

MIT Open Access Articles

*Deterministic Coresets for Stochastic Matrices
with Applications to Scalable Sparse PageRank*

The MIT Faculty has made this article openly available. **Please share**
how this access benefits you. Your story matters.

Citation: Lang, Harry, Baykal, Cenk, Samra, Najib Abu, Tannous, Tony, Feldman, Dan et al. 2019. "Deterministic Coresets for Stochastic Matrices with Applications to Scalable Sparse PageRank."

As Published: 10.1007/978-3-030-14812-6_25

Publisher: Springer International Publishing

Persistent URL: <https://hdl.handle.net/1721.1/137195>

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

Terms of use: Creative Commons Attribution-Noncommercial-Share Alike



Deterministic Coresets for Stochastic Matrices with Applications to Scalable Sparse PageRank*

Harry Lang^{1†}, Cenk Baykal^{1†}, Najib Abu Samra², Tony Tannous², Dan Feldman²,
and Daniela Rus¹

¹ MIT CSAIL, Cambridge, USA {baykal, harry1, rus}@mit.edu

² University of Haifa, Computer Science Department, Israel najib.as1990@gmail.com,
tonytanios1994@gmail.com, dannyf.post@gmail.com

Abstract. The PageRank algorithm is used by search engines to rank websites in their search results. The algorithm outputs a probability distribution that a person randomly clicking on links will arrive at any particular page. Intuitively, a node in the center of the network should be visited with high probability even if it has few edges, and an isolated node that has many (local) neighbours will be visited with low probability. The idea of PageRank is to rank nodes according to a stable state and not according to the previous local measurement of inner/outer edges from a node that may be manipulated more easily than the corresponding entry in the stable state.

In this paper we present a deterministic and completely parallelizable algorithm for computing an ε -approximation to the PageRank of a graph of n nodes. Typical inputs consist of millions of pages, but the average number of links per page is less than ten. Our algorithm takes advantage of this sparsity, assuming the out-degree of each node at most s , and terminates in $O(ns/\varepsilon^2)$ time. Beyond the input graph, which may be stored in read-only storage, our algorithm uses only $O(n)$ memory. This is the first algorithm whose complexity takes advantage of sparsity. Real data exhibits an average out-degree of 7 while n is in the millions, so the advantage is immense. Moreover, our algorithm is simple and robust to floating point precision issues. Our sparse solution (core-set) is based on reducing the PageRank problem to an ℓ_2 approximation of the Carathéodory problem, which independently has many applications such as in machine learning and game theory. We hope that our approach will be useful for many other applications for learning sparse data and graphs.

Algorithm, analysis, and open code with experimental results are provided.

Keywords: First keyword · Second keyword · Another keyword.

*Lang, Baykal, and Rus thank NSF 1723943, NSF 1526815, and The Boeing Company. This research was supported by Grant No. 2014627 from the United States-Israel Binational Science Foundation (BSF) and by Grant No. 1526815 from the United States National Science Foundation (NSF). Dan Feldman is grateful for the support of the Simons Foundation for part of this work that was done while he was visiting the Simons Institute for the Theory of Computing.

[†]These authors contributed equally to this work

1 Introduction

Matrix notation. For an integer $n \geq 1$, let $[n] = \{1, \dots, n\}$. We denote by $\mathbb{R}^{n \times n}$ the set of $n \times n$ real matrices. The j^{th} column of G is denoted by $G_{\bullet j}$ and the entry on its i^{th} row is G_{ij} . The i^{th} entry of a column vector $v \in \mathbb{R}^n$ is denoted by v_i . We write 0_n for the n -dimensional zero vector, and I_n for the $n \times n$ identity matrix. Let e_j denote the j^{th} column of I_n .

Stochastic matrix. A *distribution vector* $z \in [0, 1]^n$ is a non-negative vector whose sum is 1. A *column-stochastic matrix* is a matrix such that every one of its columns is a distribution vector. The input matrix G is called the *transition matrix* of a graph if it is equal to the adjacency matrix but with each non-zero column scaled to have unit sum. Every positive column-stochastic matrix $A \in \mathbb{R}^{n \times n}$ (i.e. whose entries are positive) has a distribution vector $z^* \in \mathbb{R}^n$ such that $Az^* = z^*$ by the Perron-Frobenius theorem. This vector is called the *stable state* of A .

Problem setup. The input to the PageRank problem is a non-negative square matrix $G \in [0, 1]^{n \times n}$ that represents the scaled adjacency matrix of a graph, i.e. the entry G_{ij} represents the probability of moving from node i to node j , and each column of G is scaled so that its sum is 1. For simplicity, we assume (only for the moment) that there is no node with out-degree 0.

Hence, G is a column-stochastic matrix where each of its columns defines a distribution, i.e., a vector whose entries are non-negative and sum to 1. For example, its second column $G_{\bullet 2}$ defines the visited node after taking a single step (random walk) from the second node to its random neighbour. For a given vector $z \in [0, 1]^n$ that defines a distribution over the currently visited nodes, the multiplication $y = Gz$ yields the distribution y of the visited nodes after taking a single random step from the current state. In this sense, G describes a Markov Chain. If $Gz^* = z^*$ for some distribution z^* , then z^* is called a *stable state* of G . Hence, given an initial distribution z^* , the distributed on the next visited node will not change.

Stable state and page's rank. The stable state z^* is important because it can be also be proved roughly that the probability of stopping on the i^{th} node after a sufficiently long random walk (when the initial visited node has almost no influence), approaches z_i^* . Intuitively, a node in the center of the network should be visited with high probability even if it has few edges, and an isolated node that has many (local) neighbours will be visited with low probability.

The idea of PageRank is to rank nodes according to this stable state z^* and not according to the previous local measurement of inner/outer edges from a node, since these may be manipulated more easily than the corresponding entry in the stable state. Intuitively, this is because the stable state measures a global property that depends on all the nodes in the graph (connecting a single edge in one side of the graph changes the rank of a node even of the other side of the graph). Unfortunately, this is also why the problem of computing the stable state of G is hard. In fact, this vector is not unique.

Damping Factor. To solve this issue, the PageRank algorithm uses a given a parameter $d \in (0, 1)$ which is called the *damping factor*. A common value is $d = 0.15$. Let $\mathbf{1}$ denote the $n \times n$ matrix whose all entries are 1, and let $A = (1 - d)G + \frac{d}{n} \cdot \mathbf{1}$. For example, suppose that in $G_{ji} = G_{ki} = 1/2$ for the transitions $i \rightarrow j$ and $i \rightarrow k$. In the matrix A , the probability of moving to any other node increases from 0 to d/n , and the probability $1/2$ of moving from i to j or k is changed to $\frac{1-d}{2} + \frac{d}{n}$. The new sum of probabilities is still 1, so A is now a positive column-stochastic matrix. In the context of surfing the internet, if each column of G represents links in a web-page, in A , with probability d we will visit a new random page that is not linked from our current web-page.

More generally, in the *v-Personalized PageRank* we get an additional input distribution vector $v \in (0, 1)^n$ whose entries are strictly positive and sum to one, and define $A = (1 - d)G + d \cdot v \cdot (1, \dots, 1)$. In this case, v defines a distribution over the webpages to visit in case that no web-link is chosen in the existing page. For the uniform distribution $v = (1/n, \dots, 1/n)^T$ the *v-Personalized PageRank* is the same as PageRank.

Unlike G , which is a non-negative matrix, A is a *positive* column-stochastic matrix. By this fact and the Perron-Frobenius theorem, A has a unique stable state distribution z^* such that $Az^* = z^*$. Hence, the PageRank problem reduced to the problem of computing this vector. A common way to compute the largest eigenvector of such a matrix A is to use the power method. This iterative technique computes $z^{(0)} = A \cdot (1/n, \dots, 1/n)^T$, and $z^{(i)} = Az^{(i-1)}$ for every $i \geq 1$. It can be proven that the approximation error $\|Az^{(i)} - z^{(i)}\|_2 \leq \varepsilon$ after $O(\log(n)/\varepsilon)$ iterations. Recall that $\|Az^* - z^*\|_2 = 0$ since $Az^* = z^*$.

Our result. In this paper we suggest a *deterministic* algorithm for computing a provable ε -approximation z to the PageRank z^* of a graph with n nodes. The exact running time and memory depends on the sparsity s of each column of G , i.e. the maximal number of neighbours or, more precisely the out-degree, of each node in G . This number is small in social or web networks, where most of the nodes have few related links to neighbor nodes, compared to the total number n of nodes in the network. In fact, real datasets show $s \approx 7$ and $n > 10^6$ [8], making the improvement from $O(n^2)$ to $O(ns)$ quite drastic.

Formally, we present an algorithm that computes a distribution vector z such that $\|Az - z\|_2 \leq \varepsilon$, compared to the optimal solution where $\|Az^* - z^*\| = 0$. The algorithm runs in $O(ns/\varepsilon^2)$ time and requires only $O(n)$ read-write memory in addition to reading the input graph (which is sparsely represented using $O(ns)$ memory).

Our output z is a sparse vector with $O(1/\varepsilon^2)$ non-zero entries. Intuitively, the non-zero entries of z are the “heavy hitters” or important nodes, while small ranks are rounded to 0. Our algorithm is iterative, and the user can stop it after finding the top desired ranks, or after observing that the last returned rank is already small.

Our Techniques. We show that the PageRank problem can be reduced to the Approximated Carathéodory, which was recently used in applications such as machine learning, and game theory [5]. In this problem we wish to approximate a point in the convex hull of n points by a convex combination of a small subset of these points.

The problem of computing z^* such that $Az^* = z^*$ is the same as computing z^* such that $(A - \mathcal{I}_n)z^* = 0$. We define $B = A - \mathcal{I}_n$ and seek a distribution vector z such that $\|Bz\|_2 \leq \varepsilon$. Viewing the columns of B as points in \mathbb{R}^d , the problem is reduced to finding a convex combination of points close to the origin. We instantiate the Frank-Wolfe algorithm to iteratively add columns of B until we arrive sufficiently close to the origin. At most $O(1/\varepsilon^2)$ columns are needed before we arrive at a sufficient solution.

While the input G has sparse columns, the matrix B is not sparse because of the damping factor d and distribution vector v . In fact, B has no zeros. We show a technique to cache previous computations and extract the relevant information from G , d , and v without every actually computing B .

For simplicity, we prove the main claim for the PageRank application, but the proof is written in more general notation that we hope will be used by other researchers for other functions, such as estimators that are robust to outliers as the m -estimators, that usually have the required smoothness condition.

Related work. A frequently explored technique for solving PageRank involves some variant of random walks, appearing in results such as [12,2,6,16,17]. A sampling based approach is suggest in [1]. Multipass streaming algorithms for approximating the rank (i.e. value in the stable state) of a single node have been explored [15]. Under some assumptions, it is also possible to return a list of the top- k ranked nodes [3].

In the distributed setting, [17] present a $O(\frac{\log n}{d})$ -round algorithm that approximates each entry of the stable state up to a multiplicative factor of $(1 + \varepsilon)$. However, they treat computation time as an unlimited resource and the runtime is not explicitly bounded.

Extensions of PageRank where the stable state may fluctuate in time have been explored [13,4,14,18]. When considering location-based networks, an algorithm was presented by [9].

Classical techniques include the power method (iterating until the stable state is approached asymptotically) and explicit computation (involving finding a matrix inverse). These classical techniques are deterministic but require $\omega(n^2)$ time. All prior $O(n^2)$ time algorithms require randomization, making ours the first fast algorithm to be entirely deterministic. Moreover, we take advantage of sparsity and therefore run in optimal $O(n)$ time on real datasets whose average out-degree is around 7 [8].

2 Preliminaries

The network of n pages is represented by an $n \times n$ matrix G . The i^{th} column represents the links on the i^{th} page. If a page contains no links, its column is filled with zeros. Otherwise, the column's entries sum to 1 where the j^{th} entry represents the probability that a user who clicks on a link from page i will follow a link to page j .

Definition 1 (Unprocessed PageRank Matrix). *The input to the PageRank problem is a non-negative $n \times n$ matrix G where each column represents the links from a page. If the page is a sink node with no links to other pages, the column is all zeros. Otherwise, the column represents the transition probabilities to other pages and has unit ℓ_1 norm. We call such a matrix the unprocessed PageRank matrix.*

With probability $d \in (0, 1)$, a user may jump to page without clicking a link from the current page. The distribution vector v represents the probabilities that a user will randomly jump to a particular page. The matrix Ψ , introduced in the next definition, accounts for this possibility of jumping without following any link. The entry Ψ_{ij} is the probability that a user on page j will move to page i .

Definition 2 (Approximate PageRank). Let $G \in \mathbb{R}^{n \times n}$ be an unprocessed PageRank matrix. Given a damping factor $d \in (0, 1)$ and a positive distribution vector v , the processed PageRank matrix $\Psi(G, d, v)$ is defined column-by-column as:

$$\Psi(G, d, v)_{\bullet i} = \begin{cases} v & \text{if } G_{\bullet i} = 0 \\ dG_{\bullet i} + (1-d)v & \text{if } \|G_{\bullet i}\|_1 = 1 \end{cases}$$

Note that $\Psi(G, d, v)$ is a positive column-stochastic matrix, so there exists a unique distribution vector z^* such that $\Psi(G, d, v)z^* = z^*$. A distribution vector x is called an ε -approximation if $\|\Psi(G, d, v) - \mathcal{I}_n\|_2 \leq \varepsilon$.

We relate PageRank to a well-studied geometric problem called the Carathéodory problem. In what follows, 0_d denotes the origin in \mathbb{R}^d , and $\mathcal{C}(P)$ denotes the convex hull of a point-set P .

Definition 3. Let P be a set of n points in \mathbb{R}^d such that $0_d \in \mathcal{C}(P)$. An ε -approximation to the ℓ_2 -Carathéodory Problem is a multiset Q of T points from P such that:

$$\left\| \frac{1}{T} \sum_{q \in Q} q \right\|_2 \leq \varepsilon$$

In the previous definition, the unweighted average is taken over Q . However, Q is a multiset meaning that it may contain points with multiplicity. One can consider a point with multiplicity m to have weight $\frac{m}{T}$. As a simple example, let $P \subset \mathbb{R}^2$ be three points whose convex hull contains the origin. Depending on the location of the origin, an arbitrarily large Q is required for sufficiently small ε despite the fact that P contains only three distinct points.

Definition 4 (Smoothness). A continuously differentiable function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is said to be β -smooth with respect to norm $\|\cdot\|$ on domain $D \subset \mathbb{R}^d$ if:

$$\|\nabla f(x) - \nabla f(y)\| \leq \beta \|x - y\|$$

for every $x, y \in D$.

For $p \geq 1$ we write $\|x\|_p$ to denote the ℓ_p -norm of x which is $(x_1^p + \dots + x_d^p)^{\frac{1}{p}}$. Observe that the function $\|x\|_p^p$ is convex and continuously differentiable.

Lemma 1. The function $f(x) = \|x\|_2^2$ is 2-smooth with respect to any norm on \mathbb{R}^d .

Proof.

$$\begin{aligned}\nabla(\|x\|_2^2) &= \nabla(x_1^2 + \dots + x_d^2) \\ &= (2x_1 \dots 2x_d)^\top \\ &= 2x\end{aligned}$$

Therefore $\|\nabla f(x) - \nabla f(y)\| = 2\|x - y\|$ where we have pulled out the 2 by the scalability property of a norm.

Outline of Presentation. In the next section, we present the well-known Frank-Wolfe algorithm (Algorithm 1 and Theorem 1). We show that Frank-Wolfe can be used to solve the Carathéodory problem (Corollary 1). We will transform the PageRank problem so it can be solved by Frank-Wolfe, presenting a simple deterministic algorithm (Algorithm 2 and Lemma 3). Finally, we make modifications to the algorithm so that it takes advantage of the sparsity of the input (Algorithm 3 and Theorem 2).

3 Algorithm

The following algorithm was first presented by Frank and Wolfe in 1956 [7]. As written, it loops infinitely. In practice, we terminate the loop after a sufficient number of iterations given by guarantees such as in Theorem 1.

We begin by setting x to be an arbitrary point of P . In each iteration, we select a point y which is a vertex of the convex hull of P . We then redefine x as a weighted average of itself and y , where each iteration gives less weight to the new y .

Algorithm 1 Input: continuously differentiable function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ and point set $P = \{p_1, \dots, p_n\} \subset \mathbb{R}^d$

```

1:  $x^{(0)} \leftarrow p_1$ 
2:  $t \leftarrow 1$ 
3: while true do
4:    $\eta^{(t)} \leftarrow 1/t$ 
5:    $y^{(t)} \leftarrow \arg \min_{y \in P} y^\top \nabla f(x^{(t-1)})$ 
6:    $x^{(t)} \leftarrow (1 - \eta^{(t)})x^{(t-1)} + \eta^{(t)}y^{(t)}$ 
7:    $t \leftarrow t + 1$ 

```

The first observation is that the $\{x^{(t)}\}$ are simply averages of the $\{y^{(t)}\}$.

Lemma 2. $x^{(t)} = \frac{1}{t}(y^{(1)} + \dots + y^{(t)})$ for every $t \geq 1$.

Proof. Observe directly from Line 6 that $x^{(1)} = y^{(1)}$. Now assume inductively that $x^{(t-1)} = \frac{1}{t-1}(y^{(1)} + \dots + y^{(t-1)})$. $\eta^{(t)} = \frac{1}{t}$ and so:

$$x^{(t)} = \left(1 - \frac{1}{t}\right)x^{(t-1)} + \frac{1}{t}y^{(t)}$$

$$\begin{aligned}
&= \left(\frac{t-1}{t} \right) \left(\frac{1}{t-1} \right) (y^{(1)} + \dots + y^{(t-1)}) + \frac{1}{t} y^{(t)} \\
&= \frac{1}{t} (y^{(1)} + \dots + y^{(t)})
\end{aligned}$$

The following theorem governs the behavior of the Frank-Wolfe algorithm, and was presented in the original paper.

Theorem 1 ([7]). *Fix a norm $\|\cdot\|$. If Algorithm 1 is run for T iterations on input (f, P) where $\max_{p \in P} \|p\| \leq R$ and f is a β -smooth convex function with respect to $\|\cdot\|$ on $\mathcal{C}(P)$, then:*

$$f(x_T) - \min_{x \in \mathcal{C}(P)} f(x) \leq \frac{2\beta R^2}{T+1}$$

We arrive at the following corollary by applying the previous theorem with basic manipulations. In this form, it is apparent how the Carathéodory problem may be solved using Frank-Wolfe.

Corollary 1. *Let $f(x) = \|x\|_2^2$ and $T = \lceil \frac{8}{\varepsilon^2} - 1 \rceil$. If Algorithm 1 is run for T iterations on input (f, P) where $\max_{p \in P} \|p\|_2 \leq \sqrt{2}$ and $0 \in \mathcal{C}(P)$, then:*

$$\|x^{(T)}\|_2 \leq \varepsilon$$

We now show that Algorithm 1 provides an algorithm for PageRank, forging a connection between the two problems. Observe that when we form a $d \times n$ matrix B whose columns are the points of P , then $\mathcal{C}(P)$ is the union of Bz over all distribution vectors $z \in \mathbb{R}^n$.

Algorithm 2 outputs an ε -approximation for Personalized PageRank. When adapting Algorithm 1, we have set $f(x) = \|x\|_2^2$ and so $\nabla f(x^{(t-1)})$ is simply $2x^{(t-1)}$. We also introduce a new vector z which will be the approximate stable state.

Lemma 3. *Algorithm 2 outputs a distribution vector z that satisfies $\|Bz\|_2 \leq \varepsilon$.*

Proof. It is clear that z is a distribution vector since Line 10 is executed T times so z is non-negative and $\|z\|_1 = 1$. It remains to show the bound $\|Bz\|_2 \leq \varepsilon$.

Given the positive column-stochastic matrix $B = \Psi(G, d, v) - \mathcal{I}_n$, we wish to find a vector z such that $\|z\|_1 = 1$ and $\|Bz\|_2 \leq \varepsilon$. Let $P \subset \mathbb{R}^n$ be the columns of B . By the Perron-Frobenius theorem, there exists a unique stable state z^* such that $Bz^* = 0$. This implies that the convex hull of P contains the origin, since the values of z^* define weightings of a convex combination of the columns of B . The reader can verify that $\|A_{\bullet i}\|_1 = 1$ implies $\|B_{\bullet i}\|_2 \leq \sqrt{2}$. Therefore by Corollary 1, Algorithm 1 will satisfy that $\|x^{(T)}\|_2 \leq \varepsilon$ since we have set $T = \lceil \frac{8}{\varepsilon^2} - 1 \rceil$ on Line 4.

It now suffices to show that $Bz = x^{(T)}$. Observe that $Be_i = B_{\bullet i}$. Let $j^{(t)}$ be the value set on Line 8 during the t^{th} iteration of the while-loop. $y^{(t)} = B_{\bullet j^{(t)}} = Be_{j^{(t)}}$. By Line 10, $z = \frac{1}{T} \sum_{t \leq T} e_{j^{(t)}}$ when the algorithm terminates. Therefore:

$$Bz = \frac{1}{T} \sum_{t \leq T} Be_{j^{(t)}}$$

$$\begin{aligned}
&= \frac{1}{T} \sum_{t \leq T} y^{(t)} \\
&= x^{(T)}
\end{aligned}$$

where the last line follows from Lemma 2.

Algorithm 2 Input: an unprocessed $n \times n$ PageRank matrix G , damping factor $d \in (0, 1)$, positive distribution vector $v \in (0, 1)^n$, approximation factor $\varepsilon > 0$

```

1:  $B \leftarrow \Psi(G, d, v) - \mathcal{I}_n$ 
2:  $x^{(0)} \leftarrow B_{\bullet 1}$ 
3:  $t \leftarrow 1$ 
4:  $T \leftarrow \lceil \frac{8}{\varepsilon^2} - 1 \rceil$ 
5:  $z \leftarrow 0_n$ 
6: while  $t \leq T$  do
7:    $\eta^{(t)} \leftarrow 1/t$ 
8:    $j \leftarrow \arg \min_{i \in [n]} B_{\bullet i}^\top x^{(t-1)}$ 
9:    $y^{(t)} \leftarrow B_{\bullet j}$ 
10:   $z_j \leftarrow z_j + \frac{1}{T}$ 
11:   $x^{(t)} \leftarrow (1 - \eta^{(t)})x^{(t-1)} + \eta^{(t)}y^{(t)}$ 
12:   $t \leftarrow t + 1$ 
13: Output  $z$ 

```

Theorem 3 guarantees an ε -approximation, but the matrix $B = \Psi(G, d, v) - \mathcal{I}_n$ is not sparse. In fact, B does not have any zero entries. As written, Algorithm 2 runs in $O(n^2/\varepsilon^2)$ time. In the next section, we will improve the runtime by taking advantage of the fact that although B is not sparse, it can be represented as a function of the sparse matrix G .

4 Sparse Runtime

Each column of G has at most s non-zero entries. We can leverage this in the computation to improve the $O(n^2/\varepsilon^2)$ runtime of Algorithm 2 to the vastly improved $O(ns/\varepsilon^2)$ time. Recall that the average out-degree of a page is around 7 while the total number of pages is at least in the millions, so this causes a drastic improvement.

To take advantage of the sparsity, we assume that the columns of G are stored in memory with an *s-sparse representation* which means a list of at most s index-value pairs. In contrast, by a *standard representation* we mean a list of n values including the zeros such that the value for any index can be retrieved in $O(1)$ time.

Fact 1. *Let u and w be two vectors in \mathbb{R}^n . If u has an s -sparse representation, then $u^\top w$ can be computed in $O(s)$ time and $O(1)$ space if w has either an s -sparse representation or a standard representation.*

The proof of Fact 1 is clear. When u is represented as a list of s index-value pairs, we need not even bother to read the values of any entries of v that are not in the list of u . Since these entries are zero in u , their value in v is irrelevant.

We begin with a lemma on how to compute the dot product efficiently between columns of B . This is important because we must compute (see Line 8 of Algorithm 2) the value of i that minimizes the dot product $B_{\bullet i}^\top x^{(t-1)}$. Recall from Lemma 2 that $x^{(t-1)} = \frac{1}{t-1}(y^{(1)} + \dots + y^{(t-1)})$. From Line 9, each $y^{(i)}$ is a column of B . Therefore we may distribute and write this dot product as a sum of dot products between columns of B .

Lemma 4. *Assume that the value $v^\top v$ is known. For any $1 \leq i, j \leq n$, the value $B_{\bullet i}^\top B_{\bullet j}$ can be computed in $O(s)$ time and $O(1)$ memory.*

Proof. We break into three cases about the two columns $G_{\bullet i}$ and $G_{\bullet j}$: (1) both are zero-columns, (2) only $G_{\bullet j}$ is a zero-column, (3) neither are zero-columns. Note that Case (2) has no loss of generality since if only $G_{\bullet i}$ is a zero-column then we can simply swap i and j before proceeding.

In what follows, we assume that $i \neq j$. If $i = j$ we simply add 1 due to the term $e_i^\top e_j$. Otherwise $e_i^\top e_j = 0$.

Case 1:

$$\begin{aligned} B_{\bullet i}^\top B_{\bullet j} &= (v - e_i)^\top (v - e_j) \\ &= v^\top v - v_i - v_j \end{aligned}$$

Case 2:

$$\begin{aligned} B_{\bullet i}^\top B_{\bullet j} &= (dG_{\bullet i} + (1-d)v - e_i)^\top (v - e_j) \\ &= dG_{\bullet i}^\top v + (1-d)v^\top v - v_i - dG_{ji} + (1-d)v_j \end{aligned}$$

Case 3:

$$\begin{aligned} B_{\bullet i}^\top B_{\bullet j} &= (dG_{\bullet i} + (1-d)v - e_i)^\top (dG_{\bullet j} + (1-d)v - e_j) \\ &= d^2 G_{\bullet i}^\top G_{\bullet j} + d(1-d)(G_{\bullet i}^\top v + G_{\bullet j}^\top v) \\ &\quad - d(G_{ij} + G_{ji}) - (1-d)(v_i + v_j) \\ &\quad + (1-d)^2 v^\top v \end{aligned}$$

Given the input tuple (G, d, v, X, i, j) where $X = v^\top v$, the time and space complexities are proven as follows. Case 1 is simple since we return X along with entries for v (retrievable in $O(1)$ time from the standard representation). For Case 2, we must also use Fact 1 to address the $G_{\bullet i}^\top v$ term. For the G_{ji} term, this can be found in $O(s)$ time (not necessarily $O(1)$ time since $G_{\bullet i}$ has an s -sparse representation). Case 3 follows using the same principles.

We write $\text{DOT}(G, d, v, X, i, j)$ to denote the $O(s)$ time procedure from Lemma 4 which computes $B_{\bullet i}^\top B_{\bullet j}$ given that $X = v^\top v$. We use this procedure in Algorithm 3 which is an efficient implementation of Algorithm 2.

Algorithm 3 Input: an unprocessed $n \times n$ PageRank matrix G , damping factor $d \in (0, 1)$, positive distribution vector $v \in (0, 1)^n$, approximation factor $\varepsilon > 0$

```

1:  $T \leftarrow \lceil \frac{8}{\varepsilon^2} - 1 \rceil$ 
2:  $X \leftarrow v^\top v$ 
3:  $j \leftarrow 1$ 
4:  $\text{CACHE} \leftarrow 0_n$ 
5:  $z \leftarrow 0_n$ 
6: for  $t \in [T]$  do
7:    $j \leftarrow \arg \min_{i \in [n]} \text{CACHE}_i + \text{DOT}(G, d, v, X, i, j)$ 
8:    $z_j \leftarrow z_j + \frac{1}{T}$ 
9:   for  $i \in [n]$  do
10:     $\text{CACHE}_i \leftarrow \text{CACHE}_i + \text{DOT}(G, d, v, X, i, j)$ 
11: Output  $z$ 

```

The vectors $x^{(t)}$ and $y^{(t)}$, which were central to Algorithms 1 and 2, are only implicitly present in Algorithm 3. At the end of the t^{th} iteration of the while-loop, $y^{(t)} = B_{\bullet j}$ and $x^t = \frac{T}{t} Bz$.

The vector CACHE stores previous computations of $B_{\bullet i}^\top x^{(t)}$ so we can simply compute $B_{\bullet i}^\top y^{(t+1)}$ in $O(s)$ time instead of re-computing this entire quantity in $O(ts)$ time. Without caching the computation, we would have an overall runtime of $O(ns/\varepsilon^4)$.

Lemma 5. *In Algorithm 3, $\text{CACHE}_i = tB_{\bullet i}^\top x^{(t)}$ after the t^{th} iteration of the while-loop. Here $x^{(t)}$ is the value from Algorithm 2.*

Proof. Let $\text{CACHE}^{(t)}$ denote the value of CACHE after the t^{th} iteration of the while-loop. Then $\text{CACHE}^{(0)} = 0_n$. By Lemma 2, $tx^{(t)} = y^{(1)} + \dots + y^{(t)}$. By induction, it suffices to show that $\text{CACHE}_i^{(t)} = \text{CACHE}_i^{(t-1)} + B_{\bullet i}^\top y^{(t)}$.

After Line 7 of the t^{th} iteration, j is such that $y^{(t)} = B_{\bullet j}$. Lemma 4 establishes that $\text{DOT}(G, d, v, X, i, j) = B_{\bullet i}^\top B_{\bullet j}$. Combining these statements shows that Line 10 sets $\text{CACHE}_i^{(t)} = \text{CACHE}_i^{(t-1)} + B_{\bullet i}^\top y^{(t)}$ as desired.

We now present our main theoretical result. Algorithm 3 is simple, deterministic, parallelizable, and takes advantage of the sparsity of the input.

Theorem 2. *Algorithm 3 is deterministic, terminates in $O(ns/\varepsilon^2)$ time, requires $O(n)$ memory in addition to the read-only input, and returns an ε -approximation for Personalized PageRank of an input (G, d, v) . Moreover, the algorithm is parallelizable with $P \leq n$ processors with runtime $O((\frac{n}{P} + \log P)s/\varepsilon^2)$ and the same memory requirement.*

Proof. It is clear that the algorithm is deterministic. Correctness follows from equivalence to Algorithm 2 which is straightforward to verify.

For runtime, Line 2 takes $O(n)$ time. On Line 7, there are n invocations of DOT , each of which takes $O(s)$ time by Lemma 4. Therefore this line takes $O(ns)$ time. The loop has $O(\frac{1}{\varepsilon^2})$ iterations, resulting in the runtime of $O(ns/\varepsilon^2)$.

For space, besides constant-size variables, only two vectors `CACHE` and z are stored in memory. It has been established in Lemma 4 that the `DOT` procedure requires $O(1)$ memory, so $P \leq n$ processors could compute Lines 2 and 7 in parallel in $O(n/P + \log P)$ time. The only other line of code requiring $\omega(1)$ time is Line 10, which can clearly be parallelized to $O(n/P)$ time.

5 Results

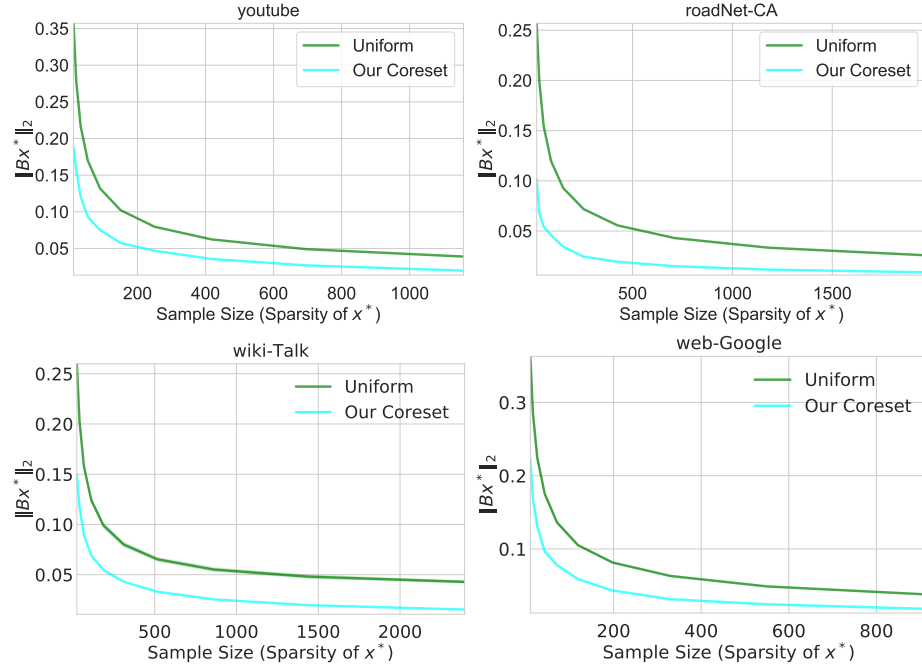


Fig. 1: Evaluation of the relative error $\|Bx^*\|_2 = \|(\Psi(G, d, v) - \mathcal{I}_n)x^*\|_2$ of the sparse, approximate distribution vector x^* .

In this section, we evaluate the practical effectiveness of our scalable and sparse PageRank algorithm on real world, benchmark data sets [11]. In particular, we evaluated our algorithm on the following data sets:

1. Youtube Social Network and Ground Truth Communities (*youtube*) — network representation of the Youtube social network. Contains 1,134,890 nodes and 2,987,624 edges.
2. California Road Network (*roadNet-CA*) — A road network of California consisting of 1,965,206 nodes and 2,766,607 edges.
3. Wikipedia Talk Network (*wiki-Talk*) — representation of the activity and discussions (edges) between the users (nodes) on the Wikipedia page. A directed edge

(u, v) between two users u and v exists if user u edited a page of user v . The graph contains 2,394,385 nodes and 5,021,410 edges.

4. Google Web Graph (*web-Google*) — Network composed of 875,713 nodes and 5,105,039 edges denoting web pages and the hyperlinks between them, respectively.

We compared the performance of our Frank-Wolfe-based approach in generating a sparse solution x that minimizes $\|Bx\|_2$ to that of constructing an x via uniform sampling for various values of sparsity (i.e., sample size). All algorithms were implemented in Python using the Sparse package of the SciPy library [10] and simulations were conducted on a computer with a 2.60 GHz Intel i9-7980XE processor (18 cores total) and 128 GB RAM.

Experimental Setup For each data set represented by the scaled adjacency matrix $G \in \mathbb{R}^{n \times n}$, we constructed a set of $k = 10$ geometrically-spaced subsample sizes $S = \{S_1, \dots, S_m\} \subseteq [\log n, \sqrt{n}]^k$ and for each sample size $m \in S$, we invoked each algorithm to construct an approximate distribution vector x with sparsity m . The results were averaged across 100 trials for each subsample size. More specifically, for each $m \in S$, we ran our algorithm for m iterations to construct an m -sparse vector x . The uniform sampling procedure was implemented in a similar iterative manner, where at each iteration a random index $j \in [n]$ was selected and the vector x (initially all zeros) was modified to be

$$x_j \leftarrow x_j + \frac{1}{T}.$$

Empirical Results Evaluations of our algorithm and comparisons to uniform sampling on the 4 benchmark networks are shown in Figure 1. Our empirical results reaffirm the favorable theoretical properties of our algorithm and show that it can efficiently and judiciously generate a sparse distribution vector with significantly smaller error than that constructed by uniform sampling.

References

1. Ahmed, N.K., Neville, J., Kompella, R.: Network sampling: From static to streaming graphs. *ACM Trans. Knowl. Discov. Data* **8**(2), 7:1–7:56 (Jun 2013). <https://doi.org/10.1145/2601438>, <http://doi.acm.org/10.1145/2601438> 4
2. Bahmani, B., Chakrabarti, K., Xin, D.: Fast personalized pagerank on mapreduce. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data. pp. 973–984. ACM (2011) 4
3. Bahmani, B., Chowdhury, A., Goel, A.: Fast incremental and personalized pagerank. *Proc. VLDB Endow.* **4**(3), 173–184 (Dec 2010). <https://doi.org/10.14778/1929861.1929864>, <http://dx.doi.org/10.14778/1929861.1929864> 4
4. Bahmani, B., Kumar, R., Mahdian, M., Upfal, E.: Pagerank on an evolving graph. In: Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 24–32. ACM (2012) 4
5. Barman, S.: Approximating nash equilibria and dense bipartite subgraphs via an approximate version of caratheodory’s theorem. In: Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing. pp. 361–369. ACM (2015) 3

6. Das Sarma, A., Nanongkai, D., Pandurangan, G.: Fast distributed random walks. In: Proceedings of the 28th ACM Symposium on Principles of Distributed Computing. pp. 161–170. PODC '09, ACM, New York, NY, USA (2009). <https://doi.org/10.1145/1582716.1582745>, <http://doi.acm.org/10.1145/1582716.1582745> 4
7. Frank, M., Wolfe, P.: An algorithm for quadratic programming. *Naval Research Logistics Quarterly* **3**(12), 95–110 (1956) 6, 7
8. Haveliwala, T., Kamvar, A., Klein, D., Manning, C., Golub, G.: Computing pagerank using power extrapolation (08 2003) 3, 4
9. Jin, Z., Shi, D., Wu, Q., Yan, H., Fan, H.: Lbsnrank: personalized pagerank on location-based social networks. In: Proceedings of the 2012 ACM Conference on Ubiquitous Computing. pp. 980–987. ACM (2012) 4
10. Jones, E., Oliphant, T., Peterson, P., et al.: SciPy: Open source scientific tools for Python (2001–), <http://www.scipy.org/>, [Online; accessed 7today] 12
11. Leskovec, J., Sosič, R.: Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)* **8**(1), 1 (2016) 11
12. Mitliagkas, I., Borokhovich, M., Dimakis, A.G., Caramanis, C.: Frogwild!: fast pagerank approximations on graph engines. *Proceedings of the VLDB Endowment* **8**(8), 874–885 (2015) 4
13. Rossi, R.A., Gleich, D.F.: Dynamic pagerank using evolving teleportation. In: International Workshop on Algorithms and Models for the Web-Graph. pp. 126–137. Springer (2012) 4
14. Rozenshtein, P., Gionis, A.: Temporal pagerank. In: Frasconi, P., Landwehr, N., Manco, G., Vreeken, J. (eds.) *Machine Learning and Knowledge Discovery in Databases*. pp. 674–689. Springer International Publishing, Cham (2016) 4
15. Sarma, A.D., Gollapudi, S., Panigrahy, R.: Estimating pagerank on graph streams. *J. ACM* **58**(3), 13:1–13:19 (Jun 2011). <https://doi.org/10.1145/1970392.1970397>, <http://doi.acm.org/10.1145/1970392.1970397> 4
16. Sarma, A.D., Molla, A.R., Pandurangan, G.: Near-optimal random walk sampling in distributed networks. *arXiv preprint arXiv:1201.1363* (2012) 4
17. Sarma, A.D., Molla, A.R., Pandurangan, G., Upfal, E.: Fast distributed pagerank computation. In: International Conference on Distributed Computing and Networking. pp. 11–26. Springer (2013) 4
18. Yu, W., Lin, X., Zhang, W.: Fast incremental simrank on link-evolving graphs. In: 2014 IEEE 30th International Conference on Data Engineering (ICDE). pp. 304–315. IEEE (2014) 4