

MIT Open Access Articles

Walking Randomly, Massively, and Efficiently

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

Citation: Lacki, Jakub, Onak, Krzysztof, Sankowski, Piotr and Mitrovic, Slobodan. 2020. "Walking Randomly, Massively, and Efficiently."

As Published: <https://doi.org/10.1145/3357713.3384303>

Publisher: ACM|Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing

Persistent URL: <https://hdl.handle.net/1721.1/146116>

Version: Final published version: final published article, as it appeared in a journal, conference proceedings, or other formally published context

Terms of use: Creative Commons Attribution 4.0 International license



Walking Randomly, Massively, and Efficiently*

Jakub Łącki
Google Research
USA

Krzysztof Onak
IBM Research
MIT-IBM Watson AI Lab
USA

Slobodan Mitrović
MIT
USA

Piotr Sankowski
University of Warsaw
Poland

ABSTRACT

We introduce a set of techniques that allow for efficiently generating many independent random walks in the Massively Parallel Computation (MPC) model with space per machine strongly sub-linear in the number of vertices. In this space-per-machine regime, many natural approaches to graph problems struggle to overcome the $\Theta(\log n)$ MPC round complexity barrier, where n is the number of vertices. Our techniques enable achieving this for PageRank—one of the most important applications of random walks—even in more challenging directed graphs, as well as for approximate bipartiteness and expansion testing.

In the undirected case, we start our random walks from the stationary distribution, which implies that we approximately know the empirical distribution of their next steps. This allows for preparing continuations of random walks in advance and applying a doubling approach. As a result we can generate multiple random walks of length l in $\Theta(\log l)$ rounds on MPC. Moreover, we show that under the popular 1-vs.-2-CYCLES conjecture, this round complexity is asymptotically tight.

For directed graphs, our approach stems from our treatment of the PageRank Markov chain. We first compute the PageRank for the undirected version of the input graph and then slowly transition towards the directed case, considering convex combinations of the transition matrices in the process.

For PageRank, we achieve the following round complexities for damping factor equal to $1 - \epsilon$:

- $O(\log \log n + \log 1/\epsilon)$ rounds for undirected graphs (with $\tilde{O}(m/\epsilon^2)$ total space),
- $\tilde{O}(\log^2 \log n + \log^2 1/\epsilon)$ rounds for directed graphs (with $\tilde{O}((m + n^{1+o(1)})/\text{poly}(\epsilon))$ total space).

The round complexity of our result for computing PageRank has only logarithmic dependence on $1/\epsilon$. We use this to show that our PageRank algorithm can be used to construct directed length- l random walks in $O(\log^2 \log n + \log^2 l)$ rounds with $\tilde{O}((m + n^{1+o(1)})\text{poly}(l))$ total space.

*The full version of the paper is available at <https://arxiv.org/pdf/1907.05391.pdf>.



This work is licensed under a Creative Commons Attribution 4.0 International License.

STOC '20, June 22–26, 2020, Chicago, IL, USA

© 2020 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-6979-4/20/06.

<https://doi.org/10.1145/3357713.3384303>

CCS CONCEPTS

• **Theory of computation** → **MapReduce algorithms; Massively parallel algorithms; Graph algorithms analysis; Random walks and Markov chains.**

KEYWORDS

random walks, PageRank, Massively Parallel Computation

ACM Reference Format:

Jakub Łącki, Slobodan Mitrović, Krzysztof Onak, and Piotr Sankowski. 2020. Walking Randomly, Massively, and Efficiently. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC '20)*, June 22–26, 2020, Chicago, IL, USA. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3357713.3384303>

1 INTRODUCTION

Computing random walks in graphs is a fundamental algorithmic primitive. Random walks find applications in a plethora of computer science research areas. A non-exhaustive list includes optimal PRAM algorithms for connectivity [39, 40, 55], rating web pages [27, 54], partitioning graphs [1], minimizing query complexity in property testing [21, 23–25, 37, 44, 47, 52], finding graph matchings [36], generating random spanning trees [48], and counting problems [42].

Intuitively, computing random walks, especially independent random walks from all vertices, should be highly parallelizable, since random walks are memoryless. However, even if we start a single random walk from each vertex, after a constant number of steps many of the walks may collide in the same vertex. This is especially problematic in directed graphs, since it is not known how to precompute the vertices where many collisions would happen, other than by simulating length- l walks in l steps or computing the l -th power of the transition matrix, which takes quadratic space.

The focus of this work is on generating a large number of independent random walks in a parallel setting. For undirected graphs, we take advantage of the fact that the stationary distribution is proportional to vertex degrees and can thus be computed in a trivial way. If the starting points of the random walks are distributed according to the stationary distribution, then after any number of steps the distribution of the endpoints is the same. We exploit this to pre-sample continuations of random walks and recursively stitch them together in order to generate length- l walks in $O(\log l)$ parallel rounds.

The situation becomes more complicated for directed graphs (or Markov chains such as the one defining PageRank), since we do not know the stationary distribution a priori and hence cannot

apply this approach directly. Instead, to compute PageRank, we start from the undirected closure \tilde{G} of the input graph G , for which we can generate random walks, using the ideas described above. We then slowly transition from \tilde{G} to G , and gradually update our approximation of the stationary distribution. Roughly speaking, at each step we consider a convex combination of the transition matrices of \tilde{G} and G . This technique, together with its analysis, is the most important and complex technical contribution of the paper.

Another challenge in computing random walks in directed graphs is the fact that the probabilities of some vertices in the stationary distribution π can be as low as $O(1/2^n)$. Hence, sampling $t(v)$ random walks from each vertex v , where $t(v) \geq 1$ is proportional to $\pi(v)$, would result in exponentially many walks from some vertices. This challenge can be addressed by using the Markov chain that is used to define PageRank. In this Markov chain, a random walk that reached a vertex v is either extended with a random outedge of v (with probability $1 - \epsilon$), or jumps to a random vertex of the graph (with probability ϵ). This small change influences the stationary distribution π significantly, since in the modified graph we have $\pi(v) \geq \epsilon/n$ for each vertex v . At the same time, by setting $\epsilon = O(1/l)$, one can guarantee that each random walk does not make a random jump with constant probability, and is thus a random walk in the original graph. Altogether, our ideas lead to an algorithm for computing length- l random walks in directed graphs in $O(\log^2 \log n + \log^2 l)$ rounds and an algorithm for PageRank that uses $\tilde{O}(\log^2 \log n)$ rounds. Both these algorithms require total space that is almost linear in the input size (assuming we only compute random walks of length $l = \text{poly log } n$).

We show that our algorithms can be implemented in the Massively Parallel Computation (MPC) model, which has been extensively studied by the theory community in the recent years [2–6, 10, 11, 14, 22, 31–35, 41, 50, 53]. We use the most challenging space-per-machine regime of the MPC model, in which the space available on each machine is strongly sublinear in the number of vertices of the graph, i.e., at most $n^{1-\Omega(1)}$. This allows for handling large graphs that do not fit onto a single machine, even if they are sparse, which is the case for social networks and the webgraph.

1.1 Our Results

We give new algorithms for sampling independent random walks and show that they can be efficiently implemented in the MPC model. For the formal description of the model see Section 2.1.

We write n and m to denote the number of nodes and edges in the input graph, respectively.

The first result is an algorithm for sampling random walks in undirected graphs, where $\deg(v)$ denotes the degree of a vertex v .

THEOREM 1. *Let G be an undirected graph on n vertices with m edges and $C \geq 1$. Let l be a positive integer such that $l = o(S)$, where S is the available space per machine. There exists an MPC algorithm that samples $\deg(v) \cdot \lceil C \ln n \rceil$ independent random walks of length l starting in v for each vertex v in G . The algorithm runs in $O(\log l)$ rounds and uses $O(Cml \log l \log n)$ total space and strongly sublinear space per machine. If the algorithm has to return only the endpoints of each random walk, the total space complexity can be reduced to*

$O(Cml \log n)$. The algorithm is an imperfect sampler (see Definition 7) that does not fail with probability $1 - n^{-\frac{C}{6}+1}$.

Our algorithm assumes that the length of each random walk is at most the space available per machine, which is n^γ for $\gamma \in (0, 1)$. We believe this assumption is not limiting, since the most interesting regime in applications is $l = n^{o(1)}$, especially $l = O(\text{poly log } n)$.

One of the main results of this paper is an algorithm for sampling random walks in directed graphs. To the best of our knowledge, this is the first $o(l)$ -round algorithm for sampling length- l independent random walks in any distributed or parallel model, other than the trivial algorithm based on matrix exponentiation, which requires quadratic total space.

THEOREM 2. *Let G be a directed graph on n vertices with m edges. Let D and l be positive integers such that $l = o(S)/\log^3 n$, where S is the available space per machine. There exists an MPC algorithm that samples D independent random walks of length l starting in v for each v in G . The algorithm runs in $O(\log^2 \log n + \log^2 l)$ rounds and uses $\tilde{O}\left(m + n^{1+o(1)}l^{3.5} + Dnl^{2+o(1)}\right)$ total space and strongly sublinear space per machine. The algorithm is an imperfect sampler (see Definition 7) that does not fail with probability $1 - O(n^{-1})$.*

We also show that the algorithm for undirected graphs is almost optimal under the 1-vs.-2-CYCLES conjecture, which is the most popular conjecture for showing conditional hardness in the MPC model.

THEOREM 3. *Let $\gamma \in (0, 1)$ be a constant. Consider the MPC model with $O(n^{1-\gamma} \text{poly log } n)$ machines, each having space $O(n^\gamma)$, where n is the number of vertices in the input graph. Each algorithm that can sample $\Theta(\log n)$ independent random walks of length $\Theta(\log^4 n)$ starting at each vertex of the graph requires $\Omega(\log \log n)$ rounds, unless the 1-vs.-2-CYCLES conjecture does not hold.*

By using our random walk sampling primitive, we give an algorithm for computing PageRank in undirected graphs.

THEOREM 4. *Let n be the number of vertices in the input graph. Let $\alpha \in [1/n, 1/4]$ and $\epsilon \in [\log n/o(S), 1]$, where S is the available space per machine. There exists an MPC algorithm that, with probability at least $1 - \frac{5}{n^2}$, computes a $(1 + \alpha)$ -approximate PageRank vector in undirected graphs with jumping probability ϵ in $O(\log \log n + \log 1/\epsilon)$ rounds, using $O\left(\frac{m \log^2 n \log \log n}{\epsilon^2 \alpha^2}\right)$ space.*

Our next result is an algorithm for computing PageRank in directed graphs. This is by far the most technically involved part of the paper. In fact, our algorithm for sampling random walks in directed graphs, from Theorem 2, is a corollary from the lemmas that we develop to obtain the following result.

THEOREM 5. *Let G be a directed graph on n vertices with m edges. Let $\alpha \in [1/n, 1/4]$ and $\epsilon \in [\log^3 n/o(S), 1]$, where S is the available space per machine. There exists an MPC algorithm that, with probability at least $1 - O(\frac{1}{n})$, computes a $(1 + \alpha)$ -approximate ϵ -PageRank vector of G in $\tilde{O}\left(\log^2 \log n + \log^2 1/\epsilon\right)$ rounds, using $\tilde{O}\left(\frac{m}{\alpha^2} + \frac{n^{1+o(1)}}{\epsilon^{3.5} \alpha^2}\right)$ total space and strongly sublinear space per machine.*

This gives an exponential improvement in the number of rounds with respect to the previously known results [27].

Recently, it was shown that computing $\Theta(1)$ -approximate maximum matching, $\Theta(1)$ -approximate minimum vertex cover, and maximal independent set admit $\Omega(\log \log n)$ conditional lower bound on the round complexity in the MPC model [33] in the regime with strongly sublinear space per machine. Hence, $O(\text{poly}(\log \log n))$ -round algorithms seem to be the new complexity benchmark to reach.

Random Walks in PRAM. We show that our algorithm for sampling random walks is generic enough to yield interesting results beyond the MPC model. In particular, we show that it can also be implemented in PRAM. Note that the following theorem gives a nontrivial result whenever $l = \omega(\text{poly} \log n)$.

THEOREM 6. *Let G be a directed graph on n vertices with m edges. Let $1 \leq l \leq n$. There exists an RNC algorithm that uses $O((n + m)^{1+o(1)})$ processors and samples one random walk from each vertex of G . All sampled walks are independent. The algorithm is an imperfect sampler (see Definition 7) that fails with probability $1 - O(n^{-1})$.*

Due to space constraints, we provide the details of the algorithm in the full version of the paper. We could also formulate a similar theorem for computing PageRank, but an NC algorithm for PageRank is already known, since it can be obtained by simply using the power method.

Applications to Property Testing. We show how to use our random walk algorithm to solve other problems in the MPC model. Instead of solving the exact version of the problems, we consider relaxed versions of these problems known from property testing. Our algorithms either show that the graph is close to having a given property or far away from satisfying it. It is unlikely that the exact versions of these problems have $o(\log n)$ -round algorithms, due to the 1-vs.-2-CYCLES conjecture (see Section 4.2 for conjecture statement). For the description of these approximate algorithms, we refer the reader to the full version of the paper.

1.2 Previous Research

Random walks in the streaming model. The problem of generating random walks was considered in a number of streaming and parallel computation papers. The paper of Sarma, Gollapudi, and Panigrahy [26] introduced multi-pass streaming algorithms for simulating random walks. For a single starting point, they can, for instance, simulate single length- l random walk in $\tilde{O}(n)$ space and $\tilde{O}(\sqrt{l})$ passes. The paper by Jin [43] gives algorithms for generating a single random walk from a prespecified vertex in one pass. For directed graphs, the algorithm requires $\Theta(nl)$ space and for undirected graphs, $\Theta(n\sqrt{l})$ space.

Parallel distributed computation. Bahmani, Chakrabarti, and Xin [8] give an MPC algorithm for constructing length- l random walks in directed graphs. Their algorithm runs in $O(\log l)$ rounds, but the walks it generates are *not independent*.

A recent result by Assadi, Sun and Weinstein [6] gives an MPC algorithm for detecting well-connected components in small space per machine and with exponential speed-up over the direct exploration. As a subroutine, the paper presents an algorithm for generating random walks in an undirected regular graph. Note that in regular graphs the stationary distribution is uniform and thus the

problem of generating random walks becomes somewhat simpler. Let us also mention that there exist random walk generating algorithms for the distributed CONGEST model [28]. These algorithms require however a number of rounds that is at least linear in the diameter, which can be $\Omega(\log n)$ even for expanders.

Computing random walks has also been used in PRAM model as a subroutine in algorithms for computing connected components, using a near-linear number of processors [39, 45]. However, in both these algorithms, random walks starting at different vertices are not independent.

PageRank. Since it was introduced in [18, 54], the computation of PageRank has been extensively studied in various settings. We refer a reader to [12, 30, 49] for the early development and theoretical foundations of PageRank. [26] consider PageRank approximation in the streaming setting. As their main result, they show how to compute an l -length random walk in $O(\sqrt{l})$ passes, using $O(n)$ space. By using this primitive, [26] show how to approximate PageRank for directed graphs in $\tilde{O}(M^{3/4})$ passes by using $\tilde{O}(nM^{-1/4})$ space, where M is the mixing time of the underlying graph.

By building on [7], [27] studied PageRank in a distributed model of computation. For directed graphs, they design a PageRank approximation algorithm that runs in $O\left(\frac{\log n}{\epsilon}\right)$ rounds, where ϵ is the jumping probability, i.e., $1 - \epsilon$ is the damping factor. To achieve this bound, they use the fact that w.h.p. a random walk from a vertex jumps to a random vertex within $O\left(\frac{\log n}{\epsilon}\right)$ steps. Moreover, to estimate PageRank it suffices to count the number of random walks ending at a given vertex while ignoring how those random walks reached the vertex. The authors of [27] exploited this observation to show that many PageRank random walks can be simulated in parallel while not over-congesting the network. This approach can be implemented in $O\left(\frac{\log n}{\epsilon}\right)$ MPC rounds. The authors also show how to extend the ideas of [26] and obtain an algorithm for undirected graphs that approximates PageRank in $O\left(\frac{\sqrt{\log n}}{\epsilon}\right)$ rounds. Better round complexity was achieved in the congested clique model. Namely [51] gives $O(\log \log n)$ round algorithm for computing PageRank of an undirected graph. This implies an MPC algorithm that uses the same number of rounds and $\tilde{O}(n)$ space per machine.

Another line of work considered approximate PageRank in the context of sublinear time algorithms [13, 15, 16, 20]. This research culminated in a method for approximating PageRank of a given vertex by examining only $\tilde{O}(\min\{n^{2/3}\Delta^{1/3}, n^{4/5}d^{1/5}\})$ many vertices/arcs, where Δ and d are the maximum and average degree, respectively [15]. It is not clear how to simulate this approach in MPC efficiently for all vertices, in terms of both round and total space complexity.

Bahmani, Chakrabarti, and Xin [8] show how to use their result for generating random walks to get a constant *additive* approximation to Personalized PageRank¹ of each vertex, which can be used to obtain a constant additive approximation to PageRank. Note that, in the case of PageRank, constant additive approximation is a weak

¹In PageRank, a random walk jumps to a random vertex with some probability ϵ in each step. In the Personalized PageRank for vertex v , this random jump is always performed back to v .

guarantee, since an all-zero vector $\vec{0}$ provides a correct constant additive approximation for all but $O(1)$ vertices. In particular, the algorithm of [8] provides non-zero estimates for only $O(\log n)$ vertices; since each vertex has strictly positive PageRank, an estimate of value zero is clearly not a multiplicative approximation. In this paper, we show how to compute a multiplicative approximation of PageRank, and hence provide an estimate for each vertex.

We now comment on why using random walks generated by the algorithm of [8] would require at least quadratic space to obtain multiplicative approximation to PageRank (assuming a standard approach). To compute a length- l random walk from each vertex, the algorithm of [8] samples $O(l)$ random edges from each vertex. These random edges are then used to construct the desired random walks by doubling, i.e., two walks of length 2^i are concatenated to form a walk of length 2^{i+1} . The same random edge sample can be used for multiple walks, which results in the following undesirable behavior.

Consider a star-graph with the center being vertex s and use the algorithm of [8] to construct T random walks of length $O(l)$ from each vertex. Let W be the collection of these walks. Each of the walks in W either starts at s or visits s directly after the first step. Since to construct W the algorithm *in total* samples $O(l \cdot T)$ random edges from s , there exists a set of vertices V' of size $O(l \cdot T)$ such that except for starting vertices all the vertices of W belong to V' . Hence, for $l \cdot T = o(n)$ there exists a vertex different than s that among the walks of W appears $\Omega(nT/(l \cdot T)) = \Omega(n/l)$ many times, and also a vertex different than s that appears only T times (but only as the starting vertex of some of the walks of W).

The standard approach to estimating the PageRank of vertex v is to count the number of occurrences of v as the endpoints of walks in W . To estimate PageRank, the value of l is set to be $O(\log n)$. Therefore, unless $T = \Omega(n/\log n)$, there exist two vertices u and v , both different from s , whose visit counts differ by a super-constant multiplicative factor. However, u and v are symmetric and their PageRanks, both personalized and non-personalized, are equal.

2 PRELIMINARIES

In this paper we consider directed multigraphs, i.e., we allow multiple edges between each pair of vertices. Let $G = (V, E)$ be a directed multigraph. We use $\deg_G^+(v)$ and $\deg_G^-(v)$ to denote the number of outgoing and incoming edges of v respectively. We also define $\deg_G(v) := \deg_G^-(v) + \deg_G^+(v)$. The subscript G is often omitted if it is clear from the context.

A graph $G = (V, E, w)$ is *weighted* if $w : E \rightarrow \mathbb{R}$ is a function assigning real weights to the edges. Note that different edges between the same pair of vertices may have different weights. We extend the definitions for unweighted graphs to weighted graphs in a natural way.

For a weighted graph $G = (V, E, w)$ and $s \in \mathbb{R}$, we define $s \cdot G$ (often abbreviated as sG) to be a graph $G' = (V, E, w')$, where $w'(e) := s \cdot w(e)$ for each $e \in E$. For two weighted graphs $G_1 = (V, E_1, w_1)$ and $G_2 = (V, E_2, w_2)$, we define $G_1 + G_2 := (V, E_1 + E_2, w_1 \cdot w_2)$. Here, $E_1 + E_2$ denotes the *multiset sum* of E_1 and E_2 , i.e., an element of cardinality c_1 in E_1 and c_2 in E_2 has cardinality

$c_1 + c_2$ in the sum. The weights $w_1 \cdot w_2 : E_1 + E_2 \rightarrow \mathbb{R}$ are defined as

$$(w_1 \cdot w_2)(e) = \begin{cases} w_1(e) & \text{if } e \in E_1, \\ w_2(e) & \text{if } e \in E_2. \end{cases}$$

If $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ are unweighted, then $G_1 + G_2 = (V, E_1 + E_2)$.

The transpose of a graph $G = (V, E)$ is a graph $G^T = (V, E^T)$ where $E^T = \{vu \mid uv \in E\}$. A graph is called *undirected* if it is equal to its transpose. We define the *undirected closure* of G , denoted by \bar{G} to be $\bar{G} = G + G^T$. Note that $\deg_{\bar{G}}^+(v) = \deg_G(v)$.

We call a weighted graph $G = (V, E, w)$ *stochastic* if all edge weights are nonnegative and for each $v \in V$, $\sum_{vx \in E} w(vx) = 1$.² Observe that if $G_1 = (V, E_1, w_1)$ and $G_2 = (V, E_2, w_2)$ are stochastic, $x, y \geq 0$, and $x + y = 1$, then $xG_1 + yG_2$ is stochastic as well.

For a stochastic graph $G = (V, E, w)$, we define $T(G) : V \times V \rightarrow \mathbb{R}$ to be the *transition matrix* of G , where $T(G)_{u,v}$ is the total weight of edges uv . For a transition matrix $T(G)$, a *stationary distribution* is a probability distribution $\pi : V \rightarrow \mathbb{R}$, such that $\pi \cdot T(G) = \pi$. Note that the stationary distribution may not be unique. A stationary distribution of a stochastic graph is a stationary distribution of its transition matrix.

We say that a vertex $v \in V$ is *dangling* if $\deg_G^+(v) = 0$. For an unweighted graph $G = (V, E)$ with no dangling vertices, we write $\text{RW}(G)$ to denote the weighted graph obtained by assigning a weight of $1/\deg_G^+(v)$ to each outgoing edge of v . Note that $\text{RW}(G)$ is stochastic.

A *walk* in a graph G is a sequence of edges $u_1v_1, u_2v_2, \dots, u_kv_k$, such that for $1 \leq i < k$, $v_i = u_{i+1}$. The *length* of a walk is the number of edges of which it consists.

For a stochastic graph $G = (V, E, w)$, a *random walk* in G of length k starting in $s \in V$ is a walk $W = u_1v_1, u_2v_2, \dots, u_kv_k$ in G , where $s = u_1$, which is constructed with the following algorithm. For each $1 \leq i \leq k$, the edge u_iv_i is chosen independently at random among all outgoing edges of u_i . The probability of choosing a particular outgoing edge is equal to the edge weight.

2.1 The MPC Model

We design algorithms for the model introduced by Karloff, Suri, and Vassilvitskii [46] and refined in later works [2, 9, 38]. We refer to it as *massively parallel computation* (MPC), which is similar to the name in Beame et al. [9].

This model captures main aspects of modern parallel systems, where we have M machines and each of them has S words of space. The total amount of space available on all the machines should not be much greater than the size of the input. In the case of graph algorithms studied here the input is a collection E of edges and each machine receives approximately $|E|/M$ of them before the computation starts.

In the MPC model, the computation proceeds in *rounds*. During the round, each machine first processes its local data without communicating with other machines, and then machines exchange messages. When sending a message the machine specifies a unique recipient of this message. Moreover, we require that all messages sent and received by each machine in each round fit into local

²Note that we slightly abuse notation and each outgoing edge of v corresponds to a separate summand, even in the presence of parallel edges.

memory of this machine. Hence, the total length of these messages is bounded by S .³ This implies that the total communication of the MPC model is bounded by $M \cdot S$ in each round. Each message is placed in the locals space of the receiving machine and can be processed by it in the next round.

At the end of the computation, machines collectively output the solution, i.e., the solution is formed by the union of the outputs of all the machines. The data output by each machine has to fit in its local memory. Hence again, each machine can output at most S words.

Possible Values for S and M . A typical assumption in the MPC model is that S , the space per machine, is of order $\Theta(N^{1-\epsilon})$, where N is the size of the input and $\epsilon \in (0, 1)$ is a constant. The total space has to fit the input, and therefore $M \geq N/S$. In the design of algorithms, we hope to make the total space $M \cdot S$ as close to N as possible. The motivation for this is that for processing large data sets, we are unlikely to have significantly more space than the input size. We specify the total required space when we describe each of our algorithms.

In this paper, the focus is on graph algorithms for which it is helpful to parameterize the space per machine as a function of n , the number of vertices in the input graph. In this paper, we focus on the space-per-machine regime that is strongly sublinear in the number of vertices. More precisely, we assume that $S = \Theta(n^\gamma)$ for a fixed constant $\gamma \in (0, 1)$. This is the most interesting space-per-machine regime (compared to $S = \Theta(n)$ and $S = n^{1+\Theta(1)}$), since it allows for distributing computation across several machines, even for very large graphs that are sparse, i.e., have a number of vertices that is near-linear in the number of vertices. This regime is interesting also due to the popular 1-vs.-2-CYCLES conjecture. This conjecture implies that in this regime, many graph problems cannot be solved in $o(\log n)$ rounds (see Section 4.2). This is contrary to the performance of our algorithms, which for a wide range of interesting parameters, run in $O(\text{poly}(\log \log n))$ rounds.

When describing our algorithms, we do not specify the dependence on γ , which is in the worst case $\text{poly}(1/\gamma)$ and in any reasonable practical setting, $\gamma \geq 1/2$.

2.2 PageRank

Let G be a directed graph and let $\epsilon \in (0, 1)$. PageRank [18] is the stationary distribution of the following Markov chain on the vertices of G . At a given vertex v , with probability ϵ , the next vertex is selected uniformly at random from the set of all vertices of G , and with probability $1 - \epsilon$, it is selected uniformly at random from the heads of outedges of v . Note that PageRank is unique, because the Markov chain described above is ergodic. $1 - \epsilon$ is called the *damping factor*.

3 OVERVIEW OF OUR TECHNIQUES

In order to speed up the generation of a random walk, one may be tempted to generate in parallel its different sections and then stitch them together. This approach becomes challenging with limited space if, for instance, one wants to generate a large number of

random walks that all start from the same vertex. Unfortunately, until we know where the walks are after k steps, it is difficult to limit the number of continuations corresponding to the consecutive steps— $k + 1$, $k + 2$, and so on—that we have to consider. This seems to be an issue that many, if not all, attempts at generating random walks with limited space encounter [26].

However, if starting points of random walks are sampled from the stationary distribution, the distribution after any number of steps is the same. (This observation was previously used by Censor-Hillel et al. [19] to construct bipartiteness testers in constant-degree graphs for the CONGEST model.) Hence we can sample only slightly more mid-points from the same stationary distribution and recursively generate random walks from them. The key observation is that the stationary distribution in undirected graphs is known in advance, i.e., it is proportional to vertex degrees. This approach enables generating many fully independent random walks of length l from each vertex in $\Theta(\log l)$ rounds of MPC using space near-linear in the size of the input. We present this idea in detail in Section 4. We also argue that this round complexity is tight under the 1-vs.-2-CYCLES conjecture (see Section 4.2).

3.1 Random Walks and PageRank in Directed Graphs

The problem of generating random walks is much more challenging in directed graphs. We do not know a priori a stationary distribution, so it is not possible to directly apply the previous approach. Namely, our sampler for undirected graphs crucially uses the fact that we know the stationary distribution in advance and the probabilities in this distribution are not too small. This is because the number of random walks sampled from each vertex has to be both proportional to the stationary distribution and large enough to obtain concentration guarantees.

In general directed graphs these assumptions do not hold. Some vertices in the stationary distribution may have small or even zero probability. One can also show that even an exponential number of samples (i.e., $2^{\Theta(n)}$) may not be enough to estimate the stationary distribution for some vertices. Hence, even if we knew the stationary distribution π and wanted to compute $t(v)$ random walks from each vertex v , such that $t(v) \geq 1$ is proportional to $\pi(v)$, we would need to compute an exponential number of walks overall. In fact, in some computational models, there is a known separation between the difficulty of generating random walks for directed and undirected graphs [43].

If the outdegrees in the graph are bounded by $\text{poly} \log n$ we can use the following simple approach (see full version of this paper for a more detailed description). For each vertex v , we find all vertices whose distance from v is at most $\epsilon \log n / \log \log n$. Once we know this set, we are able to simulate $\epsilon \log n / \log \log n$ steps of a random walk in a single round. At the same time the assumption on the outdegrees allows us to bound the total space usage. This approach does not generalize to the case when the outdegrees are arbitrary or we want to generate random walks of length $\omega(\log n)$.

Instead of dealing with the problem of sampling random walks in directed graphs, we first consider the specific case of computing PageRank. The starting point of our approach is the algorithm of [17] that we review in Section 5. At a high level, the algorithm boils

³This for instance allows a machine to send a single word to $S/100$ machines or $S/100$ words to one machine, but not $S/100$ words to $S/100$ machines if $S = \omega(1)$, even if the messages are identical.

down to computing from each vertex $O(\log n/\epsilon)$ random walks of length $O(\log n/\epsilon)$ in a graph G_ϵ , which is defined by

$$G_\epsilon = (1 - \epsilon)RW(G) + \frac{\epsilon}{n}J,$$

where J is complete directed graphs with self loops. In other words, we consider walks that with probability $1 - \epsilon$ perform a single step of a random walk on G , and with probability ϵ jump to a random vertex in G . Note that PageRank is exactly the stationary distribution of G_ϵ .

Note that in order to make G_ϵ well defined, we need to assume that G does not contain *dangling vertices*, i.e., vertices without out-edges. The usual approaches for handling graphs with dangling vertices are discussed in the full version of this paper. In particular, we show that the two most typical approaches: adding self-loops and restarting the walks, are equivalent up to a simple transformation. To our surprise this relation was not previously observed, e.g., [29] argues why one method is better than the other.

Using G_ϵ instead of G for sampling random walks resolves one of the challenges. Namely, it follows from the definition of PageRank that in the stationary distribution of G_ϵ , the probability of each vertex is at least ϵ/n . Hence, by sampling $\Theta(n \log n)$ random walks, we are actually able to approximate the stationary distribution. This observation was already used by Das Sarma et al. [27] to obtain an $O(\log n)$ -round algorithm for directed and $O(\sqrt{\log n})$ round algorithm for undirected graphs.

However, we are still left with the other difficulty: we do not know the stationary distribution. We overcome it by using a novel sampling technique, which is the main technical contribution of the paper and may be of independent interest. In the remaining part of this section, we give an overview of the technique.

The core part of our algorithm is a procedure for adjusting sampling probabilities. Let D_1 and D_2 be two discrete probability distributions that are, roughly speaking, similar, i.e., the elementary events are assigned similar probabilities in both distributions. The procedure, given a random sample from D_1 either produces a sample from D_2 or fails. As long as D_1 does not differ much from D_2 , the failure probability is small. We give two implementations of the procedure, each of which is well suited for some specific distributions D_1 and D_2 . The easier implementation is based on *rejection sampling*. This means that the procedure has the property that it either returns the sample it gets or fails. As a result, the procedure never actually needs to sample from any of the distributions D_1 or D_2 . It actually only needs to make a single Bernoulli trial in order to decide whether to fail. This is a very useful property, as in our case sampling a random walk is much more expensive than doing a Bernoulli trial.

Let us now describe our sampling technique using the above idea. Recall that \bar{G} denotes the undirected closure of G . We note that by using our undirected sampler (Algorithm 1) we can efficiently sample PageRank walks in \bar{G} . Hence we are going to first sample PageRank walks in \bar{G} and then gradually shift towards PageRank walks in G by adjusting transition probabilities.

The main technical challenge is to overcome the fact that even small changes to the transition probabilities can cause large changes to the stationary distribution. For instance, they can multiplicatively accumulate $\Theta(n)$ times along a long directed path. Therefore, in

general, we cannot use a stationary distribution computed in a given step for sampling walks in the following step, even if the difference in the transition probabilities is small. Instead, we reinterpret walks directly, using the procedure for adjusting sampling probabilities. More specifically, in each step, we sample slightly more walks, and then use rejection sampling to obtain random walks in the modified graph, loosing some of them in the process. Finally, we use the resulting walks to estimate the stationary distribution for the next step.

Let us now describe this idea more formally. Consider a sequence $\bar{G} = G_1, G_2, \dots, G_k = G$ of graphs, where, informally speaking, each G_i is a mixture of \bar{G} and G . (In the algorithm we are going to use $k = \Theta(\log \log n)$.) Formally, $G_i = (k-i)/(k-1)\bar{G} + (i-1)/(k-1)G$.

Our algorithm computes PageRank walks in G_i for $i = 1, \dots, k$. The first step is easy: as we noted above, computing PageRank walks in G_1 can be done using Algorithm 1. In each of the remaining steps the algorithm starts with PageRank walks in G_i . Then, it uses the procedure for adjusting sampling probabilities to obtain PageRank walks in G_{i+1} . However, the number of random walks in G_{i+1} obtained this way is significantly smaller than the number of walks in G_i that we had at the beginning of the step. Still, the number of walks is large enough to estimate the stationary distribution of PageRank walks in G_{i+1} . The algorithm then uses this estimated stationary distribution to compute more PageRank walks in G_{i+1} , which allows the process to continue.

Intuitively, the first steps of this process are the most challenging due to potential existence of vertices, whose degrees in G and \bar{G} differ significantly. As an example, consider a vertex v in G with a single outgoing edge vw and $n-1$ incoming edges. A random walk in \bar{G} of length 1 starting at v (that is, a random edge incident to v) is the edge vw only with probability $O(1/n)$. Hence, the rejection sampling would fail with a very large probability.

To alleviate that, we first transform our input graph into a directed graph such that for each vertex v it holds $c \cdot \deg^+(v) \geq \deg(v)$, where c is a constant that we set later. We call such graphs *c-balanced*. This transformation is explained in Section 5.2. We show how to compute PageRank of *c*-balanced graphs in Section 5.1.

In the process of transforming an input to a *c*-balanced graph, each edge is replaced by a path of length $\log n$. This means that a PageRank walk of length k in the input graph corresponds to a PageRank walk of length $k \log n$ in the transformed *c*-balanced one. As a result, informally speaking, in a *c*-balanced graph PageRank walks with jumping probability ϵ jump to random vertices more often than the corresponding walks in the original graph (since traversing a single edge corresponds to roughly $\log n$ steps in the modified graph). Hence, if we applied the algorithm unchanged to the transformed algorithm, we would effectively compute $(\epsilon \log n)$ -PageRank walks.

A natural idea is to run our algorithm for $\epsilon' = \epsilon/\log n$, but this would increase the round complexity or space usage of our algorithm significantly. Instead, we reuse the idea of gradually adjusting the transition matrix that we use for sampling random walks. We start with jumping probability of $1/2$ and then move towards $\epsilon/\log n$ in steps, in that way ensuring that the space requirement and round complexity is as desired. This approach is presented in Section 5.3.

Once we obtain PageRank walks for a c -balanced graph with respect to jumping probability $\epsilon/\log n$, in Section 5.4 we show how to map those walks back to the input graph.

Finally, we observe that the algorithm for computing random walks in G_ϵ can be used to compute random walks in G . Assume we are interested in random walks of length l . Then, each walk in G_ϵ for $\epsilon = 1/l$ does not contain any random jumps with constant probability and is thus a random walk in G . Hence, by sampling sufficiently many random walks in $G_{1/l}$ we are able to compute random walks in G . The key property here is that the round complexity of sampling random walks in G_ϵ is $\tilde{O}(\log^2(1/\epsilon) + \log^2 \log n)$, so as long as $l = \text{poly log } n$, the overall round complexity is $O(\text{poly log log } n)$.

4 SAMPLING RANDOM WALKS

In this section we show algorithms for sampling random walks from a given stochastic graph G . The algorithms require that we know (at least approximately) the stationary distribution of G . This is easy in the case when we deal with undirected graphs, that is $G = \text{RW}(G_U)$, where G_U is an undirected graph. In this case the stationary distribution of G is given by

$$\pi(v) = \frac{\deg_G^+(v)}{2m}. \quad (1)$$

Hence, if we sample the starting point of a random walk from π , then after any number of steps the endpoint will follow distribution π as well. This allows us to use doubling. Since the number of random walks ending in each vertex is (in expectation) the same as the number of random walks starting in each vertex, we can pair up these random walks and stitch together each pair of walks of length k into one walk of length $2k$. The pseudocode of our algorithm is given as Algorithm 1.

Algorithm 1 Given $G, l, t_0, \dots, t_{\lceil \log l \rceil}$, sample $t_{\lceil \log l \rceil}(v)$ random walks of length l according to T from each $v \in V(G)$.

```

1: function RANDOMWALKS( $G, l, t$ )
2:   for all  $v \in V(G)$  in parallel do
3:     Generate  $t_0(v)$  length 1 random walks in  $G$  starting in  $v$ .
     Let  $W_0(v)$  be the set of these walks.
4:   for  $i \leftarrow 1 \dots \lceil \log l \rceil$  do
5:     for all  $v \in V$  in parallel do
6:       Select  $t_i(v)$  random walks from  $W_{i-1}(v)$ . Let that
       set be  $U_i(v)$ .
7:       For each walk  $w \in U_i(v)$ , consider its endpoint  $u$ .
       Ask  $u$  to extend  $w$  by a yet unused walk from  $W_{i-1}(u) \setminus U_i(u)$ .
       Let  $W_i(v)$  denote the set of all these extended walks originating
       at  $v$ . If  $u$  does not have unused walks anymore, the algorithm
       fails.
8:   For each  $v \in V$  truncate walks in  $W_{\lceil \log l \rceil}(v)$  to length  $l$ .
9:   Return  $W_{\lceil \log l \rceil}(v)$  for each  $v \in V$ 

```

The algorithm takes the following parameters. G is the input graph, which has to be stochastic, l is the desired walk length and $t_i(v)$ controls the number of walks starting in vertex v that we would like to sample in the i -th iteration of the algorithm. Note that the algorithm requires that t_i has certain properties, e.g., for

undirected graphs we show that the algorithm works for $t_i(v)$ being proportional to $\deg_v^+(G)$. Also, for a fixed v the sequence $t_0(v), t_1(v), \dots$ has to fulfill certain properties.

In the ideal scenario for each vertex the number of random walks that start and finish in each vertex are equal to the expected value. In such case, in each step we could match all walks into pairs and obtain two times fewer walks of twice the length. However, the numbers may diverge from the expected values and thus we need to sample a bit more random walks to ensure that there are enough of them with high probability. We set $t_i(v) = \deg_G^+(v) \lceil C \log n \cdot k_i \rceil$, whereas the sequence k_i controls how many more walks we sample in each step. A simple solution is to set $k_i = 2^{2^{\lceil \log l \rceil - 2i}}$, which implies that $k_{\lceil \log l \rceil} = 1$ and $k_i = k_{i-1}/4$. By doing so, in each step, in expectation we have twice as many walks as we really need, and it is easy to show that the number of walks is sufficient with high probability. For the proof we are going to use fewer walks and thus slightly reduce the space complexity.

Observe that if Algorithm 1 never failed, it would have generated independent random walks. However, when many walks collide, i.e., end in the same vertex, the algorithm is forced to fail. This means that we get a random sample from a modified distribution in which the probability of some elements is decreased. This fraction on which the algorithm fails will be very small. We formalize this notion using the following definition.

Definition 7. Let (X, p) be a discrete probability space. An imperfect sampler for (X, p) is an algorithm that returns samples from a probability space $(X \cup \{\text{fail}\}, p')$, such that $p'(x) \leq p(x)$ for all $x \in X$. The failure probability of the sampler is $p'(\text{fail})$.

We are going to construct samplers where $p'(\text{fail})$ is arbitrarily small. Note that an algorithm \mathcal{A} using a (perfect) sampler for a probability space (X, p) can be naturally translated to an algorithm \mathcal{A}' using an imperfect sampler. Unless the sampler fails, \mathcal{A}' produces the same result as \mathcal{A} .

We now define the sequence k_i that we use

$$k_0 = 2^{\lceil \log l \rceil + 6} \cdot (\lceil \log l \rceil + 6) \quad (2)$$

$$k_{i-1} = 2k_i + \sqrt{k_i}. \quad (3)$$

The following bounds can be proven for k_i .

Lemma 8. For $0 \leq i \leq \lceil \log l \rceil$, we have

- (i) $k_i \geq 2^6$,
- (ii) $k_i \leq 2^{\lceil \log l \rceil - i + 6} \cdot (\lceil \log l \rceil + 6)$.

PROOF. In order to show (i) we will prove that

$$k_i \geq 2^{\lceil \log l \rceil - i + 6} \cdot (\lceil \log l \rceil - i + 6). \quad (4)$$

We will show (4) by induction on i .

For $i = 0$, (4) follows by definition of k_0 .

Assume now that $k_j \geq 2^{(\lceil \log l \rceil) - j + 6} \cdot (\lceil \log l \rceil - j + 6)$ for each $j \leq i - 1$, and we want to prove that the inequality holds for $j = i$ as well. Recall that $k_{i-1} = 2k_i + \sqrt{k_i}$. Towards a contradiction, assume that $k_i < 2^{\lceil \log l \rceil - i + 6} \cdot (\lceil \log l \rceil - i + 6)$. For the sake of brevity, define $t \stackrel{\text{def}}{=} \lceil \log l \rceil - i + 6$. Then, we have

$$\begin{aligned} k_{i-1} &= 2k_i + \sqrt{k_i} < 2^{t+1} \cdot t + \sqrt{2^t \cdot t} \\ &< 2^{t+1} \cdot t + 2^t < 2^{t+1} \cdot (t+1) \leq k_{i-1}, \end{aligned}$$

which is a contradiction. Hence, (4) holds, and (i) follows.

As for (ii) we have $k_{i-1} = 2k_i + \sqrt{k_i} \geq 2k_i$. Hence, $k_i \leq 2^{-1}k_{i-1} \leq 2^{-i}k_0 = 2^{\lceil \log l \rceil - i + 6} \cdot (\lceil \log l \rceil + 6)$. \square

We now show that [Algorithm 1](#) can be used to sample random walks in undirected graphs. The proof is a relatively simple application of Chernoff bound.

Lemma 9. *Let G be a stochastic graph such that $G = RW(G_U)$ for some undirected graph G_U , $l, C \geq 1$, $l = o(S)$, $t_i(v) = \deg_G^+(v) \lceil C \log n \cdot k_i \rceil$, where k_i is given by (2) and (3). Then $\text{RANDOMWALKS}(G, l, t)$ ([Algorithm 1](#)) does not fail with probability at least $1 - n^{-\frac{C}{6} + 1}$.*

The proof of [Lemma 9](#) follows by an application of Chernoff bound and is given in the full version of this paper.

Lemma 10. *Assume that G, l, t , are defined as in [Lemma 9](#). Then $\text{RANDOMWALKS}(G, l, t)$ ([Algorithm 1](#)) can be implemented to run in $O(\log l)$ MPC rounds using $O(Cml \log l \log n)$ total space.*

PROOF. The space is upper-bounded by $\max_{1 \leq i \leq \lceil \log l \rceil} O(\sum_{v \in V} t_i(v) \cdot 2^i \cdot k_i)$. [Lemma 8](#) (ii) implies

$$\begin{aligned} & \sum_{v \in V} t_i(v) \cdot 2^i \cdot k_i \\ &= \sum_{v \in V} \deg^+(v) \cdot \lceil C \ln n \rceil \cdot 2^i \cdot 2^{\lceil \log l \rceil - i + 6} \\ &= O(Cml \log l \ln n). \end{aligned}$$

The MPC implementation is provided in the full version. \square

4.1 Sampling Random Walks Given Approximate π

While computing the stationary distribution π is easy for undirected graphs, the problem is significantly more involved if we consider directed graphs. In this section we show how to use [Algorithm 1](#) to sample random walks in a directed graph given an approximation $\tilde{\pi}$ of π . Our approach requires that π is bounded away from 0 on each vertex. In [Section 5](#), we show how to use this sampling procedure for computing PageRank and for sampling random walks in directed graphs. We now state our main result for sampling random walks given $\tilde{\pi}$, while its proof is deferred to the full version of this paper.

Lemma 11. *Let G be a stochastic graph. Let π be the stationary distribution of G , and ϵ be such that $\pi(v) \geq \epsilon/n$ for all v . Let $l, C \geq 1$ and $l = o(S)$. Finally, let $\alpha \in (0, 1/4]$ be a parameter, and let $\tilde{\pi}$ be a probability distribution on V such that $|\tilde{\pi}(v) - \pi(v)| \leq \alpha\pi(v)$ for all v . Define $k_i = (2 + 4\alpha)^{\lceil \log l \rceil - i}$ and $t_i(v) = \lceil C\tilde{\pi}(v)n \ln n \cdot k_i \rceil$. Then, $\text{RANDOMWALKS}(G, l, t)$ ([Algorithm 1](#)) has the following properties:*

- (i) *The algorithm is an imperfect sampler (see [Definition 7](#)) for generating $\lceil C\tilde{\pi}(v)n \ln n \rceil$ length l random walks starting from each vertex $v \in V$. The failure probability is at most $n^{-\frac{C\alpha\epsilon}{3} + 2} e^2$.*
- (ii) *The algorithm can be executed in $O(\log l)$ MPC rounds.*
- (iii) *The algorithm requires $O(m + Cl^{1+2\alpha}n \ln n)$ space.*

4.2 Lower Bound

In this section we show that our algorithm for sampling random walks in undirected graphs is conditionally optimal under the popular 1-vs.-2-CYCLES conjecture [9, 56]. The conjecture states that

any algorithm in the MPC model which distinguishes between a graph being a cycle of length n from a graph consisting of two cycles of length $n/2$, and uses $O(n^\gamma)$ space per machine and $O(n^{1-\gamma})$ machines requires $\Omega(\log n)$ rounds.

THEOREM 3. *Let $\gamma \in (0, 1)$ be a constant. Consider the MPC model with $O(n^{1-\gamma} \text{poly } \log n)$ machines, each having space $O(n^\gamma)$, where n is the number of vertices in the input graph. Each algorithm that can sample $\Theta(\log n)$ independent random walks of length $\Theta(\log^4 n)$ starting at each vertex of the graph requires $\Omega(\log \log n)$ rounds, unless the 1-vs.-2-CYCLES conjecture does not hold.*

The proof is based on the fact that by running $\Theta(\log n)$ random walks of length $\Theta(\log^4 n)$ from a vertex v one discovers $\Theta(\log^2 n)$ nearest vertices to v with high probability. We refer a reader to the full version for details.

5 PAGERANK

In this section we show MPC algorithms for computing PageRank both in undirected and directed graphs, that is we prove [Theorem 4](#) and [Theorem 5](#). As a corollary we obtain an algorithm for sampling random walks in directed graphs ([Theorem 2](#)).

We will use the following most basic algorithm for estimating PageRank using random walks. Let $0 \leq \epsilon \leq 1$ be a parameter. We will sample random walks from a stochastic graph

$$G_\epsilon = (1 - \epsilon)G + \frac{\epsilon}{n}J, \quad (5)$$

where J is a complete directed graph on the vertex set $V(G)$ (containing a self loop in every vertex). In other words, with probability ϵ we jump to a random vertex, and with probability $1 - \epsilon$ we walk according to edges of G .

Definition 12 (Jump transition). *Let G_ϵ be the stochastic graph defined by (5). Then jump transition refers to the transition performed within the graph J .*

Consider the transition matrix $T_\epsilon = T(G_\epsilon)$. The stationary distribution π_ϵ of T_ϵ satisfies $T_\epsilon \pi_\epsilon = \pi_\epsilon$. Hence, [Eq. \(5\)](#) implies

$$(I - (1 - \epsilon)T(G))\pi_\epsilon = \frac{\epsilon}{n}\vec{1}. \quad (6)$$

The crucial property of π_ϵ is that the probabilities of ending in a given vertex do not decrease much relatively to decrease of ϵ , as shown by the following lemma.

Lemma 13. *For any $0 < \delta \leq 1$, we have $\pi_{\epsilon \cdot \delta} \geq \delta \cdot \pi_\epsilon$, where inequality is taken over all coordinates.*

This result follows from the Taylor expansion of π_ϵ , obtained by transforming [Eq. \(6\)](#).

$$\pi_\epsilon = \frac{\epsilon}{n} \sum_{i=0}^{\infty} ((1 - \epsilon)T(G))^i \vec{1}. \quad (7)$$

The next follows from the observations that $\pi_1(v) = \frac{1}{n}$.

Corollary 14. *For any $0 < \epsilon \leq 1$ and $v \in V$, $\pi_\epsilon(v) \geq \frac{\epsilon}{n}$.*

The Taylor expansion [Eq. \(7\)](#) suggests the following algorithm for estimating PageRank (see e.g., [17]).

Algorithm 2 An algorithm for approximating the PageRank with damping factor $1 - \epsilon$ by using a set of random walks W .

```

1: function STATIONARYDISTRIBUTION( $W, \epsilon$ )
2:   for all  $v \in V$  in parallel do
3:     Remove from  $W$  all but  $K = \left\lceil \frac{9 \ln n}{\epsilon \alpha^2} \right\rceil$  walks starting in  $v$ .
     If  $W$  does not contain enough walks, then “fail”.
4:   Truncate each walk in  $W$  just before the first jump transition
     (see Definition 12).
5:   for all  $v \in V$  in parallel do
6:      $n_v \leftarrow$  number of the walks from  $W$  ending in  $v$ .
7:      $\tilde{\pi}(v) \leftarrow \frac{n_v}{Kn}$ .
8:   Return  $\tilde{\pi}$ 

```

Algorithm 3 An algorithm for approximating PageRank using random walks.

```

1: Sample a set  $W$  of  $K = \left\lceil \frac{9 \ln n}{\epsilon \alpha^2} \right\rceil$  random walks starting from
   each vertex of  $G_\epsilon$  of length  $l = \left\lceil \frac{9 \ln n}{\epsilon} \right\rceil$ .
2: Return STATIONARYDISTRIBUTION( $W, \epsilon$ )

```

Let us prove the approximation ratio obtained by Algorithm 3. Note that the lower bounds for α and ϵ are not limiting, since the interesting values for both parameters are constant.

Lemma 15. *Let $\alpha \in [1/n, 1/4]$, $\epsilon \in [1/n, 1]$, and $0 < \alpha < 1$. Denote by $\tilde{\pi}$ the output of Algorithm 3. Then $|\tilde{\pi}_\epsilon(v) - \pi_\epsilon(v)| \leq \alpha \pi_\epsilon(v)$ for all $v \in V$ (i.e., $\tilde{\pi}$ is $(1 + \alpha)$ -approximation of π_ϵ) with probability at least $1 - \frac{4}{n^2}$.*

THEOREM 4. *Let n be the number of vertices in the input graph. Let $\alpha \in [1/n, 1/4]$ and $\epsilon \in [\log n / o(S), 1]$, where S is the available space per machine. There exists an MPC algorithm that, with probability at least $1 - \frac{5}{n^2}$, computes a $(1 + \alpha)$ -approximate PageRank vector in undirected graphs with jumping probability ϵ in $O(\log \log n + \log 1/\epsilon)$ rounds, using $O\left(\frac{m \log^2 n \log \log n}{\epsilon^2 \alpha^2}\right)$ space.*

PROOF. We set $K = \left\lceil \frac{9 \ln n}{\epsilon \delta^2} \right\rceil$, $l = \left\lceil \frac{9 \ln n}{\epsilon} \right\rceil$ and $C = \frac{6}{\epsilon \alpha^2}$. We first execute Algorithm 1 with C and l . Next, we give sampled walks to Algorithm 3 with K and l .

Space and round complexity. By Lemma 10 the algorithm requires $O(Cml \log l \log n) = O\left(\frac{m \log^2 n \log \log n}{\epsilon^2 \alpha^2}\right)$ total space and $O(\log l) = O(\log \log n + \log 1/\epsilon)$ rounds.

Success probability. By Theorem 1 the algorithm fails with probability at most $n^{-\frac{C}{3}+3} = n^{-\frac{2}{\epsilon \alpha^2}+3} \leq n^{-32+3} = n^{-29}$. By Lemma 15 we obtain $(1 + \alpha)$ -approximation of π with probability at least $1 - \frac{4}{n^2}$. Hence, the final success probability is at least $1 - \frac{5}{n^2}$. \square

5.1 Directed Balanced Graphs

The first step towards our algorithm for directed graphs is considering balanced directed graphs. A directed graph G is called c -balanced when for all $v \in V$ we have $c \deg^+(v) \geq \deg(v)$. In particular in c -balanced graphs there are no dangling vertices, i.e.,

vertices that do not have any out edge. The idea is to first consider \bar{G} – the undirected closure of G . In \bar{G} we can compute long walks fast and then gradually move towards directed graph G . The c -balanced property allows us to argue that each edge from a random walk in \bar{G} is with probability at least $1/c$ directed according to G , enabling us to prove the following result.

THEOREM 16. *Let G be a c -balanced graph. Let $\alpha \in [1/n, 1/4]$, $\epsilon \in [\log n / o(S), 1]$, and $\delta \in (0, 1]$ such that $\delta^{-1} \in \mathbb{N}$. There exists an MPC algorithm that computes $(1 + \alpha)$ -approximate ϵ -PageRank vector of G in $O(\delta^{-1}(\log \log n + \log 1/\epsilon))$ rounds using a total space of $O\left(\frac{m \ln^2 n \ln \ln n}{\epsilon^2 \alpha^2} + n^{18 \frac{\epsilon \delta}{\epsilon}} \frac{n \ln^{2.5} n}{\epsilon^{3.5} \alpha^2}\right)$ and strongly sublinear space per machine. The algorithm succeeds with probability at least $1 - \frac{12}{\delta n^2}$.*

For this we need few more definitions. We are going to sample random walks from the stochastic graph defined as

$$G_{\epsilon, \sigma} = (1 - \epsilon)\sigma \text{RW}(\bar{G}) + (1 - \epsilon)(1 - \sigma) \text{RW}(G) + \frac{\epsilon}{n} J. \quad (8)$$

By $\pi_{\epsilon, \sigma}$ we denote the stationary distribution of $G_{\epsilon, \sigma}$. Our algorithms will be using $G_{\epsilon, \sigma}$, but the graph corresponding to J will be constructed implicitly. That is, instead of constructing graph J , which contains n^2 edges, each vertex v of $G_{\epsilon, \sigma}$ will hold a value ϵ/n implying that v has an edge of weight ϵ/n to each vertex.

We also note that $G_{\epsilon, \sigma}$ can be constructed in $O(1)$ MPC rounds. Namely, we first broadcast ϵ and σ to each machine, which can be done in $O(1)$ rounds as described in [38]. Then, each edge is copied and annotated so to construct $(1 - \epsilon)\sigma \text{RW}(\bar{G}) + (1 - \epsilon)(1 - \sigma) \text{RW}(G)$. Finally, each vertex is annotated by ϵ which, as described, suffices to implicitly construct $\frac{\epsilon}{n} J$.

Observation 17. *A stochastic graph as defined by (8) can be implicitly constructed in $O(1)$ MPC rounds.*

We now define three transition types that capture different components of stochastic graphs.

Definition 18 (Transition types). *Let $G_{\epsilon, \sigma}$ be the stochastic graph as defined by (8). Each edge of $G_{\epsilon, \sigma}$ originates from one of the graphs G, \bar{G} and J . We call an edge of $G_{\epsilon, \sigma}$ a*

- directed transition, if it originates from G , and
- undirected transition, if it originates from \bar{G} , and
- jump transition, if it originates from J .

In the following we assume that each edge of each walk in $G_{\epsilon, \sigma}$ has its edges labeled with the transition types defined above.

In the main algorithm of this section (see Algorithm 4), we start from $\sigma = 1$ and then gradually decrease the value to obtain $\sigma = 0$. This sequence is defined as

$$\sigma_{j+1} = \sigma_j - \delta,$$

for $1 \leq j \leq \delta^{-1}$. (We will set δ so that δ^{-1} is an integer.)

5.1.1 Algorithms. We now describe our algorithms used to compute approximate PageRank for c -balanced graphs. The main algorithm is `PAGERANKOFBALANCEDGRAPHS` (see Algorithm 4) that essentially repeats Algorithm 3 (see the steps within loop Line 4) δ^{-1} many times. This loop implements the gradual change from an undirected to the corresponding directed graph.

Algorithm 4 An algorithm for computing $(1 + \alpha)$ -approximate ϵ -PageRank of a c -balanced graph G .

```

1: function PAGERANKOFBALANCEDGRAPHS( $G, \epsilon, \alpha, \delta$ )
2:   Compute approximate stationary distribution  $\tilde{\pi}_{\epsilon,1}$  using
   Theorem 4.
3:    $l \leftarrow \left\lceil \frac{9 \ln n}{\epsilon} \right\rceil$ 
4:   for  $j \leftarrow 1 \dots \delta^{-1}$  do
5:     Implicitly compute  $G_{\epsilon, \sigma_j}$  using Eq. (8).
6:      $W \leftarrow \text{RANDOMWALKS}(G_{\epsilon, \sigma_j}, l, t)$  where
7:        $t_i(v) = \lceil C \tilde{\pi}_{\epsilon, \sigma_j}(v) n \ln n \cdot k_i \rceil$ 
8:        $k_i$  is defined by Lemma 11
9:        $C$  is defined in Theorem 16
10:     $W_T \leftarrow \emptyset$ 
11:    for all  $w \in W$  in parallel do
12:       $w_T \leftarrow \text{TRANSLATEWALK-}\sigma(G, \epsilon, j, w)$ 
13:      if  $w_T \neq \text{"fail"}$  then Add  $w_T$  to  $W_T$ .
14:     $\tilde{\pi}_{\epsilon, \sigma_{j+1}} = \text{STATIONARYDISTRIBUTION}(W_T, \epsilon)$ 

```

As the main primitive, PAGERANKOFBALANCEDGRAPHS invokes TRANSLATEWALK- σ (Algorithm 5) on Line 12, which takes a PageRank walk in G_{ϵ, σ_j} and either returns a PageRank walk in $G_{\epsilon, \sigma_{j+1}}$ or fails. In the following we say that it *translates* the given walk. Each translation can fail with relatively large probability, but we will take enough walks so that the whole process has a small failure probability.

Within the pseudocode we use $\text{BERNOULLI}(p)$ to denote a random Boolean function, whose each call independently returns **true** with probability p .

In the rest of this section we analyze Algorithm 4 and show that it satisfies the claim given in Theorem 5.

5.1.2 The Success Probability of Algorithm 4. We now prove the correctness of TRANSLATEWALK- σ , which transforms a random walk w in G_{ϵ, σ_j} to a random walk in $G_{\epsilon, \sigma_{j+1}}$, or returns “fail”. We show that the output walk is a random walk in $G_{\epsilon, \sigma_{j+1}}$ and that the probability of “failing” is relatively small.

Lemma 19. *Let w be a random walk of length l in G_{ϵ, σ_j} . Assume that $\delta \leq 1/(2c)$. Then, TRANSLATEWALK- $\sigma(w)$ (Algorithm 5) does not fail and outputs a random walk w_T of length l in $G_{\epsilon, \sigma_{j+1}}$ with probability at least $(1 - 2c\delta)^l$.*

The proof analyzes two cases. In one of them we use the following procedure, which is often called *rejection sampling*

Lemma 20. *Consider two discrete probability spaces P and Q over the same space $\{e_1, \dots, e_n\}$. For $1 \leq i \leq n$, let p_i and q_i give the probabilities of e_i in P and Q respectively. Finally, let $0 \leq r \leq \min_{1 \leq i \leq n, q_i \neq 0} p_i / q_i$.*

Consider an algorithm which given a random sample $X = e_i$ from P returns it with probability $q_i / p_i \cdot r$, and does nothing with the remaining probability. Then, the algorithm returns a result with probability r and each time it does so, it is an element sampled according to Q .

To prove Lemma 19 we consider a procedure that translates the input walk w in l steps, one edge at a time. We show that after i

Algorithm 5 Given a random walk w in G_{ϵ, σ_j} , return a random walk in $G_{\epsilon, \sigma_{j+1}}$ or “fail”. See proof of Lemma 19 for definitions of r , $\rho(v)$ and $\beta(v)$.

```

1: function TRANSLATEWALK- $\sigma(G, \epsilon, j, w)$ 
2:   if  $\sigma_j < 1/2$  then
3:      $d \leftarrow$  number of directed transitions in  $w$ .
4:      $u \leftarrow$  number of undirected transitions in  $w$ .
5:      $p \leftarrow r^l \frac{\sigma_{j+1}^u (1 - \sigma_{j+1})^d}{\sigma_j^u (1 - \sigma_j)^d}$ 
6:     if  $\text{BERNOULLI}(p)$  then return  $w$ .
7:     Otherwise, return “fail”.
8:   else
9:      $w_T \leftarrow w$ . Annotate edges of  $w_T$  as follows:
10:    for each edge  $e = uv$  of  $w_T$  do
11:      if  $e$  is an undirected transition in  $w$  then
12:        if  $\text{BERNOULLI}(1 - \rho(v))$  then
13:           $e$  is an undirected transition in  $w_T$ 
14:        else if  $uv$  is an arc in  $G$  then
15:          Replace  $e$  with a directed transition  $uv$ .
16:        else
17:          return “fail”.
18:      else if  $e$  is a directed transition in  $w$  then
19:        Keep  $e$  in  $w_T$  as a directed transition.
20:      else if  $e$  is a random jump in  $w$  then
21:        if  $\text{BERNOULLI}(\beta(v))$  then return “fail”
22:        else Keep  $e$  in  $w_T$  as a random jump.
23:    return  $w_T$  together with its transition types

```

steps either the procedure has failed or the first i edges of the walk are now sampled according to $G_{\epsilon, \sigma_{j+1}}$.

5.1.3 Proof of Theorem 16. We prove Theorem 16 by showing that Algorithm 4 has the properties given in the theorem statement. Correctness follows from the fact that an iteration of the loop of Algorithm 4 simulates algorithm Algorithm 3 for computing $\tilde{\pi}_{\epsilon, \sigma_{j+1}}$.

We split the rest of the proof into three parts: the space requirements; the round complexity; and, the probability of success. In the proof, we set parameters as $l = \left\lceil \frac{9 \ln n}{\epsilon} \right\rceil$ and $K = \left\lceil \frac{9 \ln n}{\epsilon \cdot \alpha^2} \right\rceil \leq \frac{10 \ln n}{\epsilon \cdot \alpha^2}$. The analysis of success probability fixes the value of C .

Round complexity. Algorithm 4 executes $O(\delta^{-1})$ iterations. Each iteration implicitly constructs a stochastic graph in Line 5, which by Observation 17 can be done in $O(1)$ rounds. Also in each iteration is invoked Algorithm 1, which takes $O(\log l) = O(\log \log n + \log 1/\epsilon)$ rounds. Since we assume that each walk is stored entirely on a machine, TRANSLATEWALK- σ in Line 12 can be implemented without extra communication. To obtain W_T defined by Line 13, each walk that “fails” is marked by flag FAIL, and otherwise marked by flag SUCCEED. Those walks marked by SUCCEED define W_T .

As we prove in the full version, Line 14 requires $O(1)$ rounds. So, Algorithm 4 requires $O(\delta^{-1}(\log \log n + \log 1/\epsilon))$ rounds.

Success probability. By Lemma 15, the probability that any $\tilde{\pi}_{\epsilon, \sigma_j}$ is not $(1 + \alpha)$ -approximation of π_{ϵ, σ_j} is at most $\frac{3}{n^2}$.

The next place where Algorithm 4 can fail is Line 14, i.e., we need to have at least K walks starting in each vertex v . Note that

algorithm generates

$$\begin{aligned} [C\tilde{\pi}_{\epsilon, \sigma_j}(v)n \ln nk_{\lceil \log l \rceil}] &\geq C\tilde{\pi}_{\epsilon, \sigma_j}(v)n \ln n \\ &\geq C\pi_{\epsilon, \sigma_j}(v)(1-\alpha)n \ln n \geq C\epsilon(1-\alpha) \ln n. \end{aligned}$$

walks from each vertex. These walks are subsampled with probability at least $(1-2c\delta)^l$ by Lemma 19. We will aim to have $2K$ walks in expectation, so that by Chernoff bound the probability of this number being smaller than K is

$$\exp(-\delta^2 \mu/3) = \exp(-K/3) \leq \exp(-3 \ln n) = \frac{1}{n^3}.$$

Using union bound over all vertices we will have not enough walks with probability at most $\frac{1}{n^2}$. Hence, we need to set C so that

$$2K \leq (1-2c\delta)^l \cdot C\epsilon(1-\alpha) \ln n.$$

This gives

$$C \geq \frac{20}{\epsilon^2 \alpha^2 (1-\alpha) \cdot (1-2c\delta)^l} \geq \frac{80}{3\epsilon^2 \alpha^2 (1-2c\delta)^l}.$$

This inequality is satisfied for $C = \frac{28}{\alpha^2 \epsilon^2 \cdot (1-2c\delta)^l}$. By Lemma 11 (i) the sampling algorithm fails with probability at most

$$n^{-\frac{C\alpha\epsilon}{3} + 2} e^2 \leq n^{-\frac{28}{3\alpha\epsilon} + 2} e^2 \leq n^{-4+2} e^2.$$

The probability of any of these failures happening in each round is at most $\frac{3}{n^2} + \frac{1}{n^2} + \frac{e^2}{n^2} < \frac{12}{n^2}$. Hence, over all rounds the failure probability is $O(\frac{1}{n^2\delta})$.

Space requirement. By Lemma 11 (iii) and from $1/n \leq \alpha \leq 1/4$, the space required is

$$O\left(m + Cl^{1+2\alpha} n \ln n\right) = O\left(m + \frac{1}{\alpha^2 \epsilon^{3.5}} \cdot n^{18\frac{\epsilon\delta}{\epsilon}} n \ln^{2.5} n\right).$$

Moreover, we need $O\left(\frac{m \log^2 n \log \log n}{\epsilon^2 \alpha^2}\right)$ space in Line 2 of Algorithm 4. This completes the proof of Theorem 16.

5.2 Transformation to a c -balanced Graph

In this section we will describe how to reduce a general graph $G = (V, E)$ without dangling vertices to a 3-balanced multigraph $G_c = (V_c, E_c)$. The ways to handle dangling vertices are discussed in the full version. The idea is to replace each vertex by a path of length $\lambda = \lceil \log n \rceil$. Formally, the graph $G_c = (V_c, E_c)$ is defined as follows

$$\begin{aligned} V_c &= \{v_i : v \in V, i \in [1, \dots, \lambda]\}, \\ E_c &= \{u_\lambda v_1 : uv \in E\} \cup \{(u_i u_{i+1})^j : \\ &\quad i \in [1, \dots, \lambda-1], j \in [0, \dots, \lceil \deg^-(u)/2^i \rceil]\}, \end{aligned}$$

where $(uv)^k$ denotes k directed edges from u to v .

Lemma 21. *If G does not contain dangling vertices then G_c is a 3-balanced graph. Moreover, G_c contains $n \lceil \log n \rceil$ vertices and at most $2m + n \lceil \log n \rceil$ edges.*

We point a reader to the full version for a proof of Lemma 21. For each $uv \in E$ we call the edge $u_\lambda v_1 \in E_c$ *core*. From the construction of G_c we easily get the following.

Observation 22. *Let $W = e_1, e_2, \dots, e_k$ be a walk in G_c . Then, there exists $1 \leq i \leq \lambda$ that has the following property. Let W_R be a subsequence of W consisting of edges $e_{i+j\lambda}$ for $0 \leq j \leq (k-i)/\lambda$. Then, W_R is a walk in G , which contains all core edges of W .*

5.3 Increasing Damping Factor

In Section 5.2 we described how to transform the input graph G to a c -balanced graph G_c . In this process, each edge of G is replaced by a path of length $\lceil \log n \rceil$. That means that a random walk of length l in G corresponds to a random walk of length $l \cdot \lceil \log n \rceil$ in G_c . In order to make a correspondence between PageRank walks in G to those in G_c , we need PageRank walks in G_c to make jump transitions roughly $\log n$ times less frequently than in G . (We make this statement precise in Section 5.4.) In light of this, we design method **PAGERANKINCDAMPINGFACTOR** (Algorithm 7) that given an approximate ϵ -PageRank of G outputs an approximate ϵ' -PageRank of G for $\epsilon' < \epsilon$. Moreover, for a given parameter τ , it does so in $O(\log_{1+\tau} \frac{\epsilon}{\epsilon'})$ iterations each of which is implemented by invoking **RANDOMWALKS** for length $O(\log n / \epsilon')$. The parameter τ also affects space complexity (for details see Theorem 23), and in the final setup, we set $\tau = o(1)$.

PAGERANKINCDAMPINGFACTOR uses **TRANSLATEWALK- ϵ** (Algorithm 6) as a subroutine. Given a random walk w in $G_{\epsilon_j, 0}$ and $\epsilon_{j+1} \leq \epsilon_j$, **TRANSLATEWALK- ϵ** either returns a walk w_T which is a random walk in $G_{\epsilon_{j+1}, 0}$ or “fails”. This method is very similar to **TRANSLATEWALK- σ** for the case $\sigma_j < 1/2$.

Algorithm 6

```

1: function TRANSLATEWALK- $\epsilon(G, \epsilon_j, \epsilon_{j+1}, w)$ 
2:   Let  $g$  be the number of directed transitions in  $w$ .
3:   Let  $t$  be the number of jump transitions in  $w$ .
4:    $p \leftarrow r \frac{\epsilon_{j+1}^{t+1}(1-\epsilon_{j+1})^g}{\epsilon_j^t(1-\epsilon_j)^g}$   $\triangleright r$  is set later to guarantee  $p \leq 1$ .
5:   With probability  $p$  return  $w$ ; otherwise, return “fail”.

```

Algorithm 7 An algorithm that given a $(1+\alpha)$ -approximate ϵ_1 -PageRank $\tilde{\pi}_{\epsilon_1}$ of G for $\epsilon_1 \leq 1/2$, outputs a $(1+\alpha)$ -approximate ϵ' -PageRank $\tilde{\pi}_{\epsilon'}$ of G for $\epsilon' < \epsilon_1$.

```

1: function PAGERANKINCDAMPINGFACTOR( $G, \tilde{\pi}_{\epsilon_1}, \epsilon', \tau$ )
2:   for  $j \leftarrow 1 \dots \lceil \log_{1/(1-\tau)} \epsilon_1 / \epsilon' \rceil$  do
3:      $l \leftarrow \lceil \frac{9 \ln n}{\epsilon_j} \rceil$ 
4:     Implicitly compute  $G_{\epsilon_j, 0}$  using Eq. (8).
5:     Run RANDOMWALKS( $G_{\epsilon_j, 0}, l, t$ ) where  $t_i(v) =$ 
        $[R\tilde{\pi}_{\epsilon_j}(v)n \ln n \cdot k_i]$ ,  $k_i$  is defined by Lemma 11. Let  $W$  be the
       set of resulting walks.  $\triangleright R$  is set in Theorem 23
6:      $W_T \leftarrow \emptyset$ 
7:      $\epsilon_{j+1} \leftarrow \max\{\epsilon', \epsilon_j(1-\tau)\}$ 
8:     for all  $w \in W$  in parallel do
9:       if  $w_T = \text{TRANSLATEWALK-}\epsilon(G, \epsilon_j, \epsilon_{j+1}, w)$  did not
         “fail” then
10:        Add  $w_T$  to  $W_T$ .
11:    $\tilde{\pi}_{\epsilon_{j+1}} = \text{STATIONARYDISTRIBUTION}(W_T, \epsilon_j)$ 

```

THEOREM 23. *Let G be a directed graph. Let $\epsilon_1 > \epsilon' \geq \log n/o(S)$, and let $\tilde{\pi}_{\epsilon_1}$ be a $(1 + \alpha)$ -approximate ϵ_1 -PageRank of G . Given $\tau \in (0, 1/2]$, **Algorithm 7** outputs a $(1 + \alpha)$ -approximate ϵ' -PageRank $\tilde{\pi}_{\epsilon'}$ of G . Moreover, **Algorithm 7** can be implemented in $O(\tau^{-1} \cdot \log 1/\epsilon' \cdot (\log \log n + \log 1/\epsilon'))$ MPC rounds and the total space of $O\left(m + \frac{1}{\epsilon^{3.5}\alpha^2} n^{36\tau} n \ln^{2.5} n\right)$ with strongly sublinear space per machine. This algorithm outputs a correct result with probability at least $1 - O(\frac{1}{\tau n^2} \cdot \log 1/\epsilon')$.*

The proof of this claim is provided in the full version.

5.4 From PageRank in c -balanced to PageRank in General Graphs

So far we developed a way for approximating PageRank of c -balanced graphs. Now we describe how to use that result to approximate the PageRank of a (not necessarily c -balanced) graph G . Let G_c be a c -balanced graph obtained from G by applying the transformation from **Section 5.2**. Recall that to approximate PageRank it suffices to sample random walks up to the point of their *first* random jump (see **Algorithm 4**). For a graph G and damping factor $1 - \epsilon$ such a random walk with high probability has length $O\left(\frac{\log n}{\epsilon}\right)$. Instead of generating PageRank walks in G , we will generate them in G_c , but for a damping factor $1 - \epsilon/\text{poly log } n$ (see **Lemma 24**). This will be done using **Algorithm 7**. Let W_c be an $(\frac{\epsilon}{\text{poly log } n})$ -PageRank walk. Observe that with constant probability the first jump in W_c appears after $O\left(\frac{\log^2 n}{\epsilon}\right)$ steps. If this event occurs for W_c , then W_c gives us a random walk W in G of length $O\left(\frac{\log n}{\epsilon}\right)$ such that no transition of W is a random jump. To obtain ϵ -PageRank we reintroduce random jumps with probability ϵ and truncate walks after first such jump. This is done by sequentially iterating over the edges of W and with probability ϵ truncating W at any given step. This process is given as algorithm **TRANSLATETOPAGERANKWALK** (**Algorithm 8**).

Algorithm 8 Let G be a graph and G_c its c -balanced version. Given a PageRank walk w in G_c that has no random jumps, the algorithm returns an ϵ -PageRank walk in G that has exactly one random jump transition or the algorithm “fails”. This random jump transition is the last one in the walk.

```

1: function TRANSLATETOPAGERANKWALK( $G, \epsilon, w$ )
2:   if  $w$  contains random jump then “fail”.
3:   Let  $w_R$  be the walk in  $G$  consisting of all core edges of  $w$ .
   ▶ See Observation 22
4:   Mark each edge of  $w_R$  independently and with probability
    $\epsilon$ .
5:   if at least one edge of  $w_R$  is marked then
6:     Truncate  $w_R$  before the first marked edge.
7:     Return  $w_R$ .
8:   else
9:     “fail”

```

Now we present the main algorithm of this section.

Algorithm 9 An algorithm for computing a $(1 + \alpha)$ -approximate PageRank $\tilde{\pi}_\epsilon$ of a graph G .

```

1: function PAGERANKOFGENERALGRAPHS( $G, \epsilon, \alpha$ )
2:   Let  $G_c$  be the balanced graph of  $G$  obtained as described in
   Section 5.2.
3:    $\tilde{\pi}_{1/2}^c \leftarrow \text{PAGERANKOFBALANCEDGRAPHS}(G_c, 1/2, \alpha)$ 
4:    $\ell \leftarrow \lceil \log n \rceil \cdot \left\lceil \frac{9 \log n}{\epsilon} \right\rceil$ , and  $\epsilon' \leftarrow 1/(4\ell)$ 
5:    $\tilde{\pi}_{\epsilon'}^c \leftarrow \text{PAGERANKINCDAMPINGFACTOR}(G_c, \tilde{\pi}_{1/2}^c, \epsilon')$ 
6:   Implicitly compute  $(G_c)_{\epsilon', 0}$  using Eq. (8).
7:   Run RANDOMWALKS $((G_c)_{\epsilon', 0}, \ell, t)$  where  $t_i(v) =$ 
    $\lceil L \tilde{\pi}_{\epsilon'}^c(v) n \ln n \cdot k_i \rceil$ ,  $k_i$  is defined by Lemma 11 and  $d'$  is
   a sufficiently large constant. Let  $W$  be the set of resulting
   walks. ▶  $L$  is set in Theorem 5
8:    $W_T \leftarrow \emptyset$ 
9:   for all  $w \in W$  in parallel do
10:    if  $w_T = \text{TRANSLATETOPAGERANKWALK}(G, \epsilon, w)$  did not
    “fail” then
11:      Add  $w_T$  to  $W_T$ .
12:    $\tilde{\pi}_\epsilon = \text{STATIONARYDISTRIBUTION}(W_T, \epsilon)$ 

```

Note that **Algorithm 9** does not directly map PageRank walks from G_c to PageRank walks in G . Instead, it takes advantage of the fact that in order to approximate PageRank one only needs to know the random walks until the first jump transition and proceeds as follows. It first computes PageRank walks in G_c , then discards all walks that have at least one jump transition and finally truncates the resulting walks by simple coin tossing. Note that this truncation step is equivalent to truncating the walks just before the first jump transition.

Now we analyze the correctness of **Algorithm 9**. The following result will be useful in establishing failure probability of **TRANSLATETOPAGERANKWALK**.

Lemma 24. *Let $\ell \geq 1$ be a parameter. Define $\epsilon' = 1/(4\ell)$. An ϵ' -PageRank walk does not make a random jump within the first ℓ steps with probability at least 0.6.*

PROOF. An ϵ' -PageRank walk does not make a random jump within the first ℓ steps with probability

$$(1 - \epsilon')^\ell \geq \exp(-2\epsilon'\ell) = \exp(-1/2) \geq 0.6. \quad \square$$

Lemma 24 essentially states the following. If we are given an ϵ' -PageRank walks in G_c then with probability at $1/2$ it can be turned into a ϵ -PageRank walk in G .

Lemma 25. ***TRANSLATETOPAGERANKWALK** (G, ϵ, w) in **Line 10** of **Algorithm 9** fails with probability at most $1/2$. If the algorithm succeeds, then it returns an ϵ -PageRank walk of G that has exactly one jump transition and that jump transition is the last one in the walk.*

We omit the proof of **Lemma 25** in this version of the paper. Now we are ready to prove the correctness of **Algorithm 9**, which establishes one of the main results of the paper.

THEOREM 5. *Let G be a directed graph on n vertices with m edges. Let $\alpha \in [1/n, 1/4]$ and $\epsilon \in [\log^3 n/o(S), 1]$, where S is the available*

space per machine. There exists an MPC algorithm that, with probability at least $1 - O(\frac{1}{n})$, computes a $(1 + \alpha)$ -approximate ϵ -PageRank vector of G in $\tilde{O}(\log^2 \log n + \log^2 1/\epsilon)$ rounds, using $\tilde{O}(\frac{m}{\alpha^2} + \frac{n^{1+o(1)}}{\epsilon^{3.5}\alpha^2})$ total space and strongly sublinear space per machine.

This claim can be shown by combining [Theorems 16](#) and [23](#) and [Lemma 11](#). We defer details to the full version of this paper. We can now use [Theorem 5](#) to sample directed random walks.

THEOREM 2. *Let G be a directed graph on n vertices with m edges. Let D and l be positive integers such that $l = o(S)/\log^3 n$, where S is the available space per machine. There exists an MPC algorithm that samples D independent random walks of length l starting in v for each v in G . The algorithm runs in $O(\log^2 \log n + \log^2 l)$ rounds and uses $\tilde{O}(m + n^{1+o(1)}l^{3.5} + Dnl^{2+o(1)})$ total space and strongly sublinear space per machine. The algorithm is an imperfect sampler (see [Definition 7](#)) that does not fail with probability $1 - O(n^{-1})$.*

PROOF. Let $\alpha = 1/\log n$ and $\epsilon = 1/(4l)$. Invoke [Theorem 5](#) to obtain a $(1 + \alpha)$ -approximate ϵ -PageRank. This invocation can be implemented in $\tilde{O}(\log^2 \log n + \log^2 l)$ rounds and the total space of $\tilde{O}(m + n^{1+o(1)}l^{3.5})$.

Let t_i be as in [Lemma 11](#). Invoke `RANDOMWALKS`(G, l, t) with $C = 20D/(\alpha\epsilon)$. By [Lemma 11](#), and given that $\tilde{\pi}(v) \geq (1 - \alpha)\pi(v) \geq (1 - \alpha)\epsilon/n$, with probability at least $1 - \Theta(1/n)$ this invocation outputs at least $C\tilde{\pi}(v)n \ln n \geq 20D \ln n$ random walks from each vertex v . Let W be the collection of those walks. Also by [Lemma 11](#), W can be obtained in $O(\log l)$ rounds by using the total space of $\tilde{O}(m + Cl^{1+2\alpha}n) \in \tilde{O}(m + Dnl^{2+o(1)})$.

The walks in W are PageRank walks. Nevertheless, by [Lemma 24](#) and the Chernoff bound, with probability at least $1 - \Theta(1/n)$, for each v , there exist D PageRank walks in W that contain no random jump. Those walks are the walks that satisfy the claim of this theorem. \square

ACKNOWLEDGMENTS

We thank Davin Choo and Julian Portmann for valuable discussions. We thank the anonymous reviewers for useful comments. S. Mitrović was supported by the Swiss NSF grant P2ELP2_181772 and MIT-IBM Watson AI Lab. P. Sankowski was supported by the ERC CoG grant TUGBOAT no 772346.

REFERENCES

- [1] Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. 2006. Local Graph Partitioning using PageRank Vectors. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006)*, 21–24 October 2006, Berkeley, California, USA, *Proceedings*. 475–486.
- [2] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. 2014. Parallel algorithms for geometric graph problems. In *Proceedings of the 46th ACM Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31–June 3, 2014*. 574–583.
- [3] Alexandr Andoni, Zhao Song, Clifford Stein, Zhengyu Wang, and Peilin Zhong. 2018. Parallel graph connectivity in log diameter rounds. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 674–685.
- [4] Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab Mirrokni, and Cliff Stein. 2019. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1616–1635.
- [5] Sepehr Assadi, Yu Chen, and Sanjeev Khanna. 2019. Sublinear algorithms for $(\Delta + 1)$ vertex coloring. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 767–786.
- [6] Sepehr Assadi, Xiaorui Sun, and Omri Weinstein. 2019. Massively Parallel Algorithms for Finding Well-Connected Components in Sparse Graphs. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*. 461–470.
- [7] Konstantin Avrachenkov, Nelly Litvak, Danil Nemirovsky, and Natalia Osipova. 2007. Monte Carlo methods in PageRank computation: When one iteration is sufficient. *SIAM J. Numer. Anal.* 45, 2 (2007), 890–904.
- [8] Bahman Bahmani, Kaushik Chakrabarti, and Dong Xin. 2011. Fast personalized PageRank on MapReduce. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12–16, 2011*. 973–984.
- [9] Paul Beame, Paraschos Koutris, and Dan Suciu. 2013. Communication steps for parallel query processing. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA, June 22–27, 2013*. 273–284.
- [10] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Łącki, and Vahab Mirrokni. 2019. Near-Optimal Massively Parallel Graph Connectivity. *FOCS* (2019).
- [11] Soheil Behnezhad, MohammadTaghi Hajiaghayi, and David G Harris. 2019. Exponentially Faster Massively Parallel Maximal Matching. *FOCS* (2019).
- [12] Pavel Berkhin. 2005. A survey on PageRank computing. *Internet Mathematics* 2, 1 (2005), 73–120.
- [13] Christian Borgs, Michael Brautbar, Jennifer Chayes, and Shang-Hua Teng. 2012. A sublinear time algorithm for PageRank computations. In *International Workshop on Algorithms and Models for the Web-Graph*. Springer, 41–53.
- [14] Sebastian Brandt, Manuela Fischer, and Jara Uitto. 2018. Matching and MIS for Uniformly Sparse Graphs in the Low-Memory MPC Model. *arXiv preprint arXiv:1807.05374* (2018).
- [15] Marco Bressan, Enoch Peserico, and Luca Pretto. 2018. Sublinear Algorithms for Local Graph Centrality Estimation. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7–9, 2018*. 709–718.
- [16] Marco Bressan and Luca Pretto. 2011. Local computation of PageRank: the ranking side. In *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 631–640.
- [17] LA Breyer. 2002. Markovian page ranking distributions: some theory and simulations. (2002).
- [18] Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems* 30, 1–7 (1998), 107–117.
- [19] Keren Censor-Hillel, Eldar Fischer, Gregory Schwartzman, and Yadu Vasudev. 2016. Fast Distributed Algorithms for Testing Graph Properties. In *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27–29, 2016. Proceedings*. 43–56.
- [20] Yen-Yu Chen, Qingqing Gan, and Torsten Suel. 2004. Local methods for estimating PageRank values. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*. ACM, 381–389.
- [21] A. Chiplunkar, M. Kapralov, S. Khanna, A. Mousavifar, and Y. Peres. 2018. Testing Graph Clusterability: Algorithms and Lower Bounds. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*. 497–508.
- [22] Artur Czumaj, Jakub Łącki, Aleksander Mądry, Slobodan Mitrović, Krzysztof Onak, and Piotr Sankowski. 2018. Round compression for parallel matching algorithms. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*. ACM, 471–484.
- [23] Artur Czumaj, Morteza Monemizadeh, Krzysztof Onak, and Christian Sohler. 2019. Planar graphs: Random walks and bipartiteness testing. *Random Structures & Algorithms* (2019).
- [24] Artur Czumaj, Pan Peng, and Christian Sohler. 2015. Testing Cluster Structure of Graphs. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing (Portland, Oregon, USA) (STOC '15)*. ACM, New York, NY, USA, 723–732.
- [25] Artur Czumaj and Christian Sohler. 2010. Testing expansion in bounded-degree graphs. *Combinatorics, Probability and Computing* 19, 5–6 (2010), 693–709.
- [26] Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy. 2011. Estimating PageRank on graph streams. *J. ACM* 58, 3 (2011), 13:1–13:19.
- [27] Atish Das Sarma, Anisur Rahaman Molla, Gopal Pandurangan, and Eli Upfal. 2015. Fast Distributed PageRank Computation. *Theor. Comput. Sci.* 561, PB (Jan. 2015), 113–121.
- [28] Atish Das Sarma, Danupon Nanongkai, Gopal Pandurangan, and Prasad Tetali. 2013. Distributed Random Walks. *J. ACM* 60, 1 (2013), 2:1–2:31.
- [29] Gianna M. Del Corso, Antonio Gulli, and Francesco Romani. 2005. Fast PageRank Computation via a Sparse Linear System. *Internet Math.* 2, 3 (2005), 251–273.
- [30] Neelam Duhan, AK Sharma, and Komal Kumar Bhatia. 2009. Page ranking algorithms: a survey. In *2009 IEEE International Advance Computing Conference*. IEEE, 1530–1537.
- [31] Buddhima Gamalath, Sagar Kale, Slobodan Mitrović, and Ola Svensson. 2018. Weighted Matchings via Unweighted Augmentations. *arXiv preprint arXiv:1811.02760* (2018).
- [32] Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, and Ronitt Rubinfeld. 2018. Improved Massively Parallel Computation Algorithms

- for MIS, Matching, and Vertex Cover. *Proceedings of the 37th ACM Principles of Distributed Computing (PODC 2018)* (2018).
- [33] Mohsen Ghaffari, Fabian Kuhn, and Jara Uitto. 2019. Conditional Hardness Results for Massively Parallel Computation from Distributed Lower Bounds. *FOCS* (2019).
- [34] Mohsen Ghaffari, Silvio Lattanzi, and Slobodan Mitrović. 2019. Improved Parallel Algorithms for Density-Based Network Clustering. In *International Conference on Machine Learning*. 2201–2210.
- [35] Mohsen Ghaffari and Jara Uitto. 2019. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1636–1653.
- [36] Ashish Goel, Michael Kapralov, and Sanjeev Khanna. 2013. Perfect Matchings in $O(n \log n)$ Time in Regular Bipartite Graphs. *SIAM J. Comput.* 42, 3 (2013), 1392–1404.
- [37] Oded Goldreich and Dana Ron. 1999. A Sublinear Bipartiteness Tester for Bounded Degree Graphs. *Combinatorica* 19, 3 (1999), 335–373.
- [38] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. 2011. Sorting, searching, and simulation in the MapReduce framework. In *International Symposium on Algorithms and Computation*. Springer, 374–383.
- [39] Shay Halperin and Uri Zwick. 1996. An Optimal Randomised Logarithmic Time Connectivity Algorithm for the EREW PRAM. *J. Comput. Syst. Sci.* 53, 3 (1996), 395–416.
- [40] Shay Halperin and Uri Zwick. 1996. Optimal Randomized EREW PRAM Algorithms for Finding Spanning Forests and for Other Basic Graph Connectivity Problems. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms* (Atlanta, Georgia, USA) (*SODA '96*). Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 438–447.
- [41] Nicholas JA Harvey, Christopher Liaw, and Paul Liu. 2018. Greedy and Local Ratio Algorithms in the MapReduce Model. In *Proceedings of the 30th Symposium on Parallelism in Algorithms and Architectures*. ACM, 43–52.
- [42] Mark Jerrum and Alistair Sinclair. 1996. The Markov chain Monte Carlo method: an approach to approximate counting and integration. *Approximation algorithms for NP-hard problems* (1996), 482–520.
- [43] Ce Jin. 2019. Simulating Random Walks on Graphs in the Streaming Model. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10–12, 2019, San Diego, California, USA*. 46:1–46:15.
- [44] Satyen Kale and Comandur Seshadhri. 2011. An expansion tester for bounded degree graphs. *SIAM J. Comput.* 40, 3 (2011), 709–720.
- [45] David R. Karger, Noam Nisan, and Michal Parnas. 1999. Fast Connected Components Algorithms for the EREW PRAM. *SIAM J. Comput.* 28, 3 (1999), 1021–1034.
- [46] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. 2010. A Model of Computation for MapReduce. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17–19, 2010*. 938–948.
- [47] Tali Kaufman, Michael Krivelevich, and Dana Ron. 2004. Tight Bounds for Testing Bipartiteness in General Graphs. *SIAM J. Comput.* 33, 6 (2004), 1441–1483.
- [48] Jonathan A. Kelner and Aleksander Mądry. 2009. Faster Generation of Random Spanning Trees. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25–27, 2009, Atlanta, Georgia, USA*. 13–21.
- [49] Amy N Langville and Carl D Meyer. 2004. Deeper inside PageRank. *Internet Mathematics* 1, 3 (2004), 335–380.
- [50] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. 2011. Filtering: a method for solving graph problems in MapReduce. In *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*. ACM, 85–94.
- [51] Siqiang Luo. 2019. Distributed PageRank Computation: An Improved Theoretical Study. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 4496–4503.
- [52] Asaf Nachmias and Asaf Shapira. 2010. Testing the expansion of a graph. *Inf. Comput.* 208, 4 (2010), 309–314.
- [53] Krzysztof Onak. 2018. Round compression for parallel graph algorithms in strongly sublinear space. *arXiv preprint arXiv:1807.08745* (2018).
- [54] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report 1999-66. Stanford InfoLab.
- [55] John H. Reif. 1985. An Optimal Parallel Algorithm for Integer Sorting. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science (SFCS '85)*. IEEE Computer Society, Washington, DC, USA, 496–504.
- [56] Tim Roughgarden, Sergei Vassilvitskii, and Joshua R Wang. 2018. Shuffles and circuits (on lower bounds for modern parallel computation). *Journal of the ACM (JACM)* 65, 6 (2018), 41.