Local Computation of PageRank Contributions

Reid Andersen ⋆, Christian Borgs ⋆⋆, Jennifer Chayes ⋆⋆,
John Hopcraft ⋆⋆⋆, Vahab S. Mirrokni ⋆⋆, and Shang-Hua Teng †

Abstract. Motivated by the problem of detecting link-spam, we consider the following graph-theoretic primitive: Given a webgraph G, a vertex v in G, and a parameter $\delta \in (0,1)$, compute the set of all vertices that contribute to v at least a δ fraction of v's PageRank. We call this set the δ-contributing set of v. To this end, we define the contribution vector of v to be the vector whose entries measure the contributions of every vertex to the PageRank of v. A local algorithm is one that produces a solution by adaptively examining only a small portion of the input graph near a specified vertex. We give an efficient local algorithm that computes an ϵ-approximation of the contribution vector for a given vertex by adaptively examining $O(1/\epsilon)$ vertices. Using this algorithm, we give a local approximation algorithm for the primitive defined above. Specifically, we give an algorithm that returns a set containing the δ-contributing set of v and at most $O(1/\delta)$ vertices from the $\delta/2$-contributing set of v, and which does so by examining at most $O(1/\delta)$ vertices. We also give a local algorithm for solving the following problem: If there exist k vertices that contribute a ρ-fraction to the PageRank of v, find a set of k vertices that contribute at least a $(\rho - \epsilon)$-fraction to the PageRank of v. In this case, we prove that our algorithm examines at most $O(k/\epsilon)$ vertices.

1 Introduction

In numerous applications of PageRank one needs to know, in addition to the rank of a given web page, which pages or sets of pages contribute most to its rank. These PageRank contributions have been used for link spam detection [4, 10] and in the classification of web pages [12]. A set of pages that contributes significantly to the PageRank of a page is often called a *contribution set* or *supporting set* of the page [4, 10].

The contribution that a vertex u makes to the PageRank of a vertex v is defined rigorously in terms of personalized PageRank. For a webgraph $G = (V, E)$ and a *teleportation constant* α (sometimes called the restart probability), let $\mathbf{PRM}_\alpha$ be the matrix whose u^{th} row is the personalized PageRank vector of u. The PageRank contribution of u to v, written $\mathrm{pr}_\alpha(u \to v)$, is defined to be the entry (u, v) of this matrix. The PageRank of a vertex v is the sum of the v^{th} column of the matrix $\mathbf{PRM}_\alpha$, and thus the PageRank of a vertex can be viewed as the sum of the contributions from all other vertices. The *contribution vector* of v is defined to be the v^{th} column of the matrix $\mathbf{PRM}_\alpha$, whose entries are the contributions of every vertex to the PageRank of v.

Given that the web graph is massive and getting larger at a substantial rate, it is essential to compute contribution vectors and identify supporting sets by examining as small a fraction of the graph as possible. In particular, it is helpful to design a *local* algorithm for computing the supporting sets of a particular vertex. Local algorithms search for a solution near a specified vertex by adaptively examining only a small subset of the input graph. They have been studied previously in distributed computing [16] and in graph partitioning and clustering [20, 2]. Personalized PageRank vectors can be approximated locally. Using one of several possible algorithms [14, 5, 19], it is possible to compute an approximation of the personalized

⋆ University of California at SanDiego, Sandiego, CA, Email: randerse@math.ucsd.edu
⋆⋆ Microsoft Research, Redmond, WA, Emails:{borgs,jchayes,mirrokni}@microsoft.com
⋆⋆⋆ Cornell University, Ithaca, NY, Email: jeh@cs.cornell.edu
† Boston University, Boston, MA, Email: steng@cs.bu.edu

PageRank vector of a vertex u by examining only $O(1/\epsilon)$ vertices, where ϵ is the desired amount of error at each vertex.

Problem Formulation. Inspired by local algorithms for computing personalized PageRank, and motivated by the importance of supporting sets in link-spam detection, we consider the problem of directly computing the contribution vector of a given vertex to quickly identify its supporting sets. In particular, we consider following graph-theoretic primitive: Given a webgraph G, a vertex v in G, and a parameter $\delta \in (0, 1)$, compute the set of all vertices each contributing at least a δ fraction to the PageRank of v. We call this set the *δ-contributing set of v*.

Such a primitive is useful for spam detection, since, given a webpage whose PageRank has recently increased suspiciously, we can quickly identify the set of pages that contribute significantly to the PageRank of that suspicious page. The above primitive may also be useful for analyzing social networks. In social networks in which the links capture the influence of vertices on each other, we can identify the nodes with the most influence to a given node.

Our Results. We give an efficient, local algorithm for computing an ϵ-approximation of the contribution vector for a given vertex v, a vector whose difference from the contribution vector is at most ϵ at each vertex. We prove that the number of the vertices examined by the algorithm is $O(1/\epsilon)$. The algorithm performs a sequence of probability-pushing operations on vertices of the graph, which we call pushback operations. When the pushback operation is applied to a vertex u, we perform a small amount of computation for each in-neighbor of u. Particularly, we add a fraction of a number stored at u to a number stored at each in-neighbor of u. The number of such operations that our algorithm performs is $O(1/\epsilon)$, and its running time can be bounded by the sum of the in-degrees of the vertices from which these operations were performed. To derive this algorithm, we adapt Jeh and Widom's technique for computing personalized PageRank vectors [14] to directly compute contribution vectors. To analyze the algorithm's running time and error bounds, we use techniques developed for the local clustering algorithm in [2].

Using our algorithm for approximating contribution vectors, we give an approximation algorithm to the primitive defined above. Explicitly, we give a local algorithm that returns a set containing the δ-contributing set of v and at most $O(1/\delta)$ vertices from the $\delta/2$-contributing set of v. Our algorithm applies at most $O(1/\delta)$ `pushback` operations. We also give a local algorithm for solving the following problem: If there are k vertices which contribute a ρ-fraction to the PageRank of v, find a set of k vertices which contribute at least $(\rho - \epsilon)$-fraction to the PageRank of v. In this case, we prove that our algorithm needs at most $O(k/\epsilon)$ `pushback` operations.

Finally, we remark that, in principle, one could directly compute the contribution vector for a vertex v by approximating the personalized PageRank vector of v in the time-reversal of the random walk Markov chain. We describe the computation required for this approach, and argue that for most graphs it is not as efficient as the method we propose.

Related Work. Supporting sets and PageRank contributions have been studied before as a tool for spam detection, notably in the SpamRank algorithm of Benczúr et al. [4], and in the Spam Mass algorithm of Gyöngyi et al. [10]. However, none of these papers developed a local algorithm for computing the contribution vector or supporting set. In the SpamRank algorithm [4], the contribution vectors are computed in the following way. One computes an approximation of each personalized PageRank vector in the graph to create an approximate PageRank matrix, and then takes the transpose of this matrix to obtain the approximate contribution vectors. This method is efficient for the task of computing the contribution vectors for every vertex in the graph, and it leverages fast algorithms for computing many personalized PageRank vectors simultaneously [9, 19], but it does not provide an efficient way to compute the contribution vectors of a

few selected suspicious vertices. Furthermore, the relative error in the resulting approximate contribution vectors may be larger than the relative error in the computed personalized PageRank vectors, since this is not preserved by the transpose operation.

PageRank contributions have also been used to estimate the PageRank of a target vertex. The algorithm in [7] heuristically identifies the top contributors to a vertex v by adaptively choosing vertices with high likelihood of being large contributors, and then locally computes personalized PageRank from those vertices. This is different from our approach of directly computing the contribution vector, and more difficult to analyze rigorously.

Local algorithms have been studied in distributed computing [16] and in graph partitioning and clustering [20, 2]. Personalized PageRank vectors can be computed locally using a number of methods [5, 2, 19], many of which are based on the algorithm of Jeh and Widom [14]. None of these algorithms can be used directly to compute a contribution vector or supporting set.

There are numerous methods for detecting link spam besides the SpamRank-type algorithms we have mentioned here. Examples include applying machine learning to link-based features [3], the analysis of page content [15, 17], TrustRank [11] and Anti-TrustRank [18], and statistical analysis of various page features [8]. Finally, as a followup of this paper, we have used the local algorithm developed in this paper to design several locally computable link-spam features and report our experimental results for link spam detection [1].

Organization. This paper will be organized as follows. In Section 2, we review the basic concepts used in this paper, including PageRank, personalized PageRank, and PageRank contribution vectors. In Section 3, we derive an alternate formula for the PageRank contribution vector. Using this formula, we present an efficient local algorithm for computing PageRank contribution and analyze its performance. In Section 4, we consider several notions of supporting sets, which are sets of vertices that contribute significantly to the PageRank of a target vertex, and show how to efficiently compute approximate supporting sets. In Section 5 we make a few concluding remarks. We also show that, in principle, the time-reverse Markov chain can be used to compute the contribution vector, but argue that our method is more efficient.

2 Preliminaries

The web can be modeled by a directed graph $G = (V, E)$ where V are webpages and a directed edge $(u \to v) \in E$ represents a hyperlink in u that references v. Although the web graph is usually viewed as an unweighted graph, our discussion can be extended to weighted models. To deal with the problem of dangling nodes with no out-edges, we assume an artificial node with a single self-loop has been added to the graph, and an edge has been added from each dangling node to this artificial node. Let A denote the adjacency matrix of G. For each $u \in V$, let $d_{out}(u)$ denote the out-degree of u and let $d_{in}(u)$ denote the in-degree of u. Let D_{out} be the diagonal matrix of out-degrees.

We will now define PageRank vectors and contribution vectors. For convenience, we will view all vectors as row vectors, unless explicitly stated otherwise.

For a teleportation constant α, the PageRank vector $\mathbf{pr}_\alpha$ defined by Brin and Page [6] satisfies the following equation:

$$\mathbf{pr}_\alpha = \alpha \cdot \mathbf{1} + (1 - \alpha) \cdot \mathbf{pr}_\alpha \cdot M, \tag{1}$$

where M is the random walk transition matrix given by $M = D_{out}^{-1} A$ and $\mathbf{1}$ is the row vector of all 1's (always of proper size). The PageRank of a page u is then $\mathrm{pr}_\alpha(u)$. When there is no danger of confusion, we may drop the subscript α. Note that the above definition corresponds to the normalization $\sum_u \mathrm{pr}_\alpha(u) = |V|$.

Similarly, the personalized PageRank vector $\mathbf{ppr}(\alpha, u)$ of a page $u \in V$, defined by Haveliwala [13], satisfies the following equation.

$$\mathbf{ppr}(\alpha, u) = \alpha \cdot \mathbf{e}_u + (1 - \alpha) \cdot \mathbf{ppr}(\alpha, u) \cdot M, \tag{2}$$

where $\mathbf{e}_u$ is the row unit vector whose u^{th} entry is equal to 1.

Let $\mathbf{PRM}_\alpha$ denote the (personalized) PageRank matrix, whose uth row is the personalized PageRank vector $\mathbf{ppr}(\alpha, u)$. The (global) PageRank vector $\mathbf{pr}_\alpha$ is then $\mathbf{1} \cdot \mathbf{PRM}_\alpha$, the sum of all the personalized PageRank vectors. The *PageRank contribution* of u to v is defined to be the (u, v)th entry of $\mathbf{PRM}_\alpha$, and will be written $\mathrm{ppr}_\alpha(u \to v)$. The contribution vector $\mathbf{cpr}(\alpha, v)$ for the vertex v is defined to be the row vector whose transpose is the vth column of $\mathbf{PRM}_\alpha$. If $\mathbf{c} = \mathbf{cpr}(\alpha, v)$ is the contribution vector for v, then we denote by $\mathbf{c}(S)$ the total contribution of the vertices in S to the PageRank of v. In particular, we have $\mathbf{c}(V) = \mathrm{pr}_\alpha(v)$ and $\mathbf{c}(u) = \mathrm{ppr}_\alpha(u \to v)$.

3 Local Approximation of PageRank Contributions

In this section, we describe an algorithm for computing an approximation of the contribution vector $\mathbf{c} = \mathbf{cpr}(\alpha, v)$ of a vertex v.

Definition 1 (Approximate Contribution). *A vector $\tilde{\mathbf{c}}$ is an* ϵ-approximation *of the contribution vector $\mathbf{c} = \mathbf{cpr}(\alpha, v)$ if $\tilde{\mathbf{c}} \geq 0$ and, for all vertices u,*

$$\mathbf{c}(u) - \epsilon \cdot \mathrm{pr}_\alpha(v) \leq \tilde{\mathbf{c}}(u) \leq \mathbf{c}(u).$$

A vector $\tilde{\mathbf{c}}$ is an ϵ-absolute-approximation *of the contribution vector $\mathbf{c} = \mathbf{cpr}(\alpha, v)$ if $\tilde{\mathbf{c}} \geq 0$ and, for all vertices u,*

$$\mathbf{c}(u) - \epsilon \leq \tilde{\mathbf{c}}(u) \leq \mathbf{c}(u).$$

Clearly, an ϵ-approximation of $\mathbf{cpr}(\alpha, v)$ is an $(\epsilon \cdot \mathrm{pr}_\alpha(v))$-absolute-approximation of $\mathbf{cpr}(\alpha, v)$. In the algorithm below, we will focus on the computation of an ϵ-absolute-approximation of the contribution vector.

The *support* of a non-negative vector $\tilde{\mathbf{c}}$, denoted by $\mathrm{Supp}(\tilde{\mathbf{c}})$, is the set of all vertices whose entries in $\tilde{\mathbf{c}}$ are strictly positive. The vector $\mathbf{c}$ has a canonical ϵ-absolute-approximation. Let $\bar{\mathbf{c}}$ denote the vector

$$\bar{\mathbf{c}}(u) = \begin{cases} \mathbf{c}(u) \text{ if } \mathbf{c}(u) > \epsilon \\ 0 \quad \text{otherwise .} \end{cases}$$

Clearly, $\bar{\mathbf{c}}$ is the ϵ-absolute-approximation of $\mathbf{c}$ with the smallest support. Moreover, $\|\bar{\mathbf{c}}\|_1 \leq \|\mathbf{c}\|_1$ and thus, $|\mathrm{Supp}(\bar{\mathbf{c}})| \leq \|\mathbf{c}\|_1/\epsilon$. Our local algorithm attempts to find an approximation $\tilde{\mathbf{c}}$ of $\mathbf{c}$ which has a similar support structure to that of $\bar{\mathbf{c}}$.

3.1 High Level Idea of the Local Algorithm

It is well known that for each α, the personalized PageRank vector which satisfies Equation 2 also satisfies

$$\mathbf{ppr}(\alpha, u) = \alpha \sum_{t=0}^{\infty} (1 - \alpha)^t \cdot \left(\mathbf{e}_u M^t\right). \tag{3}$$

The contribution of u to v can then be written in the following way.

$$\mathrm{ppr}_\alpha(u \to v) = \langle\, \mathbf{ppr}(\alpha, u)\,,\, \mathbf{e}_v \,\rangle \tag{4}$$

$$= \left\langle\, \alpha \sum_{t=0}^{\infty} (1-\alpha)^t (\mathbf{e}_u M^t)\,,\, \mathbf{e}_v \,\right\rangle \tag{5}$$

$$= \left\langle\, \mathbf{e}_u\,,\, \alpha \sum_{t=0}^{\infty} (1-\alpha)^t (\mathbf{e}_v M^T)^t \,\right\rangle. \tag{6}$$

The standard way to compute the contribution of u to v is based on Equation 5. We refer to this approach as the time-forward calculation of $\mathrm{ppr}_\alpha(u \to v)$. Recall that $\mathbf{e}_u M^t$ is the t-step random walk distribution starting from u. In the time-forward calculation, we emulate the random walk from u step by step and add up the walk distributions scaled by the power sequence of $(1-\alpha)^t$. Without knowing in advance which vertices u make large contributions to v, one may have to perform the time-forward calculation of $\mathbf{ppr}(\alpha, u)$ for many vertices u to obtain a good approximation of $\mathbf{cpr}(\alpha, v)$.

To overcome this difficulty, we can directly calculate $\mathbf{cpr}(\alpha, v)$ in the manner suggested by Equation 6. This equation implies that

$$\mathbf{cpr}(\alpha, v) = \alpha \sum_{t=0}^{\infty} (1-\alpha)^t \cdot \left(\mathbf{e}_v (M^T)^t\right). \tag{7}$$

Thus, the contribution vector can be computed by starting with $\mathbf{e}_v$, iteratively computing $\mathbf{e}_v (M^T)^t$, and adding up the resulting vectors scaled by the power sequence of $(1-\alpha)^t$. Note that the matrix M^T is no longer a random walk matrix, since the sum of each row will not generally be equal to 1. Unlike the time-forward calculation, the direct calculation of $\mathbf{cpr}(\alpha, v)$ is no longer an emulation of the random walk starting from v. This fact complicates the error analysis of the next subsection.

The discussion above provides a way to directly compute $\mathbf{cpr}(\alpha, v)$, but our local algorithm will perform a different calculation. Instead of iteratively computing the vectors $\mathbf{e}_v (M^T)^t$, we adapt the technique of Jeh and Widom [14] for computing personalized PageRank to the task of computing contribution vectors. Using this method, we can compute the contribution vector in a decentralized way, and avoid spending computational effort manipulating small numerical values. This enables us to bound the running time required to obtain a fixed level of error.

Equation 7 also enables us to compute the vector of contributions to a specified subset S of vertices, which we define to be $\mathbf{cpr}(\alpha, S) = \sum_{v \in S} \mathbf{cpr}(\alpha, v)$. Let $\mathbf{e}_S = \sum_{v \in S} \mathbf{e}_v$. Then,

$$\mathbf{cpr}(\alpha, S) = \alpha \sum_{t=0}^{\infty} (1-\alpha)^t \cdot \left(\mathbf{e}_S (M^T)^t\right). \tag{8}$$

To further abuse notation, for any non-negative vector $\mathbf{s}$, we define

$$\mathbf{cpr}(\alpha, \mathbf{s}) = \alpha \sum_{t=0}^{\infty} (1-\alpha)^t \cdot \left(\mathbf{s} (M^T)^t\right). \tag{9}$$

3.2 The Local Algorithm and its Analysis

The theorem below describes our algorithm `ApproxContributions` for computing an ϵ-absolute-approximation of the contribution vector of a target vertex v. We give an upper bound on the number of vertices examined

by the algorithm that depends on $\mathrm{pr}_\alpha(v)$, ϵ, and α, but is otherwise independent of the number of vertices in the graph. The algorithm performs a sequence of operations, which we call pushback operations. Each pushback operation is performed on a single vertex of the graph, and requires time proportional to the in-degree of that vertex. We place an upper bound on the *number* of pushback operations performed by the algorithm, rather than the total running time of the algorithm. The total running time of the algorithm depends on the in-degrees of the sequence of vertices on which the pushback operations were performed. The number of pushback operations is an upper bound on the number of vertices in the support of the resulting approximate contribution vector.

Theorem 1. *The algorithm* `ApproxContributions`$(v, \alpha, \epsilon, \mathbf{p}_{max})$ *has the following properties. The input is a vertex v, two constants α and ϵ in the interval $(0, 1]$, and a real number $\mathbf{p}_{max}$. The algorithm computes a vector $\tilde{\mathbf{c}}$ such that $0 \leq \tilde{\mathbf{c}} \leq \mathbf{c}$, and either*

1. *$\tilde{\mathbf{c}}$ is an ϵ-absolute approximation of $\mathbf{cpr}(\alpha, v)$, or*
2. *$\|\tilde{\mathbf{c}}\|_1 \geq \mathbf{p}_{max}$.*

The number of pushback operations P performed by the algorithm satisfies the following bound,

$$P \leq \frac{\min\left(\mathrm{pr}_\alpha(v), \mathbf{p}_{max}\right)}{\alpha\epsilon} + 1.$$

The proof of Theorem 1 is based on a series of facts which we describe below. The starting point is the following observation, which is easy to verify from Equation 9. For any vector $\mathbf{s}$,

$$\mathbf{cpr}(\alpha, \mathbf{s})M^T = \mathbf{cpr}(\alpha, \mathbf{s}M^T). \tag{10}$$

We can further derive the following equation,

$$\begin{aligned} \mathbf{cpr}(\alpha, \mathbf{s}) &= \alpha\mathbf{s} + (1-\alpha) \cdot \mathbf{cpr}(\alpha, \mathbf{s})M^T \\ &= \alpha\mathbf{s} + (1-\alpha) \cdot \mathbf{cpr}(\alpha, \mathbf{s}M^T). \end{aligned} \tag{11}$$

This is the transposed version of the equation that was used Jeh and Widom to compute approximate personalized PageRank vectors [14]. Very naturally, we will use it to compute approximate contribution vectors.

The algorithm `ApproxContributions`$(v, \alpha, \epsilon, \mathbf{p}_{max})$ maintains a pair of vectors $\mathbf{p}$ and $\mathbf{r}$ with nonnegative entries, starting with the trivial approximation $\mathbf{p} = \mathbf{0}$ and $r = \mathbf{e}_v$, and applies a series of pushback operations that increase $\|\mathbf{p}\|_1$ while maintaining the invariant $\mathbf{p} + \mathbf{cpr}(\alpha, \mathbf{r}) = \mathbf{cpr}(\alpha, v)$. Each pushback operation picks a single vertex u, moves an α fraction of the mass at $\mathbf{r}(u)$ to $\mathbf{p}(u)$, and then modifies the vector $\mathbf{r}$ by replacing $\mathbf{r}(u)\mathbf{e}_u$ with $(1-\alpha)\mathbf{r}(u)\mathbf{e}_u M^T$. Note that $\|\mathbf{r}\|_1$ may increase or decrease during this operation. We will define the pushback operation more formally below, and then verify that each pushback operation does indeed maintain the invariant.

`pushback` (u):
Let $\mathbf{p}' = \mathbf{p}$ and $\mathbf{r}' = \mathbf{r}$, except for these changes:

1. $\mathbf{p}'(u) = \mathbf{p}(u) + \alpha\mathbf{r}(u)$.
2. $\mathbf{r}'(u) = 0$.
3. For each vertex w such that $w \to u$:
   $\mathbf{r}'(w) = \mathbf{r}(w) + (1-\alpha)\mathbf{r}(u)/d_{out}(w)$.

Lemma 1 (Invariant). *Let* $\mathbf{p}'$ *and* $\mathbf{r}'$ *be the result of performing* `pushback`(u) *on* $\mathbf{p}$ *and* $\mathbf{r}$*. If* $\mathbf{p}$ *and* $\mathbf{r}$ *satisfy the invariant* $\mathbf{p} + \mathbf{cpr}(\alpha, \mathbf{r}) = \mathbf{cpr}(\alpha, v)$*, then* $\mathbf{p}'$ *and* $\mathbf{r}'$ *satisfy the invariant* $\mathbf{p}' + \mathbf{cpr}(\alpha, \mathbf{r}') = \mathbf{cpr}(\alpha, v)$.

Proof. After the pushback operation, we have, in vector notation,

$$\begin{aligned}\mathbf{p}' &= \mathbf{p} + \alpha\mathbf{r}(u)\mathbf{e}_u.\\ \mathbf{r}' &= \mathbf{r} - \mathbf{r}(u)\mathbf{e}_u + (1-\alpha)\mathbf{r}(u)\mathbf{e}_u M^T.\end{aligned}$$

We will apply equation (11) to $\mathbf{r}(u)\mathbf{e}_u$ to show that $\mathbf{p} + \mathbf{cpr}(\alpha, \mathbf{r}) = p' + \mathbf{cpr}(\alpha, \mathbf{r}')$.

$$\begin{aligned}\mathbf{cpr}(\alpha, \mathbf{r}) &= \mathbf{cpr}(\alpha, \mathbf{r} - \mathbf{r}(u)\mathbf{e}_u) + \mathbf{cpr}(\alpha, \mathbf{r}(u)\mathbf{e}_u)\\ &= \mathbf{cpr}(\alpha, \mathbf{r} - \mathbf{r}(u)\mathbf{e}_u) + \alpha\mathbf{r}(u)\mathbf{e}_u + \mathbf{cpr}(\alpha, (1-\alpha)\mathbf{r}(u)\mathbf{e}_u M^T)\\ &= \mathbf{cpr}(\alpha, \mathbf{r} - \mathbf{r}(u)\mathbf{e}_u + (1-\alpha)\mathbf{r}(u)\mathbf{e}_u M^T) + \alpha\mathbf{r}(u)\mathbf{e}_u\\ &= \mathbf{cpr}(\alpha, \mathbf{r}') + \mathbf{p}' - \mathbf{p}.\end{aligned}$$

During each pushback operation, the quantity $\|\mathbf{p}\|_1$ increases by $\alpha\mathbf{r}(u)$. The quantity $\|\mathbf{p}\|_1$ can never exceed $\|\mathbf{cpr}(\alpha, v)\|_1$, which is equal to $\mathrm{pr}_\alpha(v)$. By performing pushback operations only on vertices where $\mathbf{r}(u) \geq \epsilon$, we can ensure that $\|\mathbf{p}\|_1$ increases by a significant amount at each step, which allows us to bound the number of pushes required to compute an ϵ-absolute-approximation of the contribution vector. This is the idea behind the algorithm `ApproxContributions`.

`ApproxContributions`$(v, \alpha, \epsilon, \mathbf{p}_{max})$:

1. Let $\mathbf{p} = \mathbf{0}$, and $\mathbf{r} = \mathbf{e}_v$.
2. While $\mathbf{r}(u) > \epsilon$ for some vertex u:
   (a) Pick any vertex u where $\mathbf{r}(u) \geq \epsilon$.
   (b) Apply `pushback` (u).
   (c) If $\|\mathbf{p}\|_1 \geq \mathbf{p}_{max}$, halt and output $\tilde{\mathbf{c}} = \mathbf{p}$.
3. Output $\tilde{\mathbf{c}} = \mathbf{p}$.

This algorithm can be implemented by maintaining a queue containing those vertices u satisfying $\mathbf{r}(u) \geq \epsilon$. Initially, v is the only vertex in the queue. At each step, we take the first vertex u in the queue, remove it from the queue, and perform a pushback operation from that vertex. If the pushback operation raises the value of $\mathbf{r}(x)$ above ϵ for some in-neighbor x of u, then x is added to the back of the queue. This continues until the queue is empty, at which point all vertices satisfy $\mathbf{r}(u) < \epsilon$, or until $\|\mathbf{p}\|_1 \geq \mathbf{p}_{max}$. We now show that this algorithm has the properties promised in Theorem 1.

*Proof (***Proof of Theorem 1).** Let T be the total number of push operations performed by the algorithm, and let $\mathbf{p}_t$ and $\mathbf{r}_t$ be the states of the vectors $\mathbf{p}$ and $\mathbf{r}$ after t pushes. The initial setting of $\mathbf{p}_0 = 0$ and $\mathbf{r}_0 = \mathbf{e}_v$ satisfies the invariant $\mathbf{p}_t + \mathbf{cpr}(\alpha, \mathbf{r}_t) = \mathbf{cpr}(\alpha, v)$, which is maintained throughout the algorithm. Since $\mathbf{r}_t$ is nonnegative at each step, the error term $\mathbf{cpr}(\alpha, \mathbf{r}_t)$ is also nonnegative, so we have $\mathbf{cpr}(\alpha, v) - \mathbf{p}_t \geq 0$. In particular, this implies $\|\mathbf{p}_t\|_1 \leq \|\mathbf{cpr}(\alpha, v)\|_1 = \mathrm{pr}_\alpha(v)$.

Let $\tilde{\mathbf{c}} = \mathbf{p}_T$ be the vector output by the algorithm. When the algorithm terminates, we must have either $\|\tilde{\mathbf{c}}\|_1 \geq \mathbf{p}_{max}$ or $\|\mathbf{r}_T\|_\infty \leq \epsilon$. In the latter case, the following calculation shows that $\tilde{\mathbf{c}}$ is an ϵ-absolute-approximation of $\mathbf{cpr}(\alpha, v)$.

$$\begin{aligned}\|\mathbf{cpr}(\alpha, v) - \tilde{\mathbf{c}}\|_\infty &= \|\mathbf{cpr}(\alpha, \mathbf{r}_T)\|_\infty\\ &\leq \|\mathbf{r}_T\|_\infty\\ &\leq \epsilon.\end{aligned}$$

The fact that $\|\mathbf{cpr}(\alpha, \mathbf{r}_T)\|_\infty \leq \|\mathbf{r}_T\|_\infty$ holds because $\mathbf{r}_T$ is nonnegative and each row of M sums to 1.

The vector $\mathbf{p}_{T-1}$ must have satisfied $\|\mathbf{p}_{T-1}\|_1 < \mathbf{p}_{max}$, since the algorithm decided to push one more time. We have already observed that $\|\mathbf{p}_{T-1}\|_1 \leq \mathrm{pr}_\alpha(v)$. Each push operation increased $\|\mathbf{p}\|_1$ by at least $\alpha\epsilon$, so we have

$$\alpha\epsilon(T-1) \leq \|\mathbf{p}_{T-1}\|_1 \leq \min\left(\|\mathbf{cpr}(\alpha, v)\|_1, \mathbf{p}_{max}\right).$$

This gives the desired bound on T.

It is possible to perform a pushback operation on the vertex u, and to perform the necessary queue updates, in time proportional to $d_{in}(u)$. Therefore, the running time of the algorithm is proportional to the sum over all pushback operations of the in-degree of the pushed vertex.

We can compute an ϵ-approximation of $\mathbf{cpr}(\alpha, v)$, provided that $\mathrm{pr}_\alpha(v)$ is known, by calling the algorithm `ApproxContributions`$(v, \alpha, \epsilon \cdot \mathrm{pr}_\alpha(v), \mathrm{pr}_\alpha(v))$.

Corollary 1 (ϵ-Approximation of contribution vectors). *Given $\mathrm{pr}_\alpha(v)$, an ϵ-approximation of $\mathbf{cpr}(\alpha, v)$, can be computed with $\frac{1}{\alpha\epsilon} + 1$ pushback operations.*

We also observe that, using Equation 8, our algorithm can be easily adapted to compute an ϵ-absolute-approximation and ϵ-approximation of $\mathbf{cpr}(\alpha, S)$ for a group S of vertices, with a similar bound on the number of pushback operations.

3.3 The Support of the Approximate Contribution Vector

The number of vertices in the support of the ϵ-approximate contribution vector $\tilde{\mathbf{c}}$ is upper bounded by the number of pushback operations used to compute it, which is at most $\frac{1}{\alpha\epsilon} + 1$. In this section we give a stronger upper bound on the size of the support. To do this, we need to modify the pushback operation slightly. Instead of moving all the mass from $\mathbf{r}(u)$ during the pushback operation, we move all but $\epsilon/2$ units of mass, and leave $\epsilon/2$ units on $\mathbf{r}(u)$. This increases the running time bound for the algorithm by a factor of 2, but ensures that $\mathbf{r}(x) \geq \epsilon/2$ at each vertex in $\mathrm{Supp}(\tilde{\mathbf{c}})$. We use this fact to give a family of bounds on the size of $\mathrm{Supp}(\tilde{\mathbf{c}})$.

We will abuse our notation a bit by defining the following,

$$\mathrm{pr}_\alpha(\mathbf{x} \to \mathbf{y}) = \langle\, \mathbf{x}M_\alpha\,,\, \mathbf{y}\,\rangle,$$

where $M_\alpha = \mathbf{PRM}_\alpha$ is the PageRank matrix. In particular, $\mathrm{pr}_\alpha(\mathbf{x} \to \mathbf{e}_S)$ is the amount probability from the PageRank vector with starting distribution $\mathbf{x}$ on the set S.

Proposition 1. *Let $\tilde{\mathbf{c}}$ be the ϵ-approximate contribution vector for v computed by the modified algorithm described above, and let $S = \mathrm{Supp}(\tilde{\mathbf{c}})$. For any nonnegative vector $\mathbf{z}$, we have the following upper bound on S,*

$$\mathrm{pr}_\alpha(\mathbf{z} \to \mathbf{e}_S) \leq \frac{2}{\epsilon}\mathrm{pr}_\alpha(\mathbf{z} \to \mathbf{e}_v).$$

Proof. Note that $\mathbf{ppr}(\alpha, v) = \mathbf{e}_v M_\alpha$ and $\mathbf{cpr}(\alpha, v) = \mathbf{e}_v M_\alpha^T$. We know that $\mathbf{cpr}(\alpha, \mathbf{r}) \leq \mathbf{cpr}(\alpha, \mathbf{e}_v)$, which can also be written $\mathbf{r}M_\alpha^T \leq \mathbf{e}_v M_\alpha^T$. Let $S = \mathrm{Supp}(\tilde{\mathbf{c}})$ and recall that $\mathbf{r}(x) \geq \epsilon/2$ for any vertex $x \in S$. Then,

$$\langle\, \mathbf{z}M_\alpha\,,\, \mathbf{e}_v\,\rangle = \left\langle\, \mathbf{z}\,,\, \mathbf{e}_v M_\alpha^T\,\right\rangle \geq \left\langle\, \mathbf{z}\,,\, \mathbf{r}M_\alpha^T\,\right\rangle = \langle\, \mathbf{z}M_\alpha\,,\, \mathbf{r}\,\rangle \geq (\epsilon/2)\langle\, \mathbf{z}M_\alpha\,,\, \mathbf{e}_S\,\rangle.$$

In the second step we needed $\mathbf{z}$ to be nonnegative, and in the last step we needed $\mathbf{z}M_\alpha$ to be nonnegative, which is true whenever $\mathbf{z}$ is nonnegative.

In words, this proposition states that for any starting vector $\mathbf{z}$, the amount of probability from the PageRank vector $\mathbf{ppr}(\alpha, \mathbf{z})$ on the set $S = \mathrm{Supp}(\tilde{\mathbf{c}})$ is at most $2/\epsilon$ times the amount on the vertex v. If we let $\mathbf{z} = \mathbf{e}_V$, then we obtain a bound on the amount of global PageRank on the set S,

$$\mathrm{pr}_\alpha(S) \leq \frac{2}{\epsilon}\mathrm{pr}_\alpha(v).$$

To see that this bound is at least as strong as what we knew before, recall that the PageRank of any given vertex is at least α. If we make the pessimistic assumption that $\mathrm{pr}_\alpha(u) = \alpha$ for each $u \in \mathrm{Supp}(\tilde{\mathbf{c}})$, then the bound we have just proved reduces to our earlier bound on the number of pushback operations,

$$|\mathrm{Supp}(\tilde{\mathbf{c}})| \leq 2\mathrm{pr}_\alpha(v)/\alpha\epsilon.$$

4 Computing Supporting Sets

In this section, we use our local algorithm for approximating contribution vectors to compute approximate supporting sets, sets of vertices that contribute significantly to the PageRank of a target vertex. There are several natural notions of supporting sets, which we define below. For a vertex v, let π_v be the permutation that orders the entries $\mathbf{cpr}(\alpha, v)$ from the largest to the smallest. Ties may be broken arbitrarily.

- **top k contributors**: the first k pages of π_v.
- **δ-significant contributors**: $\{u \mid \mathrm{ppr}_\alpha(u \to v) > \delta\}$.
- **ρ-supporting set**: a set S of pages such that

$$\mathrm{ppr}_\alpha(S \to v) \geq \rho \cdot \mathrm{pr}_\alpha(v).$$

In addition, let $k_\rho(v)$ be the smallest integer such that

$$\mathrm{ppr}_\alpha(\pi_v(1 : k_\rho(v)) \to v) \geq \rho \cdot \mathrm{pr}_\alpha(v).$$

Clearly the set of the first $k_\rho(v)$ pages of π_v is the minimum size ρ-supporting set for v. Also, we define $\rho_k(v) = \mathrm{ppr}_\alpha(\pi_v(1 : k) \to v)/\mathrm{pr}_\alpha(v)$ to be the fraction of v's PageRank contributed by its top k contributors.

4.1 Approximating Supporting Sets

Without precisely computing $\mathbf{cpr}(\alpha, v)$ it might be impossible to identify supporting sets exactly, so we consider approximate supporting sets. For a precision parameter ϵ, we define the following.

- **ϵ-precise top k contributors**: a set of k pages that contains all pages whose contribution to v is at least $\mathrm{ppr}_\alpha(\pi_v(k) \to v) + \epsilon \cdot \mathrm{pr}_\alpha(v)$, but no page with contribution to v less than $\mathrm{ppr}_\alpha(\pi_v(k) \to v) - \epsilon \cdot \mathrm{pr}_\alpha(v)$.
- **ϵ-precise δ-significant contributors**: a set that contains the set of δ-significant contributors and is contained in the set of $(\delta - \epsilon)$-significant contributors.

The results in the remainder of this section assume that $\mathrm{pr}_\alpha(v)$ is known.

Theorem 2. *An ϵ-precise set of top k contributors of a vertex v can be found by performing $1/\alpha\epsilon + 1$* `pushback` *operations.*

Proof. Call $\tilde{\mathbf{c}} = \texttt{ApproxContributions}(v, \alpha, \epsilon \cdot \mathrm{pr}_\alpha(v), \mathrm{pr}_\alpha(v))$. Let $C = \mathrm{Supp}(\tilde{\mathbf{c}})$. If $|C| > k$, then return the vertices with the top k entries in $\tilde{\mathbf{c}}$; otherwise, return C together with $k - \mathrm{Supp}(\tilde{\mathbf{c}})$ arbitrarily chosen vertices not in C. Consider a page u with $\mathbf{cpr}(u, v) \geq \mathbf{cpr}(\pi_v(k), v) + \epsilon \cdot \mathrm{pr}_\alpha(v)$. Clearly $u \in C$ because $\tilde{\mathbf{c}}(u) \geq \mathbf{cpr}(\pi_v(k), v)$, implying $\tilde{\mathbf{c}}(u)$ is among the top k entries in $\tilde{\mathbf{c}}$. On the other hand, $\tilde{\mathbf{c}}(\pi_v(j))$ is at least $\mathbf{cpr}(\pi_v(k), v) - \epsilon \cdot \mathrm{pr}_\alpha(v)$ for all $j \in [1 : k]$. Thus, each of the vertices with the top k entries in $\tilde{\mathbf{c}}$ must contribute at least $\mathbf{cpr}(\pi_v(k), v) - \epsilon \cdot \mathrm{pr}_\alpha(v)$ to v.

Theorem 3. *An ϵ-precise δ-significant contributing set of a vertex v can be found by performing $1/\alpha\epsilon + 1$* `pushback` *operations.*

Proof. Call $\tilde{\mathbf{c}} = \texttt{ApproxContributions}(v, \alpha, \epsilon \cdot \mathrm{pr}_\alpha(v), \mathrm{pr}_\alpha(v))$ and return the vertices whose entries in $\tilde{\mathbf{c}}$ are at least $(\delta - \epsilon) \cdot \mathrm{pr}_\alpha(v)$. Clearly, the set contains the δ-contributing set of v and is contained in the $(\delta - \epsilon)$-supporting set of v. Moreover, the number of pages not in the δ-supporting set that are included is at most $1/(\delta - \epsilon)$.

In the remainder of this section, we consider the computation of approximate ρ-supporting sets. We give two different algorithms, one for finding a supporting set on a fixed number of vertices with the largest contribution possible, and one for finding a supporting set with a fixed contribution on as few vertices as possible.

Theorem 4. *Given a vertex v and an integer k, a set of k vertices that is a $(\rho_k - \epsilon)$-supporting set for v can be found by performing $k/\alpha\epsilon + 1$* `pushback` *operations.*

Proof. Compute $\tilde{\mathbf{c}} = \texttt{ApproxContributions}(v, \alpha, \epsilon\mathrm{pr}_\alpha(v)/k, \mathrm{pr}_\alpha(v))$. Let S_k be the set of k top contributors to v, which are the k vertices with the highest values in $\mathbf{c}$, and let $\tilde{S}_k$ be the set of k vertices with the highest values in $\tilde{\mathbf{c}}$. The set $\tilde{S}_k$ meets the requirements of the theorem, since we have

$$\begin{aligned}\tilde{\mathbf{c}}(\tilde{S}_k) &\geq \mathbf{c}(S_k) - k(\epsilon\mathrm{pr}_\alpha(v)/k) \\ &\geq \rho_k \cdot \mathrm{pr}_\alpha(v) - \epsilon \cdot \mathrm{pr}_\alpha(v) \\ &= \mathrm{pr}_\alpha(v)(\rho_k - \epsilon).\end{aligned}$$

Theorem 5. *Assume we are given ρ but not k_ρ. A set of at most k_ρ vertices that is a $(\rho - \epsilon)$-supporting set for v can be found by performing $O(k_\rho \log k_\rho/\alpha\epsilon)$* `pushback` *operations.*

Proof. The challenge here is that we do not know k_ρ, so we need to use a binary search procedure to find a proxy for k_ρ. We will proceed in two phases. In the first phase, we guess a value of k, starting with $k = 1$, and compute $\tilde{\mathbf{c}} = \texttt{ApproxContributions}(v, \alpha, \epsilon \cdot \mathrm{pr}_\alpha(v)/k, \mathrm{pr}_\alpha(v))$. As in Theorem 4, let $\tilde{S}_k$ be the set of k vertices with the highest values in $\tilde{\mathbf{c}}$, which we know satisfies $\tilde{\mathbf{c}}(\tilde{S}_k) \geq (\rho_k - \epsilon)$. If we observe that $\tilde{\mathbf{c}}(\tilde{S}_k) < (\rho - \epsilon)$, then we double k and repeat the procedure. If we observe that $\tilde{\mathbf{c}}(\tilde{S}_k) \geq (\rho - \epsilon)$, then we halt and proceed to the second phase, and set k_1 to be the value of k for which this happens. We must have $k_1 \leq 2k_\rho$, since we are guaranteed to halt if $k \geq k_\rho$.

Let $k_0 = k_1/2$ be the value of k from the step before the first phase halted. In the second phase, we perform binary search within the interval $[k_0, k_1]$ to find the smallest integer k_{min} for which $\tilde{\mathbf{c}}(\tilde{S}_{k_{min}}) \geq (\rho - \epsilon)$, which must satisfy $k_{min} \leq k_\rho$. We output $\tilde{S}_{k_{min}}$.

Each time we call the subroutine $\tilde{\mathbf{c}} = \texttt{ApproxContributions}(v, \alpha, \epsilon\mathrm{pr}_\alpha(v)/k, \mathrm{pr}_\alpha(v))$, it requires $k/\alpha\epsilon + 1$ push operations. In the first phase we call this subroutine with a sequence of k values that double from 1 up to at most $2k_\rho$, so the number of push operations performed is $O(k_\rho/\alpha\epsilon + \log k_\rho)$. In the second phase, the binary search makes at most $\log k_\rho$ calls to the subroutine, with k set to at most $2k_\rho$ in each step, so the number of push operations performed is $O(k_\rho \log k_\rho/\alpha\epsilon + \log k_\rho)$. The total number of push operations performed in both phases is $O(k_\rho \log k_\rho/\alpha\epsilon)$.

4.2 Local Estimation of PageRank

Up to this point, we have assumed when computing the supporting set of a vertex that its PageRank is known. We now consider how to apply our approximate contribution algorithm when nothing is known about the PageRank of the target vertex. In particular, we consider the problem of computing a lower bound on the PageRank of a vertex using local computation.

A natural lower bound on the PageRank $\mathrm{pr}_\alpha(v)$ is provided by the contribution to v of its top k contributors, $p_k = \mathbf{cpr}(\pi_v(1:k), v)$. The theorem below shows we can efficiently certify that $\mathrm{pr}_\alpha(v)$ is approximately as large as p_k without prior knowledge of $\mathrm{pr}_\alpha(v)$ or p_k. This should be contrasted with the algorithms from the previous section, for which we needed to know the value $\mathrm{pr}_\alpha(v)$ in order to set ϵ to obtain the stated running times.

Theorem 6. *Given k and δ, we can compute a real number p such that*

$$p_k(1+\delta)^{-2} \leq p \leq \mathrm{pr}_\alpha(v),$$

where $p_k = \mathbf{cpr}(\pi_v(1:k), v)$, by performing $10k\log(k/\alpha\delta)/\alpha$ `pushback` operations.

Proof. Fix k and δ, choose a value of p, and compute $\tilde{\mathbf{c}} = \texttt{ApproxContributions}(v, \alpha, \epsilon, p)$ with $\epsilon = \delta p/k$. The number of `pushback` operations performed is at most

$$1 + p/\alpha\epsilon = 1 + p/\alpha(\delta p/k) = 1 + 10k/\alpha.$$

When the algorithm halts, we either have $\|\tilde{\mathbf{c}}\|_1 \geq p$, in which case we have certified that $\mathrm{pr}_\alpha(v) \geq p$, or else we have $\|\tilde{\mathbf{c}} - \mathbf{cpr}(\alpha, v)\|_\infty \leq \delta p/k$, in which case we have certified that $p_k \leq (1+\delta)p$, by the following calculation:

$$p_k = \mathbf{cpr}(\pi_v(1:k), v) \leq \tilde{\mathbf{c}}(\pi_v(1:k), v) + (\delta p/k)k \leq p + \delta p.$$

We now perform binary search over p in the range $[\alpha, k]$. Let p_{low} be the largest value of p for which we have certified that $\mathrm{pr}_\alpha(v) \geq p$, and let p_{high} be the smallest value of p for which we have certified that $p_k \leq (1+\delta)p$. We perform binary search until $p_{high} \leq p_{low}(1+\delta)$, which requires at most $\log(k/\alpha\delta)$ steps. Then, p_{low} has the property described in the theorem,

$$\mathrm{pr}_\alpha(v) \geq p_{low} \geq p_{high}(1+\delta)^{-1} \geq p_k(1+\delta)^{-2}.$$

The total number of `pushback` operations performed during the calls to `ApproxContributions` during the binary search is at most $10k\log(k/\alpha\delta)/\alpha$.

5 Final Remarks

5.1 Improving the Dependency on In-Degrees

In our performance analysis, we give a bound of $\mathrm{pr}_\alpha(v)/(\alpha\epsilon)+1$ on the total number of `pushback` operations performed by our algorithm. In a pushback at a vertex u, we update the entry for u in the vector $\mathbf{p}$ as well the as entries in $\mathbf{r}$ for all vertices that point to u. As a result, the overall time complexity of our algorithm is proportional to the sum of the in-degrees of the sequence of vertices that we pushback from. A possible direction for future research is to devise an algorithm whose running time can be bounded in terms of the total in-degree of the supporting set that the algorithm attempts to approximate. This type of bound would offer stronger control over the running time than the result obtained in this paper, where the number of pushback operations operations is bounded in terms of the number of vertices in the supporting set, but the running time depends on the in-degrees of the vertices from which the sequence of push operations is performed.

5.2 Computing Contribution Vectors via the Time-Reverse Chain

As noted earlier, the matrix M^T in the formula of Equation 7 may not be Markov. It is natural ask whether the time-reverse Markov chain of the random walk matrix M may be used to compute the contribution vector for a vertex v, and, if so, whether this method is efficient.

For the following discussion, we assume that M has a unique stationary distribution, which will not be true for general directed graphs. Recall that,

Definition 2 (Time-reverse chain). *Given a Markov chain M with transition probability m_{ij}, and stationary distribution π, the time-reverse chain is the Markov chain R with transition probability $r_{ij} = \pi(j)m_{ji}/\pi(i)$.*

In other words, let $\mathit{\Pi}$ be the matrix whose (i,j)th entry is $\pi(j)/\pi(i)$, then $R = \mathit{\Pi} \cdot *M^T$, where the operation $\cdot *$ is the component-wise multiplication of two matrices. The time-reverse chain has the following properties.

- R has the same stationary distribution as M,
- for all i, k, and t, consider the t-step random walk starting from i in M and k in R, then

$$\left\langle \mathbf{e}_i M^t \, , \, \mathbf{e}_k \right\rangle = \left(\frac{\pi(k)}{\pi(i)} \right) \left\langle \mathbf{e}_k R^t \, , \, \mathbf{e}_i \right\rangle \tag{12}$$

Recall $\left\langle \mathbf{e}_i M^t \, , \, \mathbf{e}_k \right\rangle$ is equal to the probability that k is the vertex reached by a t-step random walk from i. Let $\mathrm{ppr}_\alpha^M(u \to v)$ denote the personalized PageRank contribution from u to v in a Markov chain M.

Theorem 7. *Suppose a Markov chain M has a stationary distribution π and R is its time-reverse chain. Then*

$$\mathrm{ppr}_\alpha^M(u \to v) = \left(\frac{\pi(v)}{\pi(u)} \right) \mathrm{ppr}_\alpha^R(v \to u). \tag{13}$$

Proof. The result follows from Equations 5 and 12.

Thus, if the stationary distribution exists, we can in principle compute the contribution vector of M by computing the personalized PageRank vector for v in the time-reverse chain. We argue that the method we presented in Section 3 is preferable to the time-reverse Markov chain method for the following reasons. Our method does not require that M has a stationary distribution. Computing a personalized PageRank vector in the time-reverse Markov chain requires that we first compute the stationary distribution π of M, which may be computationally expensive. Perhaps most important is the difference in the error analysis. If the stationary distribution exists, one can compute an ϵ-approximate contribution vector by computing a personalized PageRank vector in R for which the error at each vertex i is at most $\epsilon\pi(i)$. If $\pi(i)$ is extremely small at some vertices, and it may be exponentially small in the number of vertices in the graph, this will require a large amount of computation.

We prefer the method presented in Section 3 to the time-reverse method for most graphs that are likely to be encountered in practice. However, there are special cases where the time-reverse method will be efficient. In particular, if the Markov chain has a stationary distribution that is nearly proportional to the in-degrees of the vertices, as it would be in an undirected graph, then computing a personalized PageRank vector in the time-reverse chain is an efficient way to compute a contribution vector.

References

1. R. Andersen, C. Borgs, J. Chayes, J. Hopcroft, K. Jain, V. Mirrokni, and S. Teng. Experimental evaluation of locally computable link-spam features. Submitted, 2007.
2. R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *FOCS '06: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 475–486, Washington, DC, USA, 2006. IEEE Computer Society.
3. L. Becchetti, C. Castillo, D. Donato, S. Leonardi, and R. Baeza-Yates. Link-based characterization and detection of web spam, 2006.
4. A. A. Benczúr, K. Csalogány, T. Sarlós, and M. Uher. Spamrank - fully automatic link spam detection. In *First International Workshop on Adversarial Information Retrieval on the Web*, 2005.
5. P. Berkhin. Bookmark-coloring algorithm for personalized pagerank computing. *Internet Math.*, 3(1):41–62, 2006.
6. S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.
7. Y. Chen, Q. Gan, and T. Suel. Local methods for estimating pagerank values. In *Proc. of CIKM*, pages 381–389, 2004.
8. D. Fetterly, M. Manasse, and M. Najork. Spam, damn spam, and statistics: using statistical analysis to locate spam web pages. In *WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases*, pages 1–6, New York, NY, USA, 2004. ACM Press.
9. D. Fogaras and B. Racz. Towards scaling fully personalized pagerank. In *Proceedings of the 3rd Workshop on Algorithms and Models for the Web-Graph (WAW)*, pages 105–117, October 2004.
10. Z. Gyöngyi, P. Berkhin, H. Garcia-Molina, and J. Pedersen. Link spam detection based on mass estimation. In *Proceedings of the 32nd International Conference on Very Large Databases*. ACM, 2006.
11. Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating web spam with trustrank. In *VLDB*, pages 576–587, 2004.
12. Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Web content categorization using link information. Technical report, Stanford University, 2006.
13. T. H. Haveliwala. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE Trans. Knowl. Data Eng.*, 15(4):784–796, 2003.
14. G. Jeh and J. Widom. Scaling personalized web search. In *Proceedings of the 12th World Wide Web Conference (WWW)*, pages 271–279, 2003.
15. G. Mishne and D. Carmel. Blocking blog spam with language model disagreement, 2005.
16. M. Naor and L. Stockmeyer. What can be computed locally? *SIAM J. Comput.*, 24(6):1259–1277, 1995.
17. A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly. Detecting spam web pages through content analysis. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 83–92, New York, NY, USA, 2006. ACM Press.
18. R. Raj and V. Krishnan. Web spam detection with anti-trust rank. In *Proc. of the 2nd International Worshop on Adversarial Information Retreival on the Web*, pages 381–389, 2006.
19. T. Sarlós, A. A. Benczúr, K. Csalogány, D. Fogaras, and B. Rácz. To randomize or not to randomize: space optimal summaries for hyperlink analysis. In *WWW*, pages 297–306, 2006.
20. D. A. Spielman and S.-H. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *ACM STOC-04*, pages 81–90, New York, NY, USA, 2004. ACM Press.