Estimating PageRank on Graph Streams

ATISH DAS SARMA, Georgia Institute of Technology
SREENIVAS GOLLAPUDI and RINA PANIGRAHY, Microsoft Research

This article focuses on computations on large graphs (e.g., the web-graph) where the edges of the graph are presented as a stream. The objective in the streaming model is to use small amount of memory (preferably sub-linear in the number of nodes n) and a smaller number of passes.

In the streaming model, we show how to perform several graph computations including estimating the probability distribution after a random walk of length l, the mixing time M, and other related quantities such as the conductance of the graph. By applying our algorithm for computing probability distribution on the web-graph, we can estimate the *PageRank* p of any node up to an additive error of $\sqrt{\epsilon p} + \epsilon$ in $\tilde{O}(\sqrt{M/\alpha})$ passes and $\tilde{O}(\min(n\alpha + 1/\epsilon\sqrt{M/\alpha} + (1/\epsilon)M\alpha, \alpha n\sqrt{M\alpha} + (1/\epsilon)\sqrt{M/\alpha}))$ space, for any $\alpha \in (0, 1]$. Specifically, for $\epsilon = M/n$, $\alpha = M^{-\frac{1}{2}}$, we can compute the approximate PageRank values in $\tilde{O}(nM^{-\frac{1}{4}})$ space and $\tilde{O}(M^{\frac{3}{4}})$ passes. In comparison, a standard implementation of the PageRank algorithm will take $O(n)$ space and $O(M)$ passes. We also give an approach to approximate the PageRank values in just $\tilde{O}(1)$ passes although this requires $\tilde{O}(nM)$ space.

Categories and Subject Descriptors: F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Algorithms, graph conductance, mixing time, PageRank, random walk, streaming algorithms

ACM Reference Format:
Das Sarma, A., Gollapudi, S., and Panigrahy, R. 2011. Estimating PageRank on graph streams. J. ACM 58, 3, Article 13 (May 2011), 19 pages.
DOI = 10.1145/1970392.1970397 http://doi.acm.org/10.1145/1970392.1970397

1. INTRODUCTION

Real-world networks such as the web and social networks can be modeled as large graphs. Other instances of large graphs include click graphs generated from search engine query logs, document-term graphs computed from large collection of documents etc. The scale of these graphs has greatly increased the need for efficient algorithms to process them. While the graphs can be stored in secondary storage, processing of such data is usually performed using physical memory (RAM), which is a limited resource. The streaming model admits an ideal approach to processing such data. In this model, the data is presented as a stream and any computation on the stream relies on using a small amount of memory. Many streaming algorithms exist for computing frequency moments (with matching lower bounds), quantiles, and norms [Alon et al. 1999;

Part of this work was done while A. Das Sarma was an intern at Microsoft Search Labs, Microsoft Research. The author's current affiliation is Google Research.
Authors' present addresses: A. Das Sarma, Google Research, 1600 Amphitheatre Parkway, Mountain View, CA 94041; email: atish.dassarma@gmail.com; S. Gollapudi and R. Panigrahy, Microsoft Research, 1065 La Avenida, Mountain View, CA 94043; email:{sreenig, rina}@microsoft.com.

DOI 10.1145/1970392.1970397 http://doi.acm.org/10.1145/1970392.1970397

Indyk and Woodruff 2003; Bhuvanagiri et al. 2006; Bhuvanagiri and Ganguly 2006; Manku et al. 1999; Greenwald and Khanna 2001; Guha and McGregor 2006], and computations over graphs including counting triangles, properties of degree sequences, and connectivity [Bar-Yossef et al. 2002; Cormode and Muthukrishnan 2005; Demetrescu et al. 2006; Feigenbaum et al. 2005; Jowhari and Ghodsi 2005; McGregor 2005]. These graphs described here readily admit a streaming model. In this article, we compute the PageRank [Brin and Page 1998] of a large graph presented as a stream of edges in no particular order; neither is it required that all the edges incident on a vertex be grouped together in the stream. Moreover, they also admit link-based ranking algorithms like the PageRank algorithm to compute the relative importance of nodes in the graph.

While the basic requirements of streaming algorithms include small space and a small number of passes, these quantities can vary significantly from algorithm to algorithm. After Henzinger et al. [1999] showed linear lower bounds on the "space $\times$ passes" product for several problems including connectivity and shortest path problems, the work of Feigenbaum et al. [2005] studied the value of space in computing graph distances in the streaming model. Specifically, they studied approximate diameter and girth estimation. Their techniques are based on a single-pass randomized algorithm for constructing a $(2t+1)$-spanner. Demetrescu et al. [2006] proposed streaming graph algorithms using sublinear space and passes to compute single-source shortest paths on directed graphs and s-t connectivity on undirected graphs. In this article, we propose algorithms that require sublinear space and passes to compute the approximate PageRank values of nodes in a large directed graph.

We now give a brief description of PageRank. Given a web-graph representing the web pages and links between them, PageRank [Brin and Page 1998] computes a query-independent score for the web pages, taking into account endorsement of a web page by other web pages. The endorsement of a web page is computed from the in-links pointing to the page from other web pages. Alternately, the PageRank algorithm can also be viewed as computing the probability distribution of a random surfer visiting a page on the web. In general, the PageRank of a page u is dependent on the PageRank of all pages v that link to u as $PR(u) = \sum_{(v,u)\in E} PR(v)/d(v)$ where $d(.)$ denotes the out-degree. A standard implementation of the algorithm requires several iterations, say M, before the values converge. Alternately, this process can be viewed as a random walk requiring M steps. The typical length is about 200. In fact, the random walk is performed on a slightly modified graph that captures a random reset step in the PageRank algorithm. This step was introduced to model the random jump of a surfer from a page with no out-links to another page on the web and it also improves the convergence time of the algorithm. Finally, we note that in a PageRank computation, the nodes with large PageRank are of interest.

Besides PageRank, other graph properties of interest include connectedness, conductance, mixing time, and the sparsest cut. It is well known and was first shown by Jerrum and Sinclair [1989] that the mixing time M and conductance ϕ are related as $\phi^2/2 \leq \frac{1}{M} \leq 2\phi$.[1]

1.1. Contributions of This Article

A random walk of length l can be modeled as a matrix-vector computation $v = u.A^l$, where u (a row vector) is the initial distribution on the nodes, A is the transition matrix that corresponds to a single step in the random walk on the underlying graph, and v is the final distribution after performing the walk. The problem of computing

[1]Note that mixing time is well-defined only for *aperiodic* graphs. Bipartite graphs, for example, are not aperiodic. On the other hand, the bipartiteness of a graph can be checked in a single pass (see e.g., Feigenbaum et al. [2005]).

a single destination of a random walk of length l starting from a node picked from the distribution u is same as sampling a node from the distribution v. So, by simply maintaining an array of size n that represents the probability distribution, we can compute v in l passes and $O(n)$ space. Thus, a standard implementation of the PageRank algorithm, that computes the stationary distribution, will require M passes and $O(n)$ space, where M is the mixing time. In comparison, our work requires $\tilde{O}(\sqrt{M})$ passes and $o(n)$ space to compute the PageRank of nodes with values greater than M/n. Performing random walks of length M, the mixing time, of course, requires the knowledge of the mixing time. We therefore also provide an algorithm to estimate the mixing time. In this article we provide algorithms on a graph stream for the following problems:

—Running a single random walk of length l in $\tilde{O}(\sqrt{l})$ passes. In fact, we show how to perform n/l independent random walks using space sublinear in n and passes sublinear in l using the single random walk result as a subroutine;
—Using the sampled random walks obtained in the previous result, we show how to compute an approximate probability distribution (the PageRank vector) of nodes after a random walk of length l;
—Finally, using probability distributions of random walks for different lengths, we show how to estimate the mixing time, which in turn gives an estimate of the conductance of the graph.

For all these results, the main goal is to use as few passes over the stream as possible, while using space sub-linear in the number of nodes in the graph. Notice that for a dense graph (with number of edges being $\Omega(n^2)$), the space used by our algorithms are asymptotically less than square-root of the length of the stream (i.e., square-root of the number of edges). To compute the probability distribution over nodes after a random walk of length l, a naïve algorithm uses l passes and $O(n)$ space by performing l matrix-vector multiplications. We show how to approximate the same distribution. For any node with probability p in the distribution, we can approximate p within $p \pm \sqrt{\epsilon p} \pm \epsilon$ in $\tilde{O}(\sqrt{l/\alpha})$ passes and

$$\tilde{O}\left(\min\left(n\alpha + \frac{1}{\epsilon}\sqrt{\frac{l}{\alpha}} + \frac{1}{\epsilon}l\alpha, \alpha n\sqrt{l\alpha} + \frac{1}{\epsilon}\sqrt{\frac{l}{\alpha}}\right)\right)$$

space. Note that approximating p within $p \pm \sqrt{\epsilon p} \pm \epsilon$ can also be viewed as a $1 \pm \sqrt{\epsilon/p} \pm \epsilon/p$ approximation ratio which is close to 1 for p much larger than ϵ. This means we can estimate the probability value with high accuracy for nodes with large probability. Note that in the context of PageRank computation, we set l to the mixing time M of the random walk. For concreteness, this means we can estimate the PageRank p of a node within $p \pm \sqrt{\epsilon p} \pm \epsilon$ in $\tilde{O}(nM^{-\frac{1}{4}})$ space and $\tilde{O}(M^{\frac{3}{4}})$ passes for $\epsilon = M/n$. We also show how to find what we call the ϵ-near mixing time, that is, the time taken for the probability distribution to reach within ϵ of the steady state distribution under the L_1-norm. This automatically gives a result for estimating the *conductance* of the graph.

1.2. Related Work

Data streaming algorithms became popular since the famous result of Alon et al. [1999] on approximating frequency moments. There have been a surge of papers looking at various problems in the streaming setting. In particular, there has been significant attention to computing various frequency moments as they provide important statistics about the data. Tight results are known for computing some of them while others remain open [Bhuvanagiri et al. 2006; Indyk and Woodruff 2003]. Recent streaming results include estimating earth-movers distance [Indyk 2004],

communication complexity [Feldman et al. 2006; Woodruff 2004], counting triangles in graphs [Bar-Yossef et al. 2002; Jowhari and Ghodsi 2005; Buriol et al. 2006], quantile estimation [Guha and McGregor 2006, 2007b], sampling and entropy information [Guha et al. 2006; Guha and McGregor 2007a], and graph matchings [McGregor 2005].

This is by no means a comprehensive summary of all the results on data streams. There are many more studies on streaming problems including filtering irrelevant data from the stream, low rank approximation and fast multiplication of matrices, characterizing sketchable distances, scheduling problems, work on dynamic geometric problems, generating histograms, and finding increasing subsequences.

In comparison to the work on aggregating statistics of a data stream, the work on graphs as data streams is limited. Demetrescu et al. [2006] show space-pass trade-offs for shortest path problems in graph streams. In general, it seems hard to approximate many properties on graphs while maintaining sub-linear space in the number of vertices in the graph, by performing only a constant passes over the stream.

A very interesting piece of work is due to Sarlos et al. [2006] who give approaches to finding summaries for hyperlink analysis and personalized PageRank computation. Their study is not under the data streams setting; rather, they use sketching techniques and construct simple deterministic summaries that are later used by the algorithms for computing the PageRank vectors. They give lower bounds to prove that the space required by their algorithms is *optimal* under their setting. Given an additive error of ϵ and the probability δ of an incorrect result, their disk usage bound is $O(n \log(1/\delta)/\epsilon)$.

There has also been work on *s-t* connectivity [Feige 1997; Armoni et al. 1997] not under the streams setting; however it is not clear that extending to the streaming model is possible. Recent work by Wicks and Greenwald [2007] shows an interesting approach to parallelizing the computation of PageRank. McSherry [2005] exploits the link structure of the web graph within and across domains to accelerate the computation of PageRank. Andersen et al. [2006] computes local cuts near a specified vertex using personalized PageRank vectors.

The main ingredient in all our algorithms is to perform random walks efficiently. We begin by presenting the algorithm for running one random walk of length l in a small number of passes in Section 3. The main idea in this algorithm is to sample each of the n nodes independently with probability α and perform short (length w) random walks from each of the sampled nodes in w passes. We then try to *merge* these walks to form longer random walks from the source. The main idea in estimating the probability distribution or mixing time is running several such random walks. However, running several random walks requires some additional ideas to ensure small space. We describe how to efficiently run a large number of random walks in Section 4. This also gives an algorithm for approximating the probability distribution after a random walk. The algorithm for approximating the mixing time uses these ideas and is described in Sections 5. Section 6 provides an alternate algorithm for estimating the probability distribution with higher accuracy more efficiently for certain values of ϵ.

2. PRIMER ON RANDOM WALKS AND PAGERANK

Random Walks. Given a weighted directed graph G, the random walk process is defined as follows: The probability of transitioning from a node i to a node j, for $i, j \in V(G)$ and $(i, j) \in E(G)$ is equal to $w_{ij}/\sum_{(i,k)\in E(G)} w_{ik}$ where w_{ij} is the nonnegative weight on the edge (i, j). For $(i, j) \notin E(G)$, the probability of transitioning from i to j is 0. So, the probability of transitioning from a node to a neighbor is proportional to the weight of the corresponding edge. For undirected graphs, the weight is equal in both directions.

The random walk process can be defined for undirected graphs as well, where the sum above is taken over all undirected edges $(i, k) \in E(G)$. For unweighted graphs, the probability of transitioning from a vertex i to any neighbor is the same, and is equal to $1/d(i)$ where $d(i)$ is the degree of the vertex i in G.

Transition Matrix. The random walk transition probabilities on a graph is often expressed in the form a matrix, called the transition matrix A. The (i, j)-th entry in the matrix denotes the probability of transition from i to j by the random walk process. The transition matrix A can easily be obtained from the adjacency matrix of the graph. Every row sum of the matrix is 1.

Therefore, for edges (i, j) in the graph we have,

$$A(i, j) = \frac{w_{ij}}{\sum_{(i,k)\in E(G)} w_{ik}}$$

For all other pairs (i, j), $A(i, j) = 0$. Starting from a probability distribution u (row-vector) over the nodes, the distribution after l steps of the random walk can be computed as $u.A^l$.

Steady State Distribution. The steady state distribution of a random walk process is defined as the distribution that the walk approaches, as the number of steps in the walk goes to infinity. The steady state distribution vector v satisfies $v = v.A$. The steady state vector is uniquely determined under some basic conditions [Jerrum and Sinclair 1989] such as the graph being connected (or strongly connected in the case of directed graphs) and aperiodic. For undirected graphs, the vector v is determined by the degree distribution, and for node i, v_i is equal to the $\frac{deg(i)}{2m}$ where $deg(i)$ is the degree of node i and m is the total number of edges.

PageRank. In real world graph applications, the PageRank vector is the steady state distribution of a random walk that is a slight modification of the standard random walk over the graph. At any step, the random walk process defined previously is followed with probability $\beta \leq 1$. With probability $1 - \beta$, node i transitions to a randomly chosen node. In particular, the transition matrix with random resets incorporated is as follows:

$$A(i, j) = \beta \cdot \frac{w_{ij}}{\sum_{(i,k)\in E(G)} w_{ik}} + (1 - \beta)\frac{1}{n}$$

Alternatively, one can think of this as modifying the graph by scaling all current weights by a factor of β, and adding a complete graph scaled to a weight of $1 - \beta$. This way, each node transitions to a randomly chosen node with probably $1 - \beta$, and transitions based on the underlying graph with probability β. Here, n is the number of vertices in the graph. One of the reasons these random reset edges are incorporated is to force the graph to become strongly connected. This way, the PageRank vector becomes independent of the initial distribution. Since these edges can be implicitly assumed to be appearing in the stream by the algorithm, here after whenever we mention a graph or a random walk, we assume that it is over this modified graph with random reset edges included.

We will provide a streaming algorithm to estimate the steady state distribution of a random walk on a graph; since the PageRank vector is the steady state distribution of a random walk on a slightly altered graph, we can still use our algorithm by implicitly handling the edges corresponding to the random resets at run time. With every stream, the algorithm can pretend that the edges corresponding to the random resets are also presented, and the algorithm handles them similarly.

Mixing Time and Conductance. The mixing time of a graph is the number of steps required for the random walk process, starting at any distribution, to reach close to

the steady state distribution. Therefore, if M is the mixing time, then for any starting distribution u, the transition matrix A of the graph satisfies $u.A^M$ is close to $u.A^{M+1}$. Formally, the mixing time is defined as the smallest M such that the variation distance $|u.A^M - u.A^{M+1}|_1$ is at most ϵ for all u. Here ϵ is normally chosen to be a constant $1/2e$.

The conductance of a graph G is defined as $\phi(G) = \min_{S:|S|\leq|V|/2}(E(S, V(G) \setminus S)/E(S))$ where $E(S, V(G)\setminus S)$ is the number of the edges spanning the cut $(S, V(G)\setminus S)$ and $E(S)$ is the number of edges on the smaller side of the cut. Conductance is a good measure of how separable a graph is into two pieces. The conductance Φ of a graph and the mixing time M are related by $\Theta(1/M) \leq \Phi \leq \Theta(1/\sqrt{M})$ as shown in Jerrum and Sinclair [1989].

3. SINGLE RANDOM WALK

We first present an algorithm to perform a single random walk over a graph stream efficiently. The naïve approach is to do this in $O(1)$ space and l passes by performing one step of the random walk with every pass over the stream. At the other extreme, one can perform a random walk of length l in 1 pass and $O(nl)$ space by sampling l edges out of each of the n nodes in one pass. Subsequently, with these nl edges stored, it is possible to perform a random walk of length l without any more passes, as with l edges out of each node, the random walk cannot get stuck at a node before completing a walk of length l. In this section, we show the following result.

THEOREM 3.1. *One can perform a random walk of length l in $\tilde{O}(\sqrt{l/\alpha})$ passes* with high probability[2] *and $\tilde{O}(n\alpha + \sqrt{l/\alpha})$ space, for any choice of α with $0 < \alpha \leq 1$.*

Setting $\alpha = 1$, we get the following corollary.

COROLLARY 3.2. *One can perform a random walk of length l in $\tilde{O}(\sqrt{l})$ passes and $\tilde{O}(n)$ space.*

We start by describing the overall approach of our algorithm.

Perform Short Random Walks Out of Sampled Nodes. The main idea in our algorithm is to sample each node with probability α independently and perform short random walks of length w from each sampled node; this is done in w passes over the stream. The algorithm tries to extend the walk from the source by *merging* these short walks to form a longer random walk. It may get stuck in one of two ways. First, the walk may end up at a node that has not been sampled. Second, the walk may end up at one of the sampled nodes for a second time; notice that its stored w length walk cannot be used more than once in order to preserve the randomness of the walk. Note that sampling each node with probability α can be done by using a pseudo-random hash function on the node id.

Handling Stuck Nodes. While constructing the walk if it gets stuck at a node, from which no unused w-length walk is available, we will refer to such a node as a *stuck* node. We handle stuck nodes as follows. We keep track of the set S of sampled nodes whose w length walks have already been used in extending the random walk so far. In one additional pass, we now sample s edges out of the stuck node as well as each node in S. (If the degree of a node is present in the stream it can be done in one pass; otherwise, one can do this in two passes, one to compute its degree and another to sample the edges.) We then extend the walk as far as possible using these newly sampled edges. If the new endpoint is a sampled node whose w-length walk has not been used (i.e., it is not in S), then we continue merging as before. Otherwise, if the new end-point is a

[2]Also abbreviated as w.h.p. The probability is at least $1 - (1/poly(nl))$ if we include $\log(nl)$ factors in the $\tilde{O}$. All our theorems hold w.h.p.

ALGORITHM 1: SINGLERANDOMWALK(u, l)

Input: Starting node u, and desired walk length l.
Output: $\mathcal{L}_u$ the random walk from u of length l.
$T \leftarrow$ set of nodes obtained by sampling each node independently with probability α (in one pass).
In w passes, perform walks of length w from every node in T. Let $W[t] \leftarrow$ the end point of the walk of length w from $t \in T$ (the nodes in T whose w length walks get used towards $\mathcal{L}_u$ will get included in S).
$S \leftarrow \{\}$(we will refer to the nodes in S as *centers*).
Initialize $\mathcal{L}_u$ to a zero length walk starting at u. Let $x \leftarrow u$.
while $|\mathcal{L}_u| < l$ **do**
(1) if ($x \in T$ and $x \notin S$) extend $\mathcal{L}_u$ by appending the walk (implicit in) $W[x]$. $S \leftarrow S \cup \{x\}$. $x \leftarrow W[x]$, the new end point of $\mathcal{L}_u$. {this means we have a w length walk starting at x that has not been used so far in $\mathcal{L}_u$}
(2) if ($x \notin T$ or $x \in S$) HANDLESTUCKNODE(x, T, S, $\mathcal{L}_u$, l). {this means that either x was not in the initial set of sampled nodes, or x's w-length walk has already been used up}
end while

ALGORITHM 2: HANDLESTUCKNODE(x, T, S, $\mathcal{L}_u$, l)

$R \leftarrow x$.
while $|\mathcal{L}_u| < l$ **do**
  $E \leftarrow$ sample s edges (with repetition) out of each node in $S \cup R$.
  Extend $\mathcal{L}_u$ as far as possible by walking along the sampled edges in E (on visiting a node in $S \cup R$ for the k-th time, use the kth edge of the s sampled edges from that node).
  $x \leftarrow$ new end point of $\mathcal{L}_u$ after the extension. One of the following cases arise.
(1) if ($x \in S \cup R$) **continue** {no new node is seen, at least s progress has been made.}
(2) if ($x \in T$ and $x \notin S \cup R$) **return** {this means that x is a node that has not been seen in the walk so far, and x was among the set of nodes sampled initially; therefore, the w-length walk from x has not been used}
(3) if ($x \notin T$ and $x \notin S \cup R$) $R \leftarrow R \cup \{x\}$. {this means that x is a new node that has not been visited in this invocation, and x is not in the initial set sampled nodes T}
end while

new stuck node, we repeat the process of sampling s edges out of S and all the stuck nodes visited since the last w-length walk was used. Finally, if the new endpoint is not a stuck node, we continue appending w length walks as before.

We need to argue that whenever the algorithm hits a stuck node, it makes sufficient progress with each pass. Note that after each round of sampling s edges out of the stuck nodes and the nodes in S, the walk is extended further. Either the walk reaches a node that is not stuck, thereby resulting in w progress, or the walk is extended until it again reaches a node that is stuck. Notice that in the latter case, s steps of progress is made. The point is that we cannot keep finding new stuck nodes repeatedly for too long as each new node is a sampled node with probability α. So, it is unlikely we will visit more than $\tilde{O}(1/\alpha)$ new stuck nodes in a sequence before becoming unstuck. These steps are detailed in the algorithm SINGLERANDOMWALK.

The notation in the algorithm SINGLERANDOMWALK uses T to denote the sampled nodes obtained by sampling each node independently with probability α. The table W indexed by a sampled node (say t) stores the end point of the w length walks starting at t as $W[t]$. Note that this table can be populated in w passes using $O(\alpha n)$ space. The set S keeps track of all nodes in T whose w length walks get used up. The algorithm continues extending the walk using the w length walks implicitly stored in the table

W until it finds a stuck node. The module HANDLESTUCKNODE proceeds by sampling s edges out of $S \cup R$ where R is the set of stuck nodes visited in the current invocation of SINGLERANDOMWALK. Finally, $\mathcal{L}_u$ denotes the random walk of length l from u. We now state some claims using this notation, based on the algorithm.

Remark 3.3. The length of the walk produced by algorithm SINGLERANDOMWALK could exceed l slightly (by at most w). To prevent this, we can run the algorithm till we get a walk of length at least $l - w$ and then extend this walk to length l in at most w additional passes.

We begin the analysis with a lemma that follows immediately from the algorithm.

LEMMA 3.4. $|S| \leq l/w$.

PROOF. A node is added to the set S only after we use a w length walk from one of the sampled nodes. If we perform a walk of length l, we will end up using at most $\frac{l}{w}$ walks of length w. □

We now state and prove the main claim that is needed to bound the number of passes required by algorithm SINGLERANDOMWALK to perform a random walk of length l.

CLAIM 3.5. *With every additional pass over the edge stream (after the first w passes), the length of the random walk $\mathcal{L}_u$ either increases by s, or if it does not increase by s then with probability α it increases by w. Further, $|R| \leq \tilde{O}(1/\alpha)$ with high probability.*

PROOF. We only need to examine the algorithm HANDLESTUCKNODE. An additional pass over the stream is made when s edges are sampled (independently, with repetition) from every node in $S \cup R$. This happens when the algorithm gets stuck at a new stuck node in R. After a pass over the stream, either the algorithm makes s progress, or a *new node* is visited. This is because, after a pass, all nodes that were stuck (i.e., nodes whose w-length walks are used up) now have s edges sampled out of them. If on extending the walk, the walk continues to visit these nodes, the algorithm completes at least s steps before getting stuck again at one of the nodes in S. However, the algorithm may exit the set of stuck nodes and end up at a node outside, before completing s steps. In this case, with probability α, the new node is in T (since T contains each node with probability α), and with probability $1 - \alpha$, it is a new stuck node. This is because each of these new nodes was sampled independently with probability α at the beginning of the algorithm. If the new node is not a stuck node, w progress is made. The probability of not seeing a new node in T is $1 - \alpha$ with every additional pass that lands at a new node. Therefore, the probability that more than $O(\log(nl)/\alpha)$ new stuck nodes are seen before a new node in T is seen is small (at most $1/poly(nl)$) by Chernoff Bounds. So w.h.p., $|R|$ is less than $\tilde{O}(1/\alpha)$ in each invocation of HANDLESTUCKNODE. □

We are now ready to prove Theorem 3.1.

PROOF OF THEOREM 3.1.

Correctness. We first argue that the walk of length l from source u generated by our algorithm is indeed a *random* walk. Notice that the algorithm uses each w-length walk only once in the walk $\mathcal{L}_u$. Note that the algorithm never reuses any randomly sampled edges or walks. Whenever we sample s edges, we pick the ith sampled edge, when visiting the node for the ith time. Therefore, randomness is maintained. It is important to note that while sampling s edges, we (correctly) allow the same edge to be sampled multiple times; in particular, this would definitely happen for a vertex with degree less than s.

Space. We need space $O(\alpha n)$ for storing the sampled nodes and the end point of their w length walks. Using Lemma 3.4, sampling s edges from every node in S requires

$O(s(l/w))$ space, while sampling s edges from the nodes not in R takes up an additional $\tilde{O}(s\frac{1}{\alpha})$ space as $|R| = \tilde{O}(1/\alpha)$ w.h.p. $\tilde{O}\big(n\alpha + s\big(\frac{l}{w} + \frac{1}{\alpha}\big)\big)$.

Passes. The most crucial observation in analyzing the number of passes required by the algorithm SINGLERANDOMWALK is Claim 3.5. This claim states that in case s progress is not made in a pass then with probability α w progress is made. The number of passes in which s progress is made is at most l/s. Now let us bound the number of passes in which case s progress is not made – then with probability α, w progress is made which can happen at most l/w times. Now, in $\tilde{O}(1/\alpha)$ such passes at least once w progress is made. Thus, the number of passes when s progress is not made is at most $\tilde{O}(\frac{l}{w\alpha})$ w.h.p (the $\tilde{O}$ includes $\log(nl)$ factors to achieve a high probability of $1 - 1/poly(nl)$). Additionally, w passes are used for generating the w length walks from each of the sampled $O(\alpha n)$ nodes.

Therefore, the total number of passes used in the algorithm is $\tilde{O}(w + (l/s) + (l/w\alpha))$ with high probability. Setting $s = \sqrt{l\alpha}$ and $w = \sqrt{l/\alpha}$ completes the proof. □

Note that SINGLERANDOMWALK takes sublinear space and passes even for performing very long ($O(n)$ length) random walks. Setting $l = O(n)$ and choosing $\alpha = n^{-\frac{1}{3}}$ in Theorem 3.1 gives the following corollary.

COROLLARY 3.6. *One can perform a random walk of length $O(n)$ in $\tilde{O}(n^{2/3})$ passes and $\tilde{O}(n^{2/3})$ space.*

The previous algorithm can easily be extended to the case when the starting node of the random walk comes from a distribution, rather than a specific node. In this case, one can sample a node from the initial distribution and use this as the source node for the random walk.

Notice that if we wanted to perform a larger number of independent random walks using this algorithm directly, the space required would increase linearly in the number of walks, while the passes would remain unchanged. The bottleneck in the space requirement would arise due to two reasons. First, the algorithm would need to store multiple w-length walks from each sampled node, one for each random walk. Second, many of these random walks could get stuck at the same time, and the algorithm may be required to sample s edges out of the centers of many walks. In the following section, we reduce the space requirements arising in both these scenarios by trying to identify the *appropriate* number of w-length walks required for each sampled node.

The previous algorithm only samples end points from the distribution at length l. One can in fact regenerate the entire walk by using a pseudo-random generator for the coin tosses during the algorithm.

Remark 3.7. Note that since we only store the end-points of w length walks in W, the internal nodes are not available in $\mathcal{L}_u$ at the end of the algorithm SINGLERANDOMWALK. These w-length walks can be reconstructed by making pseudorandom choices while creating the w length random walks in Step 3 of SINGLERANDOMWALK, and reusing the coin tosses to reconstruct them at the end. A single pseudo-random hash function can be used to generate all the coin tosses.

4. ESTIMATING PROBABILITY DISTRIBUTION BY PERFORMING A LARGE NUMBER OF RANDOM WALKS

We now show how to estimate the probability distribution of the destination node after performing a random walk. We achieve this by performing several random walks. The source node may either be fixed or chosen from a certain initial distribution. A naïve method that uses algorithm SINGLERANDOMWALK to perform K random walks would

require $O(K(n\alpha + \sqrt{l/\alpha}))$ space. In this section, we show how algorithm SINGLERANDOMWALK can be extended to perform n/l random walks without significant increase in the space-pass complexity. Specifically, we show the following result.

THEOREM 4.1. *One can perform K random walks of length l in $\tilde{O}(\sqrt{l/\alpha})$ passes and $\tilde{O}(n\alpha + K\sqrt{l/\alpha} + Kl\alpha)$ space for any choice of α with $0 < \alpha \leq 1$.*

In particular, if $K = \frac{n}{l}$, then for $\alpha \geq l^{-1/3}$, the space requirement is $O(n\alpha)$ which is as good as the space complexity of SINGLERANDOMWALK. We first describe how Theorem 4.1 can be used to estimate the probability distribution after a random walk of length l.

By performing a large number of random walks and computing the fraction of walks that end at a given node gives us an estimate of the probability that a random walk ends at this node. If the actual probability of ending at a node is p, then by setting $K = \Theta(\log n/\epsilon)$, we get an estimate for p with accuracy $p \pm \sqrt{\epsilon p} \pm \epsilon$. By Chernoff bounds, due to the $\log n$ factor, this estimate is valid w.h.p. for all nodes.

The specific form of the bound we require here (and use repeatedly in our analysis later on as well) is stated in Lemma 4.2.

LEMMA 4.2. *If the probability of an event X occurring is p, then in $t = \Theta(\log n/\epsilon)$ trials, the fraction of times the event X occurs is $(p \pm \sqrt{p\epsilon} \pm \epsilon)$ times w.h.p.*[3]

PROOF. Given independent identically distributed random variables X_i, such that $Pr[X_i = 1] = p$ and $Pr[X_i = 0] = (1-p)$, and t events $X_1, X_2, \ldots, X_t$, standard Chernoff or Hoeffding Bounds says for the lower tail

$$Pr\left[\frac{1}{t}\sum_{i=1}^{t} X_i < (1-\delta)tp\right] < \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{tp} < e^{-tp\delta^2/2}.$$

For the upper tail, the bound is

$$Pr\left[\frac{1}{t}\sum_{i=1}^{t} X_i > (1+\delta)tp\right] < \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{tp}.$$

Further, for $\delta > 2e-1$, this bound reduces to $2^{-\delta tp}$ and for $\delta < 2e-1$, this bound reduces to $e^{-tp\delta^2/4}$.

For the lemma, we choose $t = \Theta((\log n/\epsilon))$, and $\delta = \sqrt{(\epsilon/p)} + (\epsilon/p)$. Now consider two cases, one where $p > \epsilon$ and when $p \leq \epsilon$. When $p \leq \epsilon$, the lower tail bound follows automatically since $p - \epsilon < 0$. For the upper tail bound, we have $\delta > 1$; in this case, the weaker bound is $2^{-\delta tp}$ and we show that this suffices to prove the lemma. We have $2^{-\delta tp} = 2^{-(\sqrt{\epsilon p}+\epsilon)t} \leq 2^{-\epsilon t} = 2^{-\log n} = 1/n$. Now consider the case when $p > \epsilon$. Now, $\delta < 2$ is small and so the lower and upper tail bounds are $e^{-tp\delta^2/2}$ and $e^{-tp\delta^2/4}$, respectively. In this case, $\sqrt{\epsilon/p} > \epsilon/p$; therefore, we get an asymptotic bound of $e^{-\Theta(tp(\sqrt{\epsilon/p})^2)} = e^{-\Theta(\frac{\log n}{\epsilon}p(\epsilon/p))} = e^{-\Theta(\log n)} = \frac{1}{n^{\Theta(1)}}$. This completes the proof. □

We then get the following corollary.

COROLLARY 4.3. *For any node with probability p in the probability distribution after a walk of length l, one can approximate its probability up to an accuracy of $p \pm \sqrt{p\epsilon} \pm \epsilon$ in $\tilde{O}(\sqrt{l/\alpha})$ passes and $\tilde{O}(n\alpha + (1/\epsilon)\sqrt{(l/\alpha)} + (1/\epsilon)l\alpha)$ space for any choice of α with $0 < \alpha \leq 1$. This is a $(1 \pm \sqrt{(\epsilon/p)} \pm (\epsilon/p))$-factor approximation for a node with probability p in the distribution.*

[3]probability $1 - 1/n^{\Omega(1)}$.

Notice that for $p \gg \epsilon$, this is a constant factor approximation close to 1.

By applying this algorithm on the web-graph (including the random reset edges that are handling implicitly), we can estimate the PageRank vector up to an accuracy of $p \pm \sqrt{p\epsilon} \pm \epsilon$ for any node with probability p in the stationary distribution, in $\tilde{O}(\sqrt{M/\alpha})$ passes and $\tilde{O}(n\alpha + 1/\epsilon\sqrt{M/\alpha} + (1/\epsilon)M\alpha)$ space, where M is the mixing time of the graph. We will show later in Section 5 how to estimate the mixing time M of any graph. Note that our algorithm is also able to handle the random resets in the standard definition of PageRank by handling these transitions implicitly; that is the transition edges corresponding to the random resets can be included into the graph implicitly.

Our technique for performing a large number of walks uses algorithm SINGLERANDOMWALK as a subroutine. The key idea in our approach is to *estimate* the probability r_i that the w length walk of node i is used in SINGLERANDOMWALK. We then use the r_i's to store the appropriate number of w length walks from each sampled node for K executions of SINGLERANDOMWALK. Estimating r_i however again requires performing random walks. For this purpose, we start by performing one random walk, then two, then four and so on, doubling the number of random walks in every phase, giving more and more accurate estimates of r_i for the sampled nodes.

Let us first define r_i for every node i assuming a given set of sampled nodes in Algorithm SINGLERANDOMWALK. Note that this definition is dependent on the length of the walk but we omit write this as a superscript.

Definition 4.4 (r_i). For any sampled node i, define r_i to be the probability that on running the algorithm SINGLERANDOMWALK, the w length walk of node i is used (and hence i gets included in the set of centers S in the performed walk of length l).

Lemma 3.4 states that $|S| \leq l/w$. Notice that a node is included in S if and only if its w length random walk is used towards the random walk $\mathcal{L}_u$. By our definition of r_i, we have that $\sum_i r_i$ is equal to the expected size of $|S|$. From these two statements, we get the following observation.

Observation. $\sum_i r_i \leq l/w$.

We now describe the algorithm for performing a large number of random walks. Whenever we say *sample x walks of length w from i*, we mean take the endpoints of x independent random walks of length w starting at i.

This algorithm runs in phases. To obtain K walks of length l, algorithm MULTIPLERANDOMWALK is run for $j = \log K$ phases. In phase $j+1$ we run $O(2^j \log n)$ parallel executions of SINGLERANDOMWALK and use these to get an estimate $\tilde{r}_i$ of r_i to an additive error of $\sqrt{r_i/2^j} + 1/2^j$. This estimate is then used in the next phase to store the appropriate number of w length walks from each i. Note that, all the executions of SINGLERANDOMWALK share the same set of sampled nodes.

We are now ready to prove Theorem 4.1.

PROOF OF THEOREM 4.1.

Correctness. We need to show that the number of w length walks we store from the sampled nodes in step 1 of each phase $j+1$ in MULTIPLERANDOMWALK is *sufficient* for K_{j+1} executions of SINGLERANDOMWALK. Note that (assuming previous phases have completed successfully) in each phase the estimate $\tilde{r}_i$ is accurate up to an additive error of $\sqrt{r_i/2^j} + 1/2^j$ w.h.p; this follows from lemma 4.2 since we are using $K_j = \tilde{O}(2^j)$ walks to estimate r_i. Therefore, our estimate $\tilde{r}_i$ is w.h.p. at least $r_i - \sqrt{r_i/2^j} - 1/2^j \geq r_i/2 - 1.5/2^j$ (since $\sqrt{xy} \leq (x+y)/2$, for any positive reals x, y). It follows that the actual value r_i is at most $O(\tilde{r}_i + 1/2^j)$ w.h.p. The number of walks in phase $j+1$ that use this node as a center is w.h.p., by Chernoff bounds, at most $O(K_{j+1}r_i + \log n) \leq O(2^{j+1}\tilde{r}_i \log n + 2 + \log n) =$

ALGORITHM 3: MULTIPLERANDOMWALK($\mathcal{I}$, l, K)

1: **Input**: Distribution of the source nodes, $\mathcal{I}$, and length of the walk, l
2: $T \leftarrow$ set of nodes obtained by sampling each node independently with probability α.
3: Perform phases 1 through $\log K$ as follows -
4: **Phase 1:**
   (1) Perform $O(\log n)$ walks of length w from each of the sampled nodes, in w passes.
   (2) Spawn $K_1 = O(\log n)$ instances of SINGLERANDOMWALK to obtain K_1 walks. All these instances use only the w-length walks in the previous step.
   (3) For each sampled node $i \in T$, set estimate $\tilde{r}_i = n_i/K_1$ where n_i is the number of walks produced in the previous step that use i as a center.
5: **Phase** $(j+1)$**:** {The value of $\tilde{r}_i$ differs from r_i by an additive error of $O(\sqrt{r_i/2^j} + 1/2^j)$ w.h.p.}.
   (1) In w passes, sample $O(2^j \tilde{r}_i \log n + \log n)$ random walks of length w for all nodes in T.
   (2) Run $K_{j+1} = 2^j O(\log n)$ independent instances of SINGLERANDOMWALK using the w- length random walks sampled in the previous step.
   (3) Set estimate $\tilde{r}_i = n_i/K_{j+1}$ where n_i is the number of walks that use i as a center.

$O(2^j \tilde{r}_i \log n + \log n)$. This is exactly the number of w-length walks we sample in phase $j+1$. This implies that w.h.p. any time one of $K_{j+1} = 2^j O(\log n)$ walks hits a sampled node for the first time, there is an unused w length walk for that node, to extend it. By including $\log nKl$ factors in the $\tilde{O}$, each individual walk holds with probability at least $1 - 1/poly(nKl)$; so it holds with high probability over all the K walks. No w length walk or sampled edge, is ever reused throughout the execution; therefore, all random choices in the eventual walk are independent.

Space. Suppose we do t phases such that $2^t = K$. Total space required would include $O(n\alpha)$ space to store the sampled nodes. Further, to store the w length walks in phase t, we need $O(\sum_{i \in T}(2^t \tilde{r}_i \log n + \log n))$ space. Now $\tilde{r}_i \leq r_i + \sqrt{r_i/2^t} + 1/2^t \leq O(r_i + 1/2^j)$ since $(\sqrt{xy} \leq (x+y)/2$ for $x, y \geq 0)$. So the space to store the w length walks in phase t is at most $O(\sum_{i \in T}(2^t r_i \log n + 1 + \log n)) = \tilde{O}(K(l/w) + n\alpha)$ space. And finally, the space for sampling s edges in each execution of HANDLESTUCKNODE amounts to $\tilde{O}(K\frac{l}{w}s + K\frac{1}{\alpha}s)$. Therefore, the algorithm MULTIPLERANDOMWALK uses a total space of $\tilde{O}(K\frac{l}{w}s + K\frac{1}{\alpha}s + \alpha n)$.

Passes. The number of passes required in algorithm MULTIPLERANDOMWALK in any given phase is the same as in SINGLERANDOMWALK, no matter how many walks are being run in that phase. So the total passes required for a given phase is $\tilde{O}(w + (l/s) + (l/w\alpha))$ again using Claim 3.5. The number of phases run is $\log K$. It follows that the total number of passes in MULTIPLERANDOMWALK is $\tilde{O}(w + (l/s) + (l/w\alpha))$. Setting $w = \sqrt{l/\alpha}$ and $s = \sqrt{l\alpha}$, the theorem follows. □

We now show how MULTIPLERANDOMWALK algorithm can be used for estimating the mixing time.

5. ESTIMATING MIXING TIME

In our previous algorithms, we need to preform walks of length l equal to the mixing time, to estimate the PageRank distribution. We now present an algorithm to estimate the mixing time of a graph. However, instead of computing the exact mixing time, we compute the time required for *approximate* mixing of a random walk. That is, we compute a length l such that running a random walk for l steps from an initial distribution ends at a node with a probability distribution that is *close* to the stationary distribution. We wish to compute l given an initial distribution u. We denote the mixing

time for an initial distribution u to reaching within ϵ in L_1 distance of the steady state distribution by $l_u(\epsilon)$. The following definition makes this precise for undirected graphs.

Definition 5.1 (ϵ-*Near Mixing Time of Undirected Graph*). We say that $l_u(\epsilon)$ is the ϵ-*near mixing time* of an undirected graph for an initial distribution u if the L_1-distance between the steady state distribution and the distribution obtained after a random walk of length $l_u(\epsilon)$ starting at node from distribution u is at most ϵ. Further, $l_u(\epsilon)$ must be the shortest such length that satisfies this condition.

For directed graphs, we have a weaker definition of ϵ-*near mixing time*.

Definition 5.2 (ϵ-*Near Mixing Time of Directed Graph*). We say that $l_u(\epsilon)$ is the ϵ-*near mixing time* of a directed graph for an initial distribution u if the L_1-distance between the distribution obtained after a random walk of length $l_u(\epsilon)$ starting at a node from distribution u, and the distribution obtained after a random walk of length $l_u(\epsilon) + poly(1/\epsilon)$, is at most ϵ.

LEMMA 5.3. *The ϵ-near mixing time is monotonic property, that is, if given a source distribution, the distribution after a walk of length l is within ϵ in L_1 distance of the steady state distribution, then so is a walk of length $l+1$.*

PROOF. The monotonicity follows from the fact that $||xA||_1 \leq ||x||_1$ for any transition probability matrix A and for any vector x. This, in turn, follows from the fact that the sum of entries of any column of A is 1.

Now let π be the stationary distribution of the transition matrix A. This implies that if l is ϵ-near mixing, then $||uA^l - \pi||_1 \leq \epsilon$, by definition of ϵ-near mixing time. Now consider $||uA^{l+1} - \pi||_1$. This is equal to $||uA^{l+1} - \pi A||_1$ since $\pi A = \pi$. However, this reduces to $||(uA^l - \pi)A||_1 \leq \epsilon$. It follows that $(l+1)$ is ϵ-near mixing. □

In this section, specifically, we show the following result.

THEOREM 5.4. *Given u, one can find a time that is between $l_u(6\epsilon)$ and $l_u(\epsilon^3/4\sqrt{n}\log n)$ in $\tilde{O}(n\alpha + poly(\epsilon^{-1})(\sqrt{(M_u n/\alpha)} + M_u\alpha\sqrt{n}))$ space and $\tilde{O}(\sqrt{M_u/\alpha})$ passes over the stream, where $M_u = l_u(\epsilon^3/4\sqrt{n}\log n)$. For directed graphs, one can find a time that is between $l_u(\epsilon)$ and $l_u(\max\{\epsilon^2/(32n^{1/3}), \epsilon/(4\sqrt{n})\})$, in $\tilde{O}(\sqrt{\frac{M_u}{\alpha}})$ passes $\tilde{O}(n\alpha + poly(\epsilon^{-1})(n^{2/3}\sqrt{(M_u/\alpha)} + M_u\alpha n^{2/3}))$ space, where $M_u = l_u(\max\{\epsilon^2/(32n^{1/3}), \epsilon/(4\sqrt{n})\})$.*

The naïve approach to compute the mixing time requires $O(n)$ space and $O(M_u)$ passes over the input stream This computes uA^{M_u} exactly where u is the initial vector of size n and A the matrix representation of the graph. It takes n space to maintain this vector, and M_u passes to multiply by A once in every pass.

The main idea in estimating the mixing time is to run many random walks of length l from the specified source using the approach described in the previous section, and use these to compute the distribution after l-length random walk. We then compare the distribution at different l, with the stationary distribution, to check if the two distributions are ϵ-near. We need to address the following issues. First, we do not know what value(s) of l to try. Second, we need to compare these distributions with the steady state distribution; while the steady state distribution is easy to compute for an undirected graph, it is hard to compute for directed graphs.

To compare two distributions, we use the technique of Batu et al. [2001] to determine if the distributions are ϵ-near. Their result is summarized in the following theorem.

THEOREM 5.5 ([BATU ET AL. 2001]). *Given $\tilde{O}(n^{1/2}poly(\epsilon^{-1}))$ samples from a black-box distribution X over $[n]$, and a specified distribution Y, there is a test that outputs PASS*

ALGORITHM 4: ModifiedSingleRandomWalk(u, l)

1: **Input:** Starting node u, and desired walk length l. Array $W[t,k]$ for $t \in T$, $0 < k \leq \infty$ of infinitely many random walks of length w for every node in T; $W[t,k]$ denotes the $k^t h$ walk for node t.
2: **Output:** $\mathcal{L}_u$ the random walk from u of length l.
3: $T \leftarrow$ set of nodes obtained by sampling each node independently with probability α.
4: Let $count[t]$ denote the next unused w length walk of t we will use. Initialize $count[t] = 1$ for all t.
5: Initialize $\mathcal{L}_u$ to a zero length walk starting at u. Let $x \leftarrow u$, the source node.
6: **while** $|\mathcal{L}_u| < l$ **do**
7: (1) if ($x \in T$) extend $\mathcal{L}_u$ by appending the walk corresponding to $W[x, count[x]]$. $count[x] \leftarrow count[x] + 1$. $x \leftarrow$ new end point of $\mathcal{L}_u$. {we increment $count[x]$ so that the next time the walk ends at x, we will use the next w length walk from x stored in the table W.}
(2) if ($x \notin T$) ModifiedHandleStuckNode(x, T, $\mathcal{L}_u$, l). {this means that x was not in the initial set of sampled nodes}
8: **end while**

with high probability if $|X-Y|_1 \leq \frac{\epsilon^3}{4\sqrt{n}\log n}$, *and outputs FAIL with constant probability if* $|X-Y|_1 > 6\epsilon$. *Similarly, given two black-box distributions* X *and* Y *over* $[n]$, *there is a test that requires* $O(n^{2/3}\epsilon^{-4}\log n \log(1/\delta))$ *samples which outputs PASS with probability at least* $1-\delta$ *if* $|X-Y|_1 \leq \max\{\epsilon^2/(32n^{1/3}), \epsilon/(4\sqrt{n})\}$, *and outputs FAIL with probability at least* $1-\delta$ *if* $|X-Y|_1 > \epsilon$.

We are now ready to prove Theorem 5.4.

Proof of Theorem 5.4. For undirected graphs, the stationary distribution of the random walk is well-known to be $\frac{deg(i)}{2m}$ for node i with degree $deg(i)$, where m is the number of edges in the graph. We only need $\tilde{O}(n^{1/2}poly(\epsilon^{-1}))$ samples from a distribution to compare it to the stationary distribution. This can be achieved by running MultipleRandomWalk to obtain $K = \tilde{O}(n^{1/2}poly(\epsilon^{-1}))$ random walks. To find the approximate mixing time, we try out increasing values of l that are powers of 2. Once we find the right consecutive powers of 2, the monotonicity property admits a binary search to determine the exact value of ϵ-near mixing time. Note that we can apply binary search as ϵ-near mixing time is a monotonic property.

The result in Batu et al. [2001] also provides an approach to determine if two unknown distributions X and Y over $[n]$ are ϵ-close in L_1 norm; however, this requires $\tilde{O}(n^{2/3}poly(\epsilon^{-1}))$ samples from each distribution. This completes the proof. □

Theorem 5.4 gives some interesting consequences for specific values of α and M_u. We state some below. In the extreme cases of $\alpha = 1$ and $\alpha = 1/M_u$, we can calculate the ϵ-near mixing time with either of the trade-offs presented in the following corollary.

Corollary 5.6. *Given* u, *one can find a time that is between* $l_u(6\epsilon)$ *and* $l_u(\epsilon^3/(4\sqrt{n}\log n))$ *in either* $\tilde{O}(n) + poly(\epsilon^{-1})(M_u\sqrt{n})$ *space and* $\tilde{O}(\sqrt{M_u})$ *passes, or* $\tilde{O}\big(\frac{n}{M_u} + M_u\sqrt{n}poly(\epsilon^{-1})\big)$ *space and* $\tilde{O}(M_u)$ *passes, where* $M_u = l_u(\max\{\epsilon^2/(32n^{1/3}), \epsilon/(4\sqrt{n})\})$

Assuming the actual mixing time of the graph (worst case over all source nodes), say M, is within constant factors of the estimated mixing time, one can compute a square-root approximation to the conductance Φ of the graph as $\Theta(1/M) \leq \Phi \leq \Theta(1/\sqrt{M})$ as

shown in Jerrum and Sinclair [1989]. The conductance of a graph G is defined as

$$\phi(G) = \min_{S:|S|\leq|V|/2} \frac{E(S, V(G) \setminus S)}{E(S)}$$

where $E(S, V(G) \setminus S)$ is the number of the edges spanning the cut, and $E(S)$ is the number of edges on the smaller side of the cut.

6. ESTIMATING DISTRIBUTIONS WITH BETTER ACCURACY

In this section, we present an algorithm that has a better space complexity when the accuracy parameter ϵ is less than $\sqrt{l}/n$. The main idea is to replace the estimation of r_i's in MULTIPLERANDOMWALK, that measures the probability that node i is used as a center, by another quantity q_i that measures how many w length walks from node i may be used. We start with a modification of SINGLERANDOMWALK, where we assume that there are infinitely many w length walks out of each sampled node, that can be accessed as an oracle. We then look at the expected number (q_i) of w length walks that will be used for each node i. Subsequently, in MODIFIEDMULTIPLERANDOMWALK, we estimate q_i in phases by doubling the number of walks in each phase getting better accuracy each time (just as in MULTIPLERANDOMWALK).

We begin by describing the modification to the algorithm SINGLERANDOMWALK. The main difference is that there are many w length walks available from the sampled nodes; so the algorithm gets stuck only when it hits a nonsampled node. This is different from the earlier version where the algorithm got stuck even if it visited a sampled node for the second time. The notation in the algorithm below uses T to denote the sampled nodes obtained by sampling each node with probability α independently. The table W stores infinitely many w length walks per node in T; $W[t, k]$ is the endpoint of the k'th w length walk starting at t. At most $count[t]$ of these walks is actually used. Note that this table can be populated in l passes, however the space requirement depends on the maximum $count[t]$ for every t. Right now, we assume that the table W is infinite and we can obtain a w length walk for any value of $count[t]$. Unlike in SINGLERANDOMWALK where we defined the set S to keep track of all nodes in T whose w length walks get used up, in MODIFIEDSINGLERANDOMWALK, we do not need S. The algorithm continues extending the walk using the w length walks implicitly stored in the table W until it finds a stuck node. The module MODIFIEDHANDLESTUCKNODE proceeds by sampling s edges out of R where R is the set of stuck nodes visited in the current invocation. In this case, only a nonsampled node can be a stuck node.

With the algorithm in place, we now need to define q_i, that we use instead of p_i, in MODIFIEDMULTIPLERANDOMWALK. Assume a given set of sampled nodes T.

Definition 6.1 (q_i). For any sampled node i, define q_i to be the expected value of $count[i]$ at the end of the execution of l-length random walk using MODIFIEDSINGLERANDOMWALK.

Notice that the main difference in MODIFIEDSINGLERANDOMWALK as compared to SINGLERANDOMWALK is that there is no set S to store the *centers* whose w length walk has been used up. Every node in the initial sampled set T has sufficient number of w length walks. So whenever this walk gets stuck it is stuck at a *nonsampled node*. In this case, s edges are sampled out of all the new nodes visited since the last time a w length walk was used; as before this set is denoted by R.

Estimating q_i using SINGLERANDOMWALK. We now show how to estimate q_i. In the first phase, we use SINGLERANDOMWALK to perform $O(\log n)$ walks. To estimate the q_i's, the entire walks of length l need to be reconstructed. This can be done using the pseudo-random coin tosses as described in Remark 3.7 which adds $\tilde{O}(l)$ to the space

ALGORITHM 5: ModifiedHandleStuckNode(x, T, $\mathcal{L}_u$, l)

1: $R \leftarrow x$.
2: **while** $|\mathcal{L}_u| < l$ **do**
3: $E \leftarrow$ sample s edges (with repetition) out of each node in R.
4: Extend $\mathcal{L}_u$ as far as possible by walking along the sampled edges in E (on visiting a node in R for the k-th time, use the k-th edge of the s sampled edges from that node).
5: $x \leftarrow$ new end point of $\mathcal{L}_u$ after the extension. One of the following cases arise.

(1) if ($x \in R$) **continue** {no new node is seen}
(2) if ($x \in T$) **return** {this means that x is a sampled node; therefore, we can use the next w-length walk from x by accessing the table W}
(3) if ($x \notin T$ and $x \notin R$) $R \leftarrow R \cup \{x\}$. {this means that x is a new node that has not been visited in this invocation, and x is not in the initial set sampled nodes T}

6: **end while**

requirements. Once the walks of length l are reconstructed, for any walk, start walking from the source node; each time a sampled node (say t) is seen, increment $count[t]$ and skip w steps, and continue walking till the end. Set the estimate $\tilde{q}_i$ to be the average of $count[i]$ over the $O(\log n)$ random walks. In phase $j+1$, we run $K_{j+1} = 2^j O(\log n)$ walks and use this to obtain improved estimates of q_i for the next phase.

ALGORITHM 6: ModifiedMultipleRandomWalk($\mathcal{I}$, l, K)

1: **Input**: Distribution of the source nodes, $\mathcal{I}$, and length of the walk, l
2: $T \leftarrow$ set of nodes obtained by sampling each node independently with probability α.
3: Perform phases 1 through $\log K$ as follows -
4: **Phase 1**: Perform $O(\log n)$ walks of length l using SingleRandomWalk. Estimate q_i using the technique described above for reconstructing the walks and tracking $count[i]$ for all $i \in T$. Set estimate $\tilde{q}_i$ for q_i to be the average of $count[i]$ over all the $\log n$ walks.
5: **Phase** $(j+1)$**:** {spawn K_{j+1} walks}

—In w passes, sample $K_{j+1} = \tilde{O}(2^j q_i + \frac{l}{w})$ walks from all $i \in T$.
—Perform $2^j O(\log n)$ random walks using ModifiedSingleRandomWalk and again compute the estimates $\tilde{q}_i$'s. The estimate $\tilde{q}_i$ is obtained by taking the average value of $count[i]$ over the K_{j+1} executions.

By definition of q_i, $\sum q_i \leq \frac{l}{w}$. Since $\frac{q_i}{l/w}$ is a random variable in the range $[0, 1]$, in K_{j+1} walks the estimate $\tilde{q}/l/w$ is such that $\tilde{q}/l/w = q/l/w \pm (\sqrt{(q_i/(2^j l/w))} + \frac{1}{2^j})$ w.h.p.

Theorem 6.2. ModifiedMultipleRandomWalk *can be used to perform K random walks of length l in $\tilde{O}(\sqrt{l/\alpha})$ passes and $\tilde{O}(\alpha n\sqrt{l\alpha} + K\sqrt{(l/\alpha)} + l)$ space.*

Proof.

Correctness. The proof is similar to that of Theorem 4.1. We need to show that the number of w length walks we store from the sampled nodes in Step 1 of each phase $j+1$ in MultipleRandomWalk is *sufficient* for K_{j+1} executions of SingleRandomWalk. Note that (assuming previous phases have completed successfully) in each phase the estimate $\frac{\tilde{q}_i}{l/w}$ is accurate up to an additive error given by

$$\frac{\tilde{q}_i}{l/w} = \frac{q_i}{l/w} \pm \left(\sqrt{\left(\frac{q_i}{l/w}\right)\frac{1}{2^j}} + \frac{1}{2^j}\right)$$

w.h.p. Simplifying as in the proof of Theorem 4.1 (using the fact that $\sqrt{xy} \leq (x+y)/2$), we get w.h.p. that

$$\frac{\tilde{q}_i}{l/w} \geq \frac{q_i}{l/w} - \sqrt{\left(\frac{q_i}{l/w}\frac{1}{2^j}\right)} - \frac{1}{2^j} \geq \frac{q_i}{2l/w} - \frac{1.5}{2^j}.$$

It follows that $\frac{q_i}{l/w}$ is at most $O\left(\frac{\tilde{q}_i}{l/w} + 1/2^j\right)$ w.h.p, or $q_i = O\left(\tilde{q}_i + \frac{l}{w2^j}\right)$ w.h.p. The number of walks in phase $j+1$ that use this node as a center is w.h.p., by Chernoff bounds, at most $O(K_{j+1}q_i + \log n) \leq O(2^{j+1}\tilde{q}_i \log n + 2l/w + \log n) = \tilde{O}(2^j q_i + l/w)$. This is exactly the number of w-length walks we sample in phase $j+1$. This implies that w.h.p. any time one of $K_{j+1} = 2^j O(\log n)$ walks hits a sampled node, there is an unused w length walk for that node, to extend it. No w length walk or sampled edge, is ever reused throughout the execution; therefore, all random choices in the eventual walk are independent.

Space. Suppose we do t phases such that $2^t = K$. Total space required would include $O(n\alpha)$ space to store the sampled nodes; to store the w length walks in the phase $(t-1)$, we need

$$\tilde{O}\left(\sum_{i\in T}\left(2^t q_i + \frac{l}{w}\right)\right) = \tilde{O}\left(K\sum_{i\in T} q_i + n\alpha\frac{l}{w}\right) = \tilde{O}\left(K\left(\frac{l}{w}\right) + n\alpha\frac{l}{w}\right)$$

space; and finally, the space for sampling s edges in each execution of ModifiedHandleStuckNode amounts to $\tilde{O}(K\frac{1}{\alpha}s)$. Observe that in ModifiedHandleStuckNode, we only sample s edges out of R instead of $S \cup R$ as in HandleStuckNode. Also, as shown before, $|R| \leq \tilde{O}(1/\alpha)$ since each new node is likely to be a sampled node with probability α. So, the space required for the executions of ModifiedhandleStuckNode is $\tilde{O}(K\frac{1}{\alpha}s)$. Also the first phase required $\tilde{O}(l)$ space to reconstruct the walks obtained from SingleRandomWalk. Therefore, the total space required for this algorithm is

$$\tilde{O}\left(K\frac{l}{w}s + K\frac{1}{\alpha}s + \alpha n\frac{l}{w} + l\right).$$

Passes. The number of passes required is same as in MultpleRandomWalk. So the total passes required for this walk is still $\tilde{O}(w + \frac{l}{s} + \frac{l}{w\alpha})$.

Setting $s = \sqrt{l\alpha}$ and $w = \sqrt{l}\alpha$ completes the proof. □

Comparing this with Theorem 4.1, we see that in the space requirement, the term $Kl\alpha$ is no longer there; however, the space of $O(n\alpha)$ in Theorem 4.1 has increased to $O(\alpha n\sqrt{l\alpha})$; additionally, for the first phase where we needed to reconstruct the walks, we incurred an additional space requirement of $O(l)$. Depending on the values and bounds required, one of these theorems is a better than the other. This gives the following corollary similar to Corollary 4.3.

Corollary 6.3. *For any node with probability p in the probability distribution after a walk of length l, one can approximate its probability up to an accuracy of $p \pm \sqrt{p\epsilon} \pm \epsilon$ for any $\epsilon > 0$ in $\tilde{O}(\sqrt{l/\alpha})$ passes and*

$$\tilde{O}\left(\alpha n\sqrt{l\alpha} + \frac{1}{\epsilon}\sqrt{\frac{l}{\alpha}} + l\right)$$

space. This also implies a

$$\left(1 \pm \sqrt{\frac{\epsilon}{p}} \pm \frac{\epsilon}{p}\right)\text{-approximation ratio}$$

for any node with value p in the probability distribution.

Notice that this is a constant (close to 1) factor approximation for any node with $p \gg \epsilon$.

7. CONCLUSIONS

We presented the following results for graphs presented as edge streams.

(1) Algorithm SINGLERANDOMWALK can perform a random walk of length l in $O(\sqrt{l/\alpha})$ passes and $O(n\alpha + \sqrt{l/\alpha})$ space.
(2) Algorithm MULTIPLERANDOMWALK and algorithm MODIFIEDMULTIPLERANDOMWALK can perform K random walks of length l in $\tilde{O}(\sqrt{l/\alpha})$ passes and $\tilde{O}(n\alpha + K\sqrt{l/\alpha} + Kl\alpha)$ space or $\tilde{O}(n\alpha\sqrt{l\alpha} + K(\sqrt{l/\alpha}))$ space, respectively. These algorithms also provide an approach to approximating the probability distribution after a random walk of length l. It follows that every node with probability p in the probability distribution after a random walk of length l can be approximated to an error of $\pm\sqrt{p\epsilon} \pm \epsilon$ using $\tilde{O}(\sqrt{l/\alpha})$ passes and $\min\{\tilde{O}(n\alpha + \frac{1}{\epsilon}\sqrt{l/\alpha} + \frac{1}{\epsilon}l\alpha), \tilde{O}(n\alpha\sqrt{l\alpha} + (1/\epsilon)(\sqrt{l/\alpha}))\}$ space. In particular, the latter algorithm performs better for thresholds $\epsilon \leq \sqrt{l}/n$.
(3) We use this technique and present an approach to determine the ϵ-near mixing time.

Some open questions that arise are:

(1) Can we estimate the distribution of nodes with accuracy $\epsilon = 1/n$ using $O(n)$ space?
(2) Can one prove any space-pass trade-off bounds? The trivial algorithm to calculate the exact distribution after a random walk of length l requires $O(nl)$ in the space$\times$passes product. Our result stated in Corollary 6.3, for a threshold of $\epsilon = 1/n$, also has the same space-pass trade-off.
(3) Is there a lower bound on the number of passes required to perform random walks or estimate distributions when the space allowed is $O(n)$? Alternatively, is there an $O(n)$ space constant pass algorithm?

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable comments.

REFERENCES

ALON, N., MATIAS, Y., AND SZEGEDY, M. 1999. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci. 58,* 1, 137–147.

ANDERSEN, R., CHUNG, F. R. K., AND LANG, K. J. 2006. Local graph partitioning using pagerank vectors. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*. 475–486.

ARMONI, R., TA-SHMA, A., WIGDERSON, A., AND ZHOU, S. 1997. $sl \leq l^{4/3}$. In *Proceedings of the ACM Symposium on Theory of Computing*. 230–239.

BAR-YOSSEF, Z., KUMAR, R., AND SIVAKUMAR, D. 2002. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*. 623–632.

BATU, T., FISCHER, E., FORTNOW, L., KUMAR, R., RUBENFELD, R., AND WHITE, P. 2001. Testing random variables for independence and identity. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*. 442–451.

BHUVANAGIRI, L., AND GANGULY, S. 2006. Estimating entropy over data streams. In *Proceedings of the European Symposium on Algorithms (ESA)*. 148–159.

BHUVANAGIRI, L., GANGULY, S., KESH, D., AND SAHA, C. 2006. Simpler algorithm for estimating frequency moments of data streams. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 708–713.

BRIN, S., AND PAGE, L. 1998. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the 7th International Conference on World Wide Web*. 107–117.

Buriol, L. S., Frahling, G., Leonardi, S., Marchetti-Spaccamela, A., and Sohle, C. 2006. Counting triangles in data streams. In *Proceedings of the ACM SIFMOD-SICACT-SILANT Symposium on Principles of Database Systems.* 253–262.

Cormode, G., and Muthukrishnan, S. 2005. Space efficient mining of multigraph streams. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*. 271–282.

Demetrescu, C., Finocchi, I., and Ribichini, A. 2006. Trading of space for passes in graph streaming problems. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 714–723.

Feige, U. 1997. A spectrum of time-space trade-offs for undirected s-t connectivity. *J. Comput. Syst. Sci. 54,* 2, 305–316.

Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., and Zhang, J. 2005. Graph distances in the streaming model: the value of space. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 745–754.

Feldman, J., Muthukrishnan, S., Sidiropoulos, A., Stein, C., and Svitkina, Z. 2006. On the complexity of processing massive, unordered, distributed data. In CoRR abs/cs/0611108.

Greenwald, M., and Khanna, S. 2001. Space-efficient online computation of quantile summaries. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 58–66.

Guha, S., and McGregor, A. 2006. Approximate quantiles and the order of the stream. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*. 273–279.

Guha, S., and McGregor, A. 2007a. Space-efficient sampling. In *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics (AISTATS)*. 169–176.

Guha, S., and McGregor, A. 2007b. Lower bounds for quantile estimation in random-order and multi-pass streaming. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP)*. 704–715.

Guha, S., McGregor, A., and Venkatasubramanian, S. 2006. Streaming and sublinear approximation of entropy and information distances. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 733–742.

Henzinger, M., Raghavan, P., and Rajagopalan, S. 1999. Computing on data streams. In *External Memory Algorithms,* DIMACS Series in Discrete Mathematics and Theoretical Computer Science. Vol. 50. 107–118.

Indyk, P. 2004. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*. 373–380.

Indyk, P., and Woodruff, D. P. 2003. Optimal approximations of the frequency moments of data streams. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*. 283–292.

Jerrum, M., and Sinclair, A. 1989. Approximating the permanent. *SIAM J. Computing 18,* 6, 1149–1178.

Jowhari, H., and Ghodsi, M. 2005. New streaming algorithms for counting triangles in graphs. In *Proceedings of the 11th International Computing and Combinatories Conference (COCOON)*. 710–716.

Manku, G., Rajagopalan, S., and Lindsay, B. 1999. Randomized sampling techniques for space efficient online computation of order statistics of large datasets. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 251–262.

McGregor, A. 2005. Finding graph matchings in data streams. In *Proceedings of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2005) and 9th International Workshop on Randomization and Computation (RANDOM 2005)*. Lecture Notes in Computer Science, Vol. 3624, Springer, 170–181.

McSherry, F. 2005. A uniform approach to accelerated pagerank computation. In *Proceedings of the 14th International World Wide Web Conference (WWW)*. 575–582.

Sarlos, T., Benczur, A., Csalogany, K., Fogaras, D., and Racz, B. 2006. To randomize or not to randomize: Space optimal summaries for hyperlink analysis. In *Proceedings of the the 15th International World Wide Web Conference (WWW)*. 297–306.

Wicks, J., and Greenwald, A. R. 2007. Parallelizing the computation of pagerank. In *Proceedings of the 5th Workshop on Algorithms and Models for the Web-Graph (WAW)*. 202–208.

Woodruff, D. P. 2004. Optimal space lower bounds for all frequency moments'. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 167–175.

Received October 2008; revised November 2009 and February 2011; accepted February 2011