least n/Δ, which also guarantees that the error $O(1/h)$ from the Riemann sum approximation does not dominate the expectation $\mathbf{E}[p_s] = \frac{1}{n} \sum_i p_i$. But now we only query p_s at the highest resolution $\epsilon_1 = 1/h$ once, leading to a much faster running time. In fact, up to log factors, the running time will be dominated by the first few queries, giving a running time of $\tilde{O}(h) = \tilde{O}(n/\Delta)$, as desired.

In the next section we proceed with the algorithm's formal description and analysis.

3.1 A Multi-Scale Algorithm for Approximating Vector Sum

The following algorithm, *MultiScaleVectorSum*, replaces the standard sampling to estimate the sum $\sum_i p_i$ by a multi-scale version which spends only a small amount of time at the computation intensive scales requiring high precision. In addition to the blackbox oracle *vectorAccess()*, this algorithm takes three other parameters: $\Delta \in (1, n)$ and $c > 1$ as defined in VECTORSUM, and a confidence parameter $\delta \in (0, 1)$: This algorithm uses randomization and we will show that it correctly solves VECTORSUM with probability at least $1 - \delta$. Our algorithm implements the strategy discussed above except for one modification: instead of sampling at a different precision ϵ_t each

time, we sample at each precision a constant number of times τ, where τ depends on the desired success probability, given a total number of queries equal to $L = \tau h$, where $h = \Theta(n/\Delta)$ with the implicit constant in the Θ-symbol depending on c in such a way that it grows with $(c-1)^{-2}$ as $c \to 1$ (somewhat arbitrary, but convenient for our notation and proofs, we introduce the c dependence of our constructions through the variable $\beta = \frac{c-1}{4c}$; in terms of this variable, we write the lower cutoff Δ/c as $\Delta(1 - 4\beta)$, and use the midpoint $\Delta(1 - 2\beta)$ between Δ and Δ/c as the cutoff for the algorithm to decide between **PASS** and **FAIL**).

Algorithm 1 MultiScaleVectorSum

Input: $vectorAccess(\cdot, \cdot)$, threshold $\Delta \in (1, n)$, cutoff parameter $c > 1$, failure probability $\delta \in (0, 1)$.

1: $\beta = \frac{c-1}{4c}$; $\tau = \lceil \log(1/\delta) \rceil$; $h = \lceil \frac{3n}{\Delta \beta^2} \rceil$; $L = \tau h$
2: $sum = 0$.
3: **for** $t = 1 : L$ **do**
4: $\quad \epsilon_t = \frac{1}{h} \lceil \frac{t}{\tau} \rceil$.
5: $\quad$ Let s_t be an uniform random element from $[1 : n]$.
6: $\quad z_t = vectorAccess(s_t, \epsilon_t)$.
7: $\quad sum = sum + z_t$.
8: **end for**
9: **if** $sum \geq (1 - 2\beta) \frac{L\Delta}{n}$ **then**
10: $\quad$ Return **PASS**.
11: **else**
12: $\quad$ return **FAIL**.
13: **end if**

Theorem 3.1 (Multi-Scale Vector Sum). *For any* $\mathbf{p} \in (0, 1)^n$ *accessible by* $vectorAccess()$, *threshold* $\Delta \in (1, n)$, *robust parameter* $c > 1$, *and failure parameter* $\delta \in (0, 1)$, *the method* MultiScaleVectorSum $(vectorAccess(), \Delta, c, \delta)$ *correctly solves* VECTORSUM *with probability at least* $(1 - \delta)$ *and costs*

$$O\left(\frac{n}{\Delta} \left(\frac{1}{c - 1} \right)^2 \log\left(\frac{n}{\Delta(c - 1)} \right) \log\left(\frac{2}{\delta} \right) \right).$$

Proof. By Steps 3-7, for any constant $c > 1$, the cost of the algorithm is

$$\sum_{t=1}^{L} \frac{1}{\epsilon_t} = \tau \sum_{i=1}^{h} \frac{h}{i} \leq L(1 + \log h) = O\left(\frac{n}{\Delta} \left(\frac{1}{c - 1} \right)^2 \log\left(\frac{n}{\Delta(c - 1)} \right) \log\left(\frac{2}{\delta} \right) \right).$$

We now prove the correctness of the algorithm.

Algorithm MultiScaleVectorSum, after the initialization Steps 1 and 2, computes the multi-scale parameters ϵ_t and applies sampling to calculate the sum

$$Q = \sum_{t=1}^{L} z_t = \sum_{t=1}^{L} \mathbf{I}\left[p_{s_t} \geq \epsilon_t \right]$$

8

where $s_1, \ldots, s_L$ are chosen i.i.d. uniformly at random from $[1 : n]$. The expectation of Q is easily estimated in terms of the bounds

$$\mathbf{E}[Q] = \frac{1}{n} \sum_{k=1}^{n} \sum_{t=1}^{L} \mathbf{I}\left[\epsilon_t \le p_k\right] = \frac{1}{n} \sum_{k=1}^{n} \sum_{t=1}^{L} \mathbf{I}\left[\lceil t/\tau \rceil \le hp_k\right]$$

$$= \frac{\tau}{n} \sum_{k=1}^{n} \sum_{i=1}^{h} \mathbf{I}\left[i \le hp_k\right] = \frac{\tau}{n} \sum_{k=1}^{n} \lfloor hp_k \rfloor \le \frac{\tau}{n} \sum_{k=1}^{n} hp_k = \frac{L}{n} \sum_{k=1}^{n} p_k \tag{3}$$

and

$$\mathbf{E}[Q] \ge \frac{\tau}{n} \sum_{k=1}^{n} \left(hp_k - 1\right) = \frac{L}{n} \sum_{k=1}^{n} p_k - \tau. \tag{4}$$

We thus use $\frac{n}{L}Q$ as an estimate of $\sum_{k=1}^{n} p_k$ when we decide on whether to output **PASS** in Step 9.

Assume first that $\sum p_k \ge \Delta$. Since $\tau \le \beta^2 \frac{L\Delta}{3n} \le \beta \frac{L\Delta}{n}$, we then have

$$\mathbf{E}[Q] \ge \frac{L\Delta}{n} - \tau \ge (1 - \beta)\frac{L\Delta}{n},$$

implying that

$$(1 - \beta)\mathbf{E}[Q] \ge (1 - 2\beta)\frac{L\Delta}{n}.$$

This allows us to use the multiplicative Chernoff bound in the form of Lemma A.1 to conclude that

$$\mathbf{Pr}\left[Q \le (1 - 2\beta)\frac{L\Delta}{n}\right] \le \mathbf{Pr}\left[Q \le (1 - \beta)\mathbf{E}[Q]\right] \le \exp\left(-\frac{\beta^2}{2}\mathbf{E}[Q]\right) \le \exp\left(-\frac{3}{8}\frac{\beta^2 L\Delta}{n}\right) \le \delta,$$

where we used $\beta \le 1/4$ in the last step.

On the other hand, if $\sum p_k \le \Delta/c = (1 - 4\beta)\Delta$, we bound

$$\mathbf{E}[Q] \le \frac{L\Delta}{n}(1 - 4\beta)$$

which in turn implies that

$$(1 + 2\beta)\mathbf{E}(Q) \le (1 - 2\beta)\frac{L\Delta}{n}.$$

Using the multiplicative Chernoff bound in the form A.1 (part 3), this gives

$$Pr\left[Q \ge (1 - 2\beta)\frac{L\Delta}{n}\right] \le \exp\left(-\beta^2\frac{L\Delta}{n}\frac{1 - 2\beta}{1 + 2\beta}\right) \le \exp\left(-\frac{\beta^2 L\Delta}{3n}\right) \le \delta.$$

where we again used $\beta \le 1/4$.

Thus, MultiScaleVectorSum $(vectorAccess(), \Delta, c, \delta)$ correctly solves VECTORSUM with probability at least $1 - \delta$.

$\square$

4 Multi-Scale Matrix Sampling

In this section, we consider SIGNIFICANTMATRIXCOLUMNS in a slightly more general matrix access model than what we defined in Sections 1 and 3. The extension of the model is also needed in our PageRank approximation algorithm, which we will present in the next section.

4.1 Notation: Sparse Vectors

To better specify this model and the subsequent algorithms, we first introduce the notation of *sparse vector* introduced by Gilbert, Moler, and Schreiber [8] for Matlab. Suppose $\mathbf{a} = (a_1, ..., a_n) \in \mathbb{R}^n$ is a vector. Let nnz($\mathbf{a}$) denotes the number of nonzero elements in $\mathbf{a}$. Let Sparse($\mathbf{a}$) denote the *sparse form* of vector $\mathbf{a}$ by "squeezing out" any zero elements in $\mathbf{a}$. Conceptually, one can view Sparse($\mathbf{a}$) as a list of nnz($\mathbf{a}$) index-entry pairs, one for each nonzero element and its index in $\mathbf{a}$. For example, we can view Sparse($[0, 0.3, 0.5, 0, 0.2]$) as $((2, 0.3), (3, 0.5), (5, 0.2))$.

A sparse vector can be easily implemented using a binary search tree [2]. Throughout the paper we shall make use of the following simple proposition:

Proposition 4.1. *For $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$, $\mathbf{a} + \mathbf{b}$ can be implemented in time $O(\text{nnz}(\mathbf{b}) \cdot \log n)$ saving the result in the data structure of $\mathbf{a}$.*

Proof. Each sparse vector can be implemented as a balanced binary search tree, where the index of an entry serves as the entry's key. When performing the addition, we update the binary search tree of $\mathbf{a}$ by inserting one by one the elements of $\mathbf{b}$ into it (and updating existing entries whenever needed). By the standard theory of binary search trees, each such insertion operation takes $O(\log n)$ time. $\qquad\square$

In the rest of the paper, without further elaboration, we assume all vectors are expressed in this sparse form. We also adpot the following notations: let Sparse($[\,]$) denote the all zero's vector in the sparse form, and for any $i \in [1 : n]$ and $b \in \mathbb{R} - \{0\}$, let Sparse(i, b) denote the sparse vector with only one nonzero element b located in the ith place in the vector. In addition, we will use the following notation: For two vectors n-place vectors $\mathbf{a} = (a_1, ..., a_n)$ and $\mathbf{b} = (b_1, ..., b_n)$, and parameters $\epsilon \in \mathbb{R}$ and $C > 0$, we use $\mathbf{a} \leq C \cdot \mathbf{b} + \epsilon$ to denote $a_i \leq Cb_i + \epsilon$, $\forall i \in [1 : n]$.

4.2 The Matrix Access Model

In the model that we will consider in the rest of this section, we can access an unknown $n \times n$ *right-stochastic matrix* $M = (m_{i,j})$ using queries of the form $matrixAccess(i, \epsilon, \lambda, p)$, where $i \in [1 : n]$ specifies a row, $\epsilon \in (0, 1]$ specifies a required additive precision, $\lambda \in (0, 1]$ specifies a multiplicative precision, and $p \in (0, 1]$ specifies the probability requirement. This query will return a sparse vector $\tilde{\mathbf{m}}_i = \text{Sparse}([\tilde{m}_{i,1}, ..., \tilde{m}_{i,n}])$ such that

- with probability at least $1 - p$,

$$(1 - \lambda) \cdot \mathbf{m}_i - \epsilon \leq \tilde{\mathbf{m}}_i \leq (1 + \lambda) \cdot \mathbf{m}_i + \epsilon, \tag{5}$$

where $\mathbf{m}_i = M(i, \cdot)$ denotes the ith row of matrix M, and

---

[2]For average case rather than worst-case guarantees, a hash table is a typical implementation choice.

- with probability at most p (the query may fail), $\tilde{\mathbf{m}}_i$ could be an arbitrary sparse vector.

We refer to this blackbox model as the *probabilistic sparse-and-approximate row-access model with additive/multiplicative errors*. For constant integers $c_1, c_2, c_3, c_4 > 0$, we say that *matrixAccess* is an (c_1, c_2, c_3, c_4)-SARA model if for all $i \in [1 : n]$, $\epsilon \in (0, 1)$, $\lambda \in (0, 1)$, and $p \in (0, 1)$, both the cost of calling $\tilde{\mathbf{m}}_i = matrixAccess(i, \epsilon, \lambda, p)$ and $\mathrm{nnz}(\tilde{\mathbf{m}}_i)$ are bounded from above by

$$c_1 \left(\frac{1}{\lambda} \right)^{c_2} \left(\frac{\log^{c_3}(1/\epsilon)}{\epsilon} \right) (\log^{c_4} n) \log(1/p).$$

4.3 The Matrix Problem

In this section, we give a solution to the following abstract problem.

> SIGNIFICANTMATRIXCOLUMNS: *Given an $n \times n$ right-stochastic matrix M in the (c_1, c_2, c_3, c_4)-SARA model, a threshold Δ and a positive constant $c > 1$, return a sparse vector cSum with the property that for all $j \in [1 : n]$, if $\mathrm{ColumnSum}(M, i) \geq \Delta$, then $cSum(j) \neq 0$ and if $\mathrm{ColumnSum}(M, i) < \Delta/c$, then $cSum(j) = 0$.*

4.4 Understanding the Impact of Additive/Multiplicative Errors

Our algorithm for SIGNIFICANTMATRIXCOLUMNS is straightforward. At a high level, it simultaneously applies Algorithm 1 to all columns of the unknown matrix. It uses a sparse-vector representation for efficient bookkeeping of the columns with large sum according to the sampled data. Our analysis of this algorithm is similar to the one presented that in Theorem 3.1 for VECTORSUM as we can use the union bound over the columns to reduce the analysis to a single column. The only technical difference is the handling of the additive/multiplicate errors.

To understand the impact of these errors, we consider a vector $\mathbf{p} = (p_1, ..., p_n) \in [0, 1]^n$ and chose ϵ_t, $t = 1, \ldots, L$ as in Algorithm 1. Fix $\phi, \lambda \in (0, 1/2)$, and suppose that we access p_i with multiplicative error λ and additive error $\phi \cdot \epsilon_t$. We will show that if this returns a number $\tilde{p}_i \geq \epsilon_t$, the actual value of p_i is at least $\rho \epsilon_t$, where $\rho = 1 - \lambda - \phi$. To see this, we bound

$$p_i \geq (1 + \lambda)^{-1}(\tilde{p}_i - \phi \cdot \epsilon_t) \geq (1 + \lambda)^{-1}(1 - \phi)\epsilon_t.$$

Since $(1 - \phi)/(1 + \lambda) \geq (1 - \lambda - \phi)$, this implies $p_i \geq \rho \epsilon_t$, as desired.

In a similar way, it is easy to see that $p_i \geq \rho^{-1} \epsilon_t$ implies that $\tilde{p}_i \geq \epsilon_t$. Indeed, if $p_i \geq \rho^{-1} \epsilon_t$ then

$$\tilde{p}_i \geq (1 - \lambda)p_i - \phi \cdot \epsilon_t \geq \left(\frac{1 - \lambda}{1 - \lambda - \phi} - \phi \right) \epsilon_t.$$

The lower bound is clearly larger than ϵ_t, , showing that $\tilde{p}_i \geq \epsilon_t$.

For $s_1 \ldots, s_L \in [1 : n]$, the sum

$$\tilde{Q} = \sum_{t=1}^{L} \mathbf{I}[\tilde{p}_{s_t} \geq \epsilon_t] \tag{6}$$

can therefore be bounded from below and above by

$$Q_- = \sum_{t=1}^{L} \mathbf{I}[p_{s_t} \geq \rho^{-1} \epsilon_t] \quad \text{and} \quad Q_+ = \sum_{t=1}^{L} \mathbf{I}[p_{s_t} \geq \rho \epsilon_t], \tag{7}$$

11

respectively:
$$Q_- \leq \tilde{Q} \leq Q_+. \tag{8}$$
Finally, we also note that if we access p_i with multiplicative error λ and additive error $\phi \cdot \epsilon_t$, then this a returns a number which is never larger than ρ^{-1}. Indeed, this follows by bounding $\tilde{p}_i$ by $1 + \lambda + \phi \cdot \epsilon_t \leq 1 + \lambda + \phi \leq \rho^{-1}$.

4.5   A Multi-Scale Algorithm

In this section we present the multi-scale algorithm in full details and proceed with an analysis of its runtime and correctness. The algorithm is essentially an extension of Algorithm 1, applying the VectorSum algorithm to all columns in parallel. As now the call to vectorAcesss has been replaced by a combined additive-multiplicative method, the constant β is set to a slightly smaller value than in Algorithm 1. In addition to the constants β, τ, h, L that are used in Algorithm 1, we also have the constant λ for the value of multiplicative approximation needed and ϕ for the additive-approximation needed. Lastly, p is the wanted success probability of the row approximation procedure (matrixAccess) invoked throughout the algorithm. We note that these constants are defined to allow complete and rigorous analysis of our algorithm and its correctness. As the multi-scale algorithm will essentially be implementing Algorithm 1 over all columns, we will need a method that can return all elements in a row that fall within a certain bin; we call it the *rangeIndicator* method.

rangeIndicator(): for a sparse vector $\mathbf{a}$, and $l, u \in \mathbb{R}$ such that $l < u$, $\mathbf{b} = \text{rangeIndicator}(\mathbf{a}, l, u)$ returns a sparse vector $\mathbf{b}$ such that for all $i \in [1 : n]$,

$$\mathbf{b}(i) = \begin{cases} 1 & \text{if } l \leq \mathbf{a}(i) \leq u \\ 0 & \text{otherwise} \end{cases}$$

For example, rangeIndicator(Sparse([0, 0.3, 0.5, 0, 0.2]), 0.1, 0.3) returns the sparse form of $[0, 1, 0, 0, 1]$. We shall use the following simple proposition:

Proposition 4.2. rangeIndicator$(\mathbf{a}, l, u)$ *takes* $O(\text{nnz}(\mathbf{a})\log n)$ *time.*

Proof. The sparse vector nnz$(\mathbf{a})$ is implemented using a binary search tree; one can therefore scan its content using, say, an inorder scan and insert each element in the range $[l, u]$ to a sparse vector $\mathbf{b}$, initially empty. The inorder scan costs $O(\text{nnz}(\mathbf{a}))$ time and each insertion into b costs $O(\log n)$ time, giving the desired result. $\square$

We are now ready to state our main theorem.

Theorem 4.3 (Multi-Scale Column Sum). *For any right-stochastic matrix M accessible by matrixAccess, threshold $\Delta \in (1, n)$, robust parameter $c > 1$, and failure parameter $\delta \in (0, 1)$, with probability at least $(1 - \delta)$,*
$$cSum = \text{MultiScaleVectorSum}\left(vectorAccess(), \Delta, c, \delta\right).$$
correctly solves SIGNIFICANTMATRIXCOLUMNS.

Furthermore, if matrixAccess is an (c_1, c_2, c_3, c_4)-SARA model, then the cost of MultiScaleVectorSum $(vectorAccess(), \Delta, c, \delta)$ *is*

$$O\left(c_1 \left(\frac{n}{\Delta} \right) \left(\frac{1}{c-1} \right)^{c_2+3} \log^{c_3+2} \left(\frac{1}{c-1} \right) \log^{c_3+c_4+3} n \log^2 \left(\frac{2}{\delta} \right) \right).$$

Algorithm 2 MultiScaleColumnSum

Input: $matrixAccess(\cdot,\cdot,\cdot,\cdot)$, threshold $\Delta \in (1,n)$, cutoff $c > 1$, failure probability $\delta \in (0,1)$.

1: $\beta = \frac{c-1}{5c}$; $\quad \tau = \lceil \log(2n/\delta) \rceil$; $\quad h = \lceil \frac{3n}{\Delta\beta^2} \rceil$; $\quad L = \tau h$; $\quad p = \delta/(2L)$; $\quad \lambda = \beta/2$; $\quad \phi = \beta/2$;
   $\rho = 1 - \lambda - \phi$.
2: $cSum = \text{Sparse}([\ ])$.
3: **for** $t = 1 : L$ **do**
4: $\quad \epsilon_t = \frac{1}{h}\lceil\frac{t}{\tau}\rceil$.
5: $\quad$ Let s_t be an uniform random element from $[1:n]$; $\quad q_t = matrixAccess(s_t, \phi \cdot \epsilon_t, \lambda, p)$.
6: $\quad \mathbf{z}_t = \text{rangeIndicator}\left(\mathbf{q}_t, \epsilon_t, \rho^{-1}\right)$.
7: $\quad cSum = cSum + \mathbf{z}_t$.
8: **end for**
9: $cSum = \text{rangeIndicator}\left(cSum, (1-2\beta)\frac{L\Delta}{n}, L\right)$;
10: Return $cSum$.

Proof. The cost of the algorithm is dominated by the sparse matrix operations in line 5-7, plus the cost of the last operation in line 9. Using our access model together with Propositions 4.1 and 4.2, the cost of the steps in line 5-7 at time t are of order

$$O\left(c_1\left(\frac{1}{\beta}\right)^{c_2}\frac{\log^{c_3}(\frac{1}{\beta\epsilon_t})}{\beta\epsilon_t}\log^{c_4+1}n\,\log(2L/\delta)\right) \le$$

$$O\left(c_1\frac{h}{\lceil t/\tau \rceil}\log^{c_3}h\left(\frac{1}{\beta}\right)^{c_2+1}\log^{c_4+1}n\log(\frac{2n}{\Delta\beta\delta})\right).$$

Note that this includes the extra factor of $\log n$ from Proposition 4.1, a factor which is absent in the sparseness of $\mathbf{q}_t$ and $\mathbf{z}_t$.

Summing over t gives a running time of order

$$O\left(c_1 L\log^{c_3+1}h\left(\frac{1}{\beta}\right)^{c_2+1}\log^{c_4+1}n\log(\frac{2n}{\Delta\beta\delta})\right)$$

$$= O\left(c_1\left(\frac{n}{\Delta}\right)\log^{c_3+1}\left(\frac{n}{\Delta\beta}\right)\left(\frac{1}{\beta}\right)^{c_2+3}\log^{c_4+1}n\log\left(\frac{2n}{\Delta\beta\delta}\right)\log\left(\frac{2}{\delta}\right)\right)$$

$$= O\left(c_1\left(\frac{n}{\Delta}\right)\left(\frac{1}{\beta}\right)^{c_2+3}\log^{c_3+2}\left(\frac{1}{\beta}\right)\log^{c_3+c_4+3}n\log^2\left(\frac{2}{\delta}\right)\right).$$

To estimate the cost of the last step of the algorithm, we bound the sparseness of $cSum$ at the completion of the FOR loop by $\text{nnz}(cSum) \le \sum_t \text{nnz}(\mathbf{z}_t)$ and then apply Proposition 4.2 once more, giving a cost which of the same order as the total cost of the algorithm accrued up to this step.

To prove the correctness of the algorithm, we first note that with probability at least $(1-p)^L \ge 1 - pL$, each of the L calls of $matrixAccess$ in line 5 will return a sparse vector obeying the bound (5). Next, we apply the union bound to reduce the focus of the analysis to a single column:

$$\mathbf{Pr}\left[\textsc{MultiScaleColumnSum is unsuccessful}\right] \le$$
$$\textstyle\sum_{i=1}^{n}\mathbf{Pr}\left[\textsc{MultiScaleColumnSum is unsuccessful on column } i\right].$$

When considering column i, we now let $\mathbf{p} = (p_1, ..., p_n)^T = M(\cdot, i)$, the ith column of M. In other words, $p_j = m_{i,j}$ for all $j \in [1:n]$. Note that the ith entry of $cSum$ after step 8 is of the form (6). Taking into account the bound (8), our proof will be very similar to that of Theorem 3.1.

We first consider the case that $\sum_i p_i \geq \Delta$ in which case we bound

$$\mathbf{E}[Q_-] \geq \frac{L\rho}{n} \sum_k p_k - \tau \geq \frac{\Delta L}{n} \left[\rho - \frac{\beta^2}{3} \right] = \frac{\Delta L}{n} \left[1 - \beta - \frac{\beta^2}{3} \right].$$

Multiplying both sides by $(1 - \beta)$, we obtain

$$(1 - \beta)\mathbf{E}[Q_-] \geq \frac{\Delta L}{n} (1 - 2\beta).$$

Combined with the bound (8) and the multiplicative Chernoff bound (Lemma A.1), this shows that conditioned on $matrixAccess$ returning a sparse vector obeying the bound (5) in each instance in line 5, we get

$$\mathbf{Pr}\left\{ cSum(i) \leq (1 - 2\beta)\frac{\Delta L}{n} \right\} \leq \exp\left(-\frac{\beta^2}{2}\mathbf{E}[Q_-] \right) \leq \exp\left(-\frac{3}{8}\frac{\beta^2 L\Delta}{n} \right) \leq \frac{\delta}{2n}.$$

In a similar way, if $\sum_k p_k \leq \Delta/c = \Delta(1 - 5\beta)$, we bound

$$\mathbf{E}[Q_+] \leq \frac{\Delta L}{n\rho}(1 - 5\beta) = \frac{\Delta L}{n}\frac{1 - 5\beta}{1 - \beta},$$

implying that

$$(1 + 2\beta)\mathbf{E}[Q_+] \leq \frac{\Delta L}{n}(1 - 2\beta)$$

and hence

$$\mathbf{Pr}\left\{ cSum(i) \geq (1 - 2\beta)\frac{\Delta L}{n} \right\} \leq \exp\left(-\beta^2\frac{\Delta L}{n}\frac{1 - 2\beta}{1 + 2\beta} \right) \leq \exp\left(-\frac{\Delta L\beta^2}{3n} \right) \leq \frac{\delta}{2n},$$

again conditioned on $matrixAccess$ returning a sparse vector obeying the bound (5) in each instance in line 5.

Thus the total failure probability is at most $Lp + n\frac{\delta}{2n} = \delta$, as desired. $\qquad\square$

5 Identifying Nodes with Significant PageRank

5.1 Robust Approximation of Personalized PageRanks

We now present our main subroutine for SIGNIFICANTPAGERANKS which, we recall, is addressing the following problem: Given a directed graph $G = (V, E)$, a threshold value $1 \leq \Delta \leq |V|$ and a positive constant $c > 1$, compute 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.

Let PPR denote the personalized PageRank Matrix of G defined in the Introduction, where we recall that $PPR(i, j)$ is equal to the personalized PageRank contribution of node i to node j in

G. Under this notation, the SIGNIFICANTPAGERANKS can be viewed as a SIGNIFICANTMATRIX-COLUMNS problem, if we can develop an efficient procedure for accessing the rows of PPR. This procedure, which we refer to as *PPRmatrixAccess()*, takes a row number i, an additive precision parameter ϵ, a multiplicative precision parameter λ and success probability p, and returns a sparse vector $\tilde{\mathbf{m}}_i = \text{Sparse}([\tilde{m}_{i,1}, ..., \tilde{m}_{i,n}])$ such that

- with probability at least $1 - p$,

$$(1 - \lambda) \cdot \mathbf{m}_i - \epsilon \leq \tilde{\mathbf{m}}_i \leq (1 + \lambda) \cdot \mathbf{m}_i + \epsilon,$$

  where $\mathbf{m}_i = PPR(i, \cdot)$, and

- with probability at most p, $\tilde{\mathbf{m}}_i$ can be any sparse vector.

Our algorith for *PPRmatrixAccess()* use the following key observation that connects personalized PageRank with the hitting probability of a Markov model.

Observation 5.1. *$PPR(v, j)$ is equal to the success probability that a random walk starting at v and independently terminating at each time step with probability α, hits j just before termination.*

Proof. Let 1_v be the indicator vector of v. Solving the system given by

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

one obtains

$$\text{PersonalizedPageRank}(v, \cdot) = \alpha \mathbf{1}_v (I - (1 - \alpha)D^{-1}A)^{-1} = \alpha \mathbf{1}_v \sum_{i=0}^{\infty} ((1 - \alpha)D^{-1}A)^i.$$

The observation then follows directly from the last equation. $\square$

Our algorithm for *PPRmatrixAccess* given below conducts a careful simulation of such restarting random walks. As such it only needs an oracle access to a random out-link of a given node.

Algorithm 3 PPRmatrixAccess

Input: node v, additive approximation ϵ, multiplicative approximation λ.

1: $cSum = \text{Sparse}([\ ])$.
2: Set $length = \lceil \log_{\frac{1}{(1-\alpha)}} (\frac{4}{\epsilon}) \rceil$.
3: Set $r = \lceil \frac{1}{\epsilon \lambda^2} \cdot 4 \ln(n/p) \rceil$.
4: **for** r rounds **do**
5:     Run one realization of a restarting random walk from v. Artificially stop the walk after *length* steps if it has not terminated already.
6:     **if** the walk visited a node j just before making a termination step **then**
7:         $cSum = cSum + Sparse(j, 1/r)$ //namely, add $1/r$ to j's value.
8:     **end if**
9:     Return $cSum$.
10: **end for**

Theorem 5.2. *For any node v, values $0 < \epsilon < 1$, $0 < \lambda < 1$, $0 < \alpha < 1$, and success probability $0 < p < 1$, PPRmatrixAccess$(v, \epsilon, \lambda, p)$ is a $(10 \max\{\log^{-1}(\frac{1}{1-\alpha}), 1\}, 2, 1, 2)$-SARA model. In particular, its runtime is upper bounded by*

$$O\left(\frac{\ln^2(n)\ln(1/p)\log(\epsilon^{-1})}{\epsilon\lambda^2}\right).$$

Proof. We start by analyzing the runtime guarantee. The algorithm performs $\lceil \frac{1}{\epsilon\lambda^2} \cdot 4\ln(n/p)\rceil$ 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 uniform sample ('termination' step) with probability α and by choosing a random out-link with probability $1 - \alpha$. The update of $cSum$ in line 7 takes at most $\log n$ (see proposition 4.1). Thus the total number of queries used is

$$\left\lceil\frac{4\ln(n/p)}{\epsilon\lambda^2}\right\rceil \cdot \left\lceil\log_{\frac{1}{(1-\alpha)}}\left(\frac{4}{\epsilon}\right)\right\rceil \log(n) \leq \left\lceil\frac{4\ln(n/p)}{\epsilon\lambda^2}\right\rceil \cdot \left\lceil\frac{\log(\frac{4}{\epsilon})}{\log(\frac{1}{1-\alpha})}\right\rceil \log(n) \leq$$

$$(8+2)\max\left\{\log^{-1}(\frac{1}{1-\alpha}), 1\right\}\frac{\ln^2(n)\log(1/p)\log(\epsilon^{-1})}{\epsilon\lambda^2}.$$

We now prove the guarantees on the returned vector $cSum$ (line 9 in the algorithm). Given a node j, denote by $p_k(v, j)$ the contribution to j from restarting walks originating at v that are of length at most k, namely,

$$p_k(v, j) = \alpha\mathbf{1}_v\sum_{i=0}^{k}(1-\alpha)D^{-1}A)^i.$$

We ask how much is contributed to j's entry from restarting 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:

$$PPR(v, j) - \frac{\epsilon}{4} \leq p_k(v, j) \leq PPR(v, j).$$

For this to hold it suffices to take $k = \lceil\log_{\frac{1}{(1-\alpha)}}\left(\frac{4}{\epsilon}\right)\rceil$, the value the parameter *length* is set to in step 2.

Next, the algorithm computes an estimate of $p_k(v, j)$ by realizing walks of length at most k. This is the value of $cSum$ at index j returned by the algorithm. Denote this by $\hat{p}_k(v, j)$. The algorithm computes such an estimation (in line 7) by taking the average number of hits over r trials (adding $1/r$ per hit).

Now, if $PPR(v, j) \geq \frac{\epsilon}{2}$ then $p_k(v, j) \geq \frac{\epsilon}{4}$ and by the multiplicative Chernoff bound (Lemma A.1),

$$Pr\left(\hat{p}_k(v, j) > (1+\lambda)p_k(v, j)\right) \leq \exp(-\ln(n/p))$$

and

$$Pr\left(\hat{p}_k(v, j) < (1-\lambda)p_k(v, j)\right) \leq \exp(-\ln(n/p)).$$

By the union bound we can conclude that with probability $1 - \frac{2p}{n}$,

$$(1-\lambda)(PPR(v, j) - \frac{\epsilon}{4}) \leq \hat{p}_k(v, j) \leq (1+\lambda)PPR(v, j).$$

16

Similarly, if $PPR(v,j) < \frac{\epsilon}{2}$ then $p_k(v,j) < \frac{\epsilon}{2}$ and by the multiplicative Chernoff bound (Lemma A.1, part 3),

$$Pr\left(\hat{p}_k(v,j) > (1+\lambda)\frac{\epsilon}{2}\right) \leq \exp(-\ln(n/p)) = p/n.$$

As $\lambda < 1$ we therefore have $0 \leq \hat{p}_k(v,j) \leq \epsilon$ with probability at least $1 - p/n$. And as $PPR(v,j) < \frac{\epsilon}{2}$ we clearly have, with probability $1 - p/n$,

$$(1-\lambda)PPR(v,j) - \epsilon \leq \hat{p}_k(v,j) \leq (1+\lambda)PPR(v,j) + \epsilon,$$

as needed.

By the union bound, the complete claim holds with probability at least $1 - p$. □

5.2 A Tight Lower Bound for Solving the SIGNIFICANTPAGERANKS Problem

In this subsection, we present a corresponding lower bound for identifying all nodes with significant PageRank values. Our lower bound holds under the stringent model where one can access any node of interest in the graph in one unit of cost and that the PageRank of the node accessed is given for free. We call such a model the *strong query model*. We first give a lower bound to illurstrate the challenge for identifying nodes with significant PageRanks, even in graphs where there is only one significant node. We then show that for any integral threshold Δ and precision c there are instances where the output size of SIGNIFICANTPAGERANKS is $\Omega(n/\Delta)$. Clearly, this also serves as a lower bound for the runtime of any algorithm that solves the SIGNIFICANTPAGERANKS problem, regardless of the computational model used to compute the required output. We note that the runtime of our algorithmic solution to SIGNIFICANTPAGERANKS is at most only a small polylogarithmic factor away from this bound.

For clarity of exposition we present our lower bounds for $\alpha = 0.5$. Similar lower bounds hold for any fixed $0 < \alpha < 1$.

Theorem 5.3 (Hardness for Identifying One Significant Node)**.** *Let $\alpha = 0.5$. For n large enough, any algorithm making less than $\frac{n}{6\Delta}$ queries in the strong query model on graphs on n nodes and threshold $\Delta \leq \frac{n}{9}$, would fail with probability at least $1/e$ to find a node with PageRank at least Δ, on at least one graph on n nodes.*

Proof. The proof will apply Yao's Minimax Principle for analyzing randomized algorithms [19], which uses the average-case complexity of the deterministic algorithms to derive a lower bound on the randomized algorithms for solving a problem.

Given positive integers n and $\Delta \leq \frac{n}{9}$, we construct a family $\mathcal{F}$ of undirected graphs on n nodes by taking a cycle subgraph on $n - d - 1$ nodes and an isolated star subgraph on the remaining $d+1$ nodes, where we set $d = 3\Delta - 1$. To complete the construction we take a random labeling of the nodes. See Figure 1 for an illustration.

Let A be a deterministic algorithm for the problem. We shall analyze the behavior of A on a uniformly random graph from $\mathcal{F}$.

First, by solving the PageRank equation system it is easy to check that each node on the cycle subgraph has PageRank value of 1, the hub of the subgraph has PageRank $\frac{d}{3} + \frac{2}{3}$, and a leaf of the star subgraph has PageRank $\frac{2}{3} + \frac{1}{3d}$. The only node with PageRank at least Δ is the hub of the star subgraph.
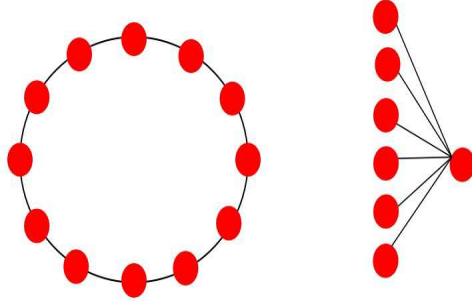
Figure 1: An example illustrating the "cycle & star" lower bound construction for PageRank computations.

Let T be the number of queries the algorithm make. The probability that none of the nodes of the star subgraph are found after T queries by A is at least

$$(1 - \frac{d+1}{n})^T \geq \exp(-2T\frac{d+1}{n}) \geq \exp(-1),$$

for $T \leq \frac{n}{6\Delta} = \frac{n}{2(d+1)}$. Here we used the fact that $1 - x \geq \exp(-2x)$, for $0 \leq x \leq 1/3$.

We define the cost of the algorithm as 0 if it has found a node of Pagerank at least Δ and 1 otherwise. Note that the cost of an algorithm equals its probability of failure. Then by Yao's Minimax Principle, any randomized algorithm that makes at most $\frac{n}{6\Delta}$ queries will have an expected cost of at least $1/e$, i.e., a failure probability of at least $1/e$ on at least one of the inputs. □

Theorem 5.4 (Graphs with Many Significant Nodes)**.** *Let $\alpha = 0.5$, Δ be integral and c be given. Then, there are infinitely many n such that there exists a graph on n nodes where the output to* SIGNIFICANTPAGERANKS *on that graph has size $\Omega(\frac{n}{\Delta})$.*

Proof. The construction is a variant of the one used in the proof of Theorem 5.4. The graph is made of $\frac{n}{(3\Delta+1)}$ identical copies of an undirected star graph on $d+1 = 3\Delta$ nodes. An easy calculation with the PageRank equations shows that each hub has PageRank of $\Delta + \frac{1}{3}$ and each leaf has PageRank of $\frac{2}{3} + \frac{1}{(9\Delta-3)} \leq 1$. The number of nodes with PageRank at least Δ is therefore $\frac{n}{d} = \Omega(\frac{n}{\Delta})$. □

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] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized pagerank. *PVLDB*, 4(3):173–184, 2010.

[5] Pavel Berkhin. Survey: A survey on pagerank computing. *Internet Mathematics*, 2(1), 2005.

[6] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[7] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[8] John R. Gilbert, Cleve Moler, and Robert Schreiber. Sparse matrices in matlab: design and implementation. *SIAM J. Matrix Anal. Appl.*, 13(1):333–356, January 1992.

[9] Oded Goldreich. Introduction to testing graph properties. In *Property Testing*, pages 105–141, 2010.

[10] 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.

[11] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *WWW*, pages 271–279, 2003.

[12] Ravi Kannan and Santosh Vempala. Spectral algorithms. *Foundations and Trends in Theoretical Computer Science*, 4(3-4):157–288, 2009.

[13] Ravindran Kannan. Spectral methods for matrices and tensors. In *STOC*, pages 1–12, 2010.

[14] Rajeev Motwani and Prabhaker Raghavan. *Randomized Algorithms*. Cambridge University Pres, 1995.

[15] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Stanford University 1998.

[16] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Using pagerank to characterize web structure. *Internet Mathematics*, 3(1):1–20, 2006.

[17] Ronitt Rubinfeld and Asaf Shapira. Sublinear time algorithms. *SIAM Journal on Discrete Math*, 25:1562–1588, 2011.

[18] Daniel Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM J. Comput.*, 42(1):1–26, 2013.

[19] Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *FOCS*, pages 222–227, 1977.

A Concentration Bounds

Lemma A.1. *(Multiplicative Chernoff Bound) Let $X = \sum_{i=1}^n X_i$ be a sum of independent (but not necessarily identical) Bernoulli random variables. Then,*

1. *For $0 < \lambda < 1$,*

$$\begin{aligned} Pr[X < (1-\lambda)E[X]] &< \exp(-\tfrac{\lambda^2}{2}E[X]) \\ Pr[X > (1+\lambda)E[X]] &< \exp(-\tfrac{\lambda^2}{4}E[X]). \end{aligned}$$

2. *For $\lambda \geq 1$,*

$$\Pr[X > (1+\lambda)E[X]] < \exp\left(-\frac{\lambda E[X]}{3}\right).$$

3. *For any constant $\Delta \geq (1+\lambda)E[X]$,*

$$Pr[X > \Delta] < \begin{cases} \exp(-\frac{\lambda^2}{4} \cdot \frac{\Delta}{1+\lambda}) & \text{if } 0 < \lambda < 1 \\ \exp(-\frac{\lambda}{3} \cdot \frac{\Delta}{(1+\lambda)}) & \text{if } \lambda \geq 1 \end{cases}$$

Proof. The case of $0 < \lambda < 1$ is standard and a proof can be found, for example, in chapter 4 of [14]. For any λ, it is also shown therein that

$$\Pr[X > (1+\lambda)\mu n] \leq \left(\frac{e^\lambda}{(1+\lambda)^{(1+\lambda)}}\right)^{\mu n}.$$

Now for $\lambda \geq 1$,

$$\frac{e^\lambda}{(1+\lambda)^{(1+\lambda)}} < \exp\left(-\frac{\lambda^2}{2+\lambda}\right) \leq \exp\left(-\frac{\lambda}{3}\right),$$

and the second claimed item follows.

We now prove the last claimed item. Assume that $\frac{\Delta}{(1+\lambda)} - E[X] > 0$ (otherwise the proof follows immediately from part 1). Define $k = \lceil \frac{\Delta}{(1+\lambda)} - E[X] \rceil$ and $Y = \sum_{i=1}^{n+k} Y_i$, where for $1 \leq i \leq n$, $Y_i = X_i$ and for $n < i \leq n+k$, Y_i are independently distributed Bernoulli random variables with expectation $(\frac{\Delta}{(1+\lambda)} - E[X])/k$ each. Note that $k \geq 1$, Y_i are indeed Bernoulli random variables as $0 < (\frac{\Delta}{1+\lambda} - E[X])/k \leq 1$, and that $E[Y] = E[X] + (\frac{\Delta}{(1+\lambda)} - E[X]) = \frac{\Delta}{(1+\lambda)}$. Now,

$$Pr(X > \Delta) = Pr(X > (1+\lambda)\Delta/(1+\lambda)) \leq$$

$$Pr(Y > (1+\lambda)\Delta/(1+\lambda)) < \begin{cases} \exp(-\frac{\lambda^2}{4} \cdot \frac{\Delta}{1+\lambda}) & \text{if } \lambda < 1 \\ \exp(-\frac{\lambda}{3} \cdot \frac{\Delta}{(1+\lambda)}) & \text{if } \lambda \geq 1 \end{cases}$$

The next to last inequality follows from the fact that Y first-order stochastically dominates X, and the last inequality follows from parts 1 and 2 of the lemma. $\square$