Fast Distributed PageRank Computation

Atish Das Sarma[1], Anisur Rahaman Molla[2],
Gopal Pandurangan[3], and Eli Upfal[4]

[1] eBay Research Labs, eBay Inc., CA, USA
atish.dassarma@gmail.com
[2] Division of Mathematical Sciences, Nanyang Technological University, Singapore 637371
anisurpm@gmail.com
[3] Division of Mathematical Sciences, Nanyang Technological University, Singapore 637371 and Department of Computer Science, Brown University, Providence, RI 02912, USA
gopalpandurangan@gmail.com
[4] Department of Computer Science, Brown University, Providence, RI 02912, USA
eli@cs.brown.edu

Abstract. Over the last decade, PageRank has gained importance in a wide range of applications and domains, ever since it first proved to be effective in determining node importance in large graphs (and was a pioneering idea behind Google's search engine). In distributed computing alone, PageRank vectors, or more generally random walk based quantities have been used for several different applications ranging from determining important nodes, load balancing, search, and identifying connectivity structures. Surprisingly, however, there has been little work towards designing provably efficient fully-distributed algorithms for computing PageRank. The difficulty is that traditional matrix-vector multiplication style iterative methods may not always adapt well to the distributed setting owing to communication bandwidth restrictions and convergence rates.

In this paper, we present fast random walk-based distributed algorithms for computing PageRank in general graphs and prove strong bounds on the round complexity. We first present an algorithm that takes $O(\log n/\epsilon)$ rounds with high probability on any graph (directed or undirected), where n is the network size and ϵ is the reset probability used in the PageRank computation (typically ϵ is a fixed constant). We then present a faster algorithm that takes $O(\sqrt{\log n}/\epsilon)$ rounds in undirected graphs. Both of the above algorithms are scalable, as each node processes and sends only small (polylogarithmic in n, the network size) number of bits per round and hence work in the CONGEST distributed computing model. For directed graphs, we present an algorithm that has a running time of $O(\sqrt{\log n/\epsilon})$, but it requires a polynomial number of bits to processed and sent per node in a round. To the best of our knowledge, these are the first fully distributed algorithms for computing PageRank vectors with provably efficient running time.

Keywords: PageRank, Distributed Algorithm, Random Walk, Monte Carlo Method.

D. Frey et al. (Eds.): ICDCN 2013, LNCS 7730, pp. 11–26, 2013.
© Springer-Verlag Berlin Heidelberg 2013

1 Introduction

In the last decade, PageRank has emerged as a very powerful measure of relative importance of nodes in a network. The term PageRank was first introduced in [14,7] where it was used to rank the importance of webpages on the web. Since then, PageRank has found a wide range of applications in a variety of domains within computer science such as distributed networks, data mining, web algorithms, and distributed computing [8,5,6,12]. Since PageRank is essentially the steady state distribution (or the top eigenvector of the Laplacian) corresponding to a slightly modified random walk process, it is a easily defined quantity. However, the power and applicability of PageRank arises from its basic intuition of being a way to naturally identify "important" nodes, or in certain cases, similarity between nodes.

While there has been recent work on performing random walks efficiently in distributed networks [19,4], surprisingly, little theoretically provable results are known towards efficient distributed computation of PageRank vectors. This is perhaps because the traditional method of computing PageRank vectors is to apply iterative methods (i.e., do matrix-vector multiplications) till (near)-convergence. While such techniques may not adapt well in certain settings, when dealing with a global network with only local views (as is common in distributed networks such as Peer-to-Peer (P2P) networks), and particularly, very large networks, it becomes crucial to design far more efficient techniques. Therefore, PageRank computation using Monte Carlo methods is more appropriate in a distributed model where only limited sized messages are allowed through each edge in each round.

To elaborate, a naive way to compute PageRank of nodes in a distributed network is simply scaling iterative PageRank algorithms to distributed environment. But this is firstly not trivial, and secondly expensive even if doable. As each iteration step needs computation results of previous steps, there needs to be continuous synchronization and several messages may need to be exchanged. Further, the convergence time may also be slow. It is important to design efficient and localized distributed algorithms as communication overhead is more important than CPU and memory usage in distributed page ranking. We take all these concerns into consideration and design highly efficient fully decentralized algorithms for efficiently computing PageRank vectors in distributed networks.

Our Contributions. In this paper, to the best of our knowledge, we present the first provably efficient fully decentralized algorithms for estimating PageRank vectors under a variety of settings. Our algorithms are scalable since, each node processes and sends only polylogarithmic in n (the network size) number of bits per round. Thus our algorithms work in the well-studied CONGEST distributed computing model [16], where there is a restriction on the number of bits (typically, polylogarithmic in n) that can be sent per edge per round. Specifically, our contributions are as follows:

- We present an algorithm, Simple-PageRank-Algorithm (cf. Algorithm 1), that computes the PageRank accurately in $O(\frac{\log n}{\epsilon})$ rounds with high

probability[1], where n is the number of nodes in the network and ϵ is the random reset probability in the PageRank random walk [2,4,19]. Our algorithms work for any arbitrary network (directed as well as undirected).

- We present an improved algorithm, Improved-PageRank-Algorithm (cf. Algorithm 2), that computes the PageRank accurately in *undirected graphs* and terminates with high probability in $O(\frac{\sqrt{\log n}}{\epsilon})$ rounds. We note that though PageRank is usually applied for directed graphs (e.g., for the World Wide Web), however, it is sometimes also applied in connection with undirected graphs as well ([10,1,17,11,21]) and is non-trivial to compute (cf. Section 2.2). In particular, it can be applied for distributed networks when modeled as undirected graphs (as is typically the case, e.g., in P2P network models). We note that the Improved-PageRank-Algorithm as well as the Simple-PageRank-Algorithm require only polylogarithmic in n number of bits to be processed and sent per round and works in the CONGEST model.
- We present an improved algorithm for directed graphs (which is a modified version of the Improved-PageRank-Algorithm) that computes PageRank accurately and terminates with high probability in $O(\sqrt{\frac{\log n}{\epsilon}})$ rounds, but it requires a polynomial number of bits to be processed and sent per node in a round. Assuming ϵ is a constant (which is typically the case), this algorithm as well as the Improved-PageRank-Algorithm yields a sub-logarithmic (in n) running time. Thus, in many networks, this running time can be substantially smaller than even the network diameter (e.g., in constant-degree networks, the diameter is $\Omega(\log n)$).

2 Background and Related Work

2.1 Distributed Computing Model

We model the communication network as an unweighted, connected n-node graph $G = (V, E)$. Each node has limited initial knowledge. Specifically, we assume that each node is associated with a distinct identity number (e.g., its IP address). At the beginning of the computation, each node v accepts as input its own identity number (which is of length $O(\log n)$ bits) and the identity numbers of its neighbors in G. The node may also accept some additional inputs as specified by the problem at hand (e.g., the number of nodes in the network). A node v can communicate with any node u if v knows the id of u.[2] (Initially, each node knows only the ids of its neighbors in G.) We assume that the communication occurs in synchronous *rounds*, i.e., nodes run at the same processing speed and

[1] Throughout, "with high probability (whp)" means with probability at least $1 - 1/n^{\Omega(1)}$, where n is the number of nodes in the networks.

[2] This is a typical assumption in the context of P2P and overlay networks, where a node can establish communication with another node if it knows the other node's IP address. We sometimes call this *direct* communication, especially when the two nodes are not neighbors in G. Note that our algorithm of Section 3 uses no direct communication between non-neighbors in G.

any message that is sent by some node v to its neighbors in some round r will be received by the end of r. To ensure scalability, we restrict the number of bits that are processed and sent per round by each node to be polylogarithmic in n, the network size. In particular, in each round, each node is allowed to send a message of size B bits (where B is polylogarithmic in n) through each communication link. This is a widely used standard model (called the CONGEST(B) model) to study distributed algorithms (e.g., see [16,15]) and captures the bandwidth constraints inherent in real-world computer networks. We assume B to be polylogarithmic in n. We relax this restriction in Section 5, where we allow polynomial (in n) number of bits to be sent across a link per round; thus our algorithm presented in this Section works in the LOCAL model [16], which is another standard model where there is no restriction on the amount of communication per link per round.

There are several measures of efficiency of distributed algorithms; here we will focus on the running time, i.e. the number of rounds of distributed communication. (Note that the computation that is performed by the nodes locally is free, i.e., it does not affect the number of rounds.)

2.2 PageRank

We formally define the PageRank of a graph $G = (V, E)$. Let ϵ be a small constant which is fixed (ϵ is called the *reset* probability, i.e., with probability ϵ, it starts from a node chosen uniformly at random among all nodes in the network). The PageRank of a graph (e.g., see [2,4,19,5]) is the *stationary distribution* vector π of the following special type of random walk: at each step of the walk, with probability ϵ it starts from a randomly chosen node and with remaining probability $1-\epsilon$, it follows a randomly chosen outgoing (neighbor) edge from the current node and moves to that neighbor.[3] Therefore the PageRank transition matrix on the state space (or vertex set) V can be written as

$$P = (\frac{\epsilon}{n})J + (1-\epsilon)Q \tag{1}$$

where J is the matrix with all entries 1 and Q is the transition matrix of a simple random walk on G defined as $Q_{ij} = 1/k$, if j is one of the $k > 0$ outgoing links of i, otherwise 0. Computing PageRank and its variants efficiently in various computation models has been of tremendous research interest in both academia and industry. For a detailed survey of PageRank see e.g., [5,12]. We note that PageRank is well-defined in both directed and undirected graphs. Note that it is difficult to compute analytically (and no such analytical formulas are known for general graphs) the PageRank distribution and hence various computational methods have been used to estimate the PageRank distribution. In fact, this is true for general *undirected* graphs as well [10].

There are mainly two broad approaches to computing PageRank (e.g., see [3]). One is to using linear algebraic techniques, (e.g., the Power Iteration [14]) and the

[3] We sometime use the terminology "PageRank random walk" for this special type of random walk process.

other approach is Monte Carlo [2]. In the Monte Carlo method, the basic idea is to approximate PageRank by directly simulating the corresponding random walk and then estimating the stationary distribution with the performed walk's distribution. In [2] Avrachenkov et al., proposed the following Monte Carlo method for PageRank approximation: Perform K random walks (according to the PageRank transition probability) starting from each node v of the graph G. For each walk, terminate the walk with its first reset instead of moving to a random node. Then, the frequencies of visits of all these random walks to different nodes will approximate the PageRank. Our distributed algorithms are based on the above method.

Monte Carlo methods are efficient, light weight and highly scalable [2]. It has proved to be a useful technique in designing algorithms for PageRank and its variants in important computational models like data streaming [19] and MapReduce [3]. The works in [20,18] study distributed implementation of PageRank in peer-to-peer networks but uses iteration methods.

3 A Distributed Algorithm for PageRank

We present a Monte Carlo based distributed algorithm for computing PageRank distribution of a network [2]. The main idea of our algorithm (formal pseudocode is given in Algorithm 1) is as follows. Perform K (K will be fixed appropriately later) random walks starting from each node of the network in parallel. In each round, each random walk independently goes to a random (outgoing) neighbor with probability $1-\epsilon$ and with the remaining probability (i.e., ϵ) terminates in the current node. (Henceforth, we call this random walk as *'PageRank random walk'*. This random walk can be shown to be equivalent to one based on the PageRank transition matrix P (defined in Section 2.2) [2].) Since, ϵ is the probability of termination of a walk in each round, the expected length of every walk is $1/\epsilon$ and the length will be at most $O(\log n/\epsilon)$ with high probability. Let every node v count the number of visits (say, ζ_v) of all the walks that go through it. Then, after termination of all walks in the network, each node v computes (estimates) its PageRank π_v as $\tilde{\pi}_v = \frac{\zeta_v \epsilon}{nK}$. Notice that $\frac{nK}{\epsilon}$ is the (expected) total number of visits over all nodes of all the nK walks. The above idea of counting the number of visits is a standard technique to approximate PageRank (see e.g., [2,4]).

We show in the next section that the above algorithm approximates PageRank vector π accurately (with high probability) for an appropriate value of K. The main technical challenge in implementing the above method is that performing many walks from each node in parallel can create a lot of congestion. Our algorithm uses a crucial idea to overcome the congestion. We show that (cf. Lemma 1) that there will be no congestion in the network even if we start a polynomial number of random walks from every node in parallel. The main idea is based on the Markovian (memoryless) properties of the random walks and the process that terminates the random walks. To calculate how many walks move from node i to node j, node i only needs to know the number of walks that reached it. It does not need to know the sources of these walks or the transitions that they took before reaching node i. Thus it is enough to send the *count* of

the number of walks that pass through a node. The algorithm runs till all the walks are terminated. It is easy to see that it finishes in $O(\log n/\epsilon)$ rounds with high probability (this is because the maximum length of any walk is $O(\log n/\epsilon)$ whp). Then every node v outputs its PageRank as the ratio between the number of visits (denoted by ζ_v) to it and the total number of visits ($\frac{nK}{\epsilon}$) over all nodes of all the walks. We show that our algorithm computes approximate PageRank accurately in $O(\log n/\epsilon)$ rounds with high probability (cf. Theorem 1).

Algorithm 1. SIMPLE-PAGERANK-ALGORITHM

Input (for every node): Number of walks $K = c \log n$ from each node (where $c = \frac{2}{\delta' \epsilon}$ and δ' is defined in Section 3.2), reset probability ϵ.
Output: PageRank of each node.

[Each node v starts $c \log n$ walks. All walks keep moving in parallel until they terminate. The termination probability of each walk is ϵ, so the expected length of each walk is $1/\epsilon$.]

1: Initially, each node v in G creates $c \log n$ messages (called coupons) $C_1, C_2, \ldots, C_{c \log n}$. Each node also maintains a counter ζ_v (for counting visits of random walks to it).
2: **while** there is at least one (alive) coupon **do**
3: This is i-th round. Each node v holding at least one coupon does the following: Consider each coupon C held by v which is received in the $(i-1)$-th round. Generate a random number $r \in [0, 1]$.
4: **if** $r < \epsilon$ **then**
5: Terminate the coupon C.
6: **else**
7: Select an outgoing neighbor uniformly at random, say u. Add one coupon counter number to T_u^v where the variable T_u^v indicates the number of coupons (or random walks) chosen to move to the neighbor u from v in the i-th round.
8: **end if**
9: Send the coupon's counter number T_u^v to the respective outgoing neighbors u.
10: Every node u adds the total counter number ($\sum_{v \in N(u)} T_u^v$—which is the total number of visits of random walks to u in i-th round) to ζ_u.
11: **end while**
12: Each node outputs its PageRank as $\frac{\zeta_v \epsilon}{cn \log n}$.

3.1 Analysis

Our algorithm computes the PageRank of each node v as $\tilde{\pi}_v = \frac{\zeta_v \epsilon}{nK}$ and we say that $\tilde{\pi}_v$ approximates original PageRank π_v. We first focus on the correctness of our approach and then analyze the running time.

3.2 Correctness of PageRank Approximation

The correctness of the above approximation follows directly from the main result of [2] (see Algorithm 4 and Theorem 1) and also from [4] (Theorem 1). In

particular, it is mentioned in [2,4] that the approximate PageRank value is quite good even for $K = 1$. It is easy to see that the expected value of $\tilde{\pi}_v$ is π_v (e.g., [2]). In [4] (Theorem 1), it shows that $\tilde{\pi}_v$ is sharply concentrated around π using a Chernoff bound technique ([13]). They show,

$$\Pr[\mid \tilde{\pi}_v - \pi_v \mid \leq \delta \pi_v] \leq e^{-nK\pi_v\delta'} \tag{2}$$

where δ' is a constant depending on ϵ and δ. From the above bound (cf. Equation 2), we see that for $K = \frac{2\log n}{\delta' n \pi_{min}}$, we get a sharp approximation of PageRank vector with high probability. Since the PageRank of any node is at least ϵ/n (i.e. the minimum PageRank value, $\pi_{min} \geq \epsilon/n$), so it gives $K = \frac{2\log n}{\delta'\epsilon}$. For simplicity we assume the constant $c = \frac{2}{\delta'\epsilon}$. Therefore, it is enough if we perform $c \log n$ PageRank random walks from each node. Now we focus on the running time of our algorithm.

3.3 Time Complexity

From the above section we see that our algorithm is able to compute the PageRank vector π in $O(\log n/\epsilon)$ rounds with high probability if we perform $c \log n$ walks from each node in parallel without any congestion. The lemma below guarantees that there will be no congestion even if we do a polynomial number of walks in parallel.

Lemma 1. *There is no congestion in the network if every node starts at most a polynomial number of random walks in parallel.*

Proof. It follows from our algorithm that each node only needs to count the number of visits of random walks to itself. Therefore nodes do not require to know from which source node or rather from where it receives the random walk coupons. Hence it is not needed to send the ID of the source node with the coupon. Recall that in our algorithm, in each round, every node currently holding at least one random walk coupon (could be many) does the following. For each coupon, either the walk is terminated with probability ϵ or with remaining probability $1 - \epsilon$, any outgoing edge is chosen uniformly at random to send the coupon. Any particular outgoing edge may be chosen for more than one coupon. Instead of sending each coupon separately through that edge, the algorithm simply sends the count, i.e., number of coupons, to the chosen outgoing neighbor. Since we consider *CONGEST* model, a polynomial in n number of coupon's count (i.e., we can send count of up to a polynomial number) can be sent in one message through each edge without any congestion. □

Theorem 1. *The algorithm* Simple-PageRank-Algorithm *(cf. Algorithm 1) computes PageRank in* $O(\frac{\log n}{\epsilon})$ *rounds with high probability.*

Proof. The algorithm stops when all the walks terminate. Since the termination probability is ϵ, so in expectation after $1/\epsilon$ steps, a walk terminates and with high probability (via the Chernoff bound) the walk terminates in $O(\log n/\epsilon)$ rounds and by union bound [13], all walks (they are only polynomially many)

terminate in $O(\log n/\epsilon)$ rounds whp. Since all the walks are moving in parallel and there is no congestion (cf. Lemma 1), all the walks in the network terminate in $O(\log n/\epsilon)$ rounds whp. Hence the algorithm stops in $O(\log n/\epsilon)$ rounds whp. The correctness of the PageRank approximation follows from [2,4] as discussed earlier in Section 3.2. □

4 A Faster Distributed PageRank Algorithm (for Undirected Graphs)

We present a faster algorithm for PageRank computation. First we present an algorithm for *undirected* graphs and in Section 5 we modify it slightly to work for directed graphs. Our algorithm's time complexity for the undirected graphs holds in the CONGEST model, whereas for directed graphs a slightly better time complexity applies only in the LOCAL model.

We use a similar Monte Carlo method as described in Section 3 to estimate PageRank. This says that the PageRank of a node v is the ratio between the number of visits of PageRank random walks to v itself and the sum of all the visits over all nodes in the network. In the previous section (cf. Section 3) we show that in $O(\log n/\epsilon)$ rounds, one can approximate RageRank accurately by walking in a naive way on general graph. We now outline how to speed up our previous algorithm (cf. Algorithm 1) using an idea similar to the one used in [9]. In [9], it is shown how one can perform *a* standard (simple) random walk in an undirected graph[4] of length L in $\tilde{O}(\sqrt{LD})$ rounds whp (D is the diameter of the network). The high level idea of their algorithm is to perform 'many' short walks in parallel and later 'stitch' them to get the desired longer length walk. To apply this idea in our case, we modify our approach accordingly as speeding up (*many*) PageRank random walks is different from speeding up *one* (standard) random walk. We show that our improved algorithm (cf. Algorithm 2) approximates PageRank in $O(\frac{\sqrt{\log n}}{\epsilon})$ rounds whp.

4.1 Description of Our Algorithm

In Section 3, we showed that by performing $\Theta(\log n)$ walks (in particular we are performing $c \log n$ walks, where $c = \frac{2}{\delta' \epsilon}$, δ' is defined in Section 3.2) of length $\log n/\epsilon$ from each node, one can approximate the PageRank vector π accurately (with high probability). In this section we focus on the problem of how efficiently one can perform $\Theta(n \log n)$ walks ($\Theta(\log n)$ from each node) each of length $\log n/\epsilon$ and count the number of visits of these walks to different nodes. Throughout, by "random walk" we mean the "PageRank random walk" (cf. Section 3).

The main idea of our algorithm is to first perform 'many' short random walks in parallel and then 'stitch' those short walks to get the longer walk of length $\log n/\epsilon$ and subsequently 'count' the number of visits of these random walks to different nodes. In particular, our algorithm runs in three phases. In the first

[4] In each step, an edge is taken from the current node x with probability proportional to $1/d(x)$ where $d(x)$ is the degree of x.

phase, each node v performs $d(v)\eta$ ($d(v)$ is degree of v) independent 'short' random walks of length λ in parallel. (The value of the parameters η and λ will be fixed later in the analysis.) This is done naively by forwarding $d(v)\eta$ 'coupons' having the ID of v from v (for each node v) for λ steps via random walks. The intuition behind performing $d(v)\eta$ short walks is that the PageRank of an undirected graph is proportional to the degree distribution [10]. Therefore we can easily bound the number of visits of random walks to any node v (cf. Lemma 2). At the end of this phase, if node u has k coupons with the ID of a node v, then u is a destination of k walks starting at v. Note that just after this phase, v has no knowledge of the destinations of its own walks, but it can be known by direct communication from the destination nodes. The destination nodes (at most $d(v)\eta$) have the ID of the source node v. So they can contact the source node via *direct* communication. We show that this takes at most constant number of rounds as only polylogarithmic number of bits are sent (since η will be at most $O(\log^3 n/\epsilon)$, shown later). It is shown that the first phase takes $O(\frac{\lambda}{\epsilon})$ rounds with high probability (cf. Lemma 3).

In the second phase, starting at source node s, we 'stitch' some of the λ-length walks prepared in first phase (note that we do this for every node v in parallel as we want to perform $\Theta(\log n)$ walks from each node). The algorithm starts from s and randomly picks one coupon distributed from s in Phase 1. We now discuss how to sample one such coupon randomly and go to the destination vertex of that coupon. One simple way to do this is as follows: In the end of Phase 1, each node v knows the destination node's ID of its $d(v)\eta$ short walks (or coupons). When a coupon needs to be sampled, node s chooses a random coupon number (from the unused set of coupons) and informs the destination node (which will be the next stitching point) holding the coupon C (by direct communication, since s knows the ID of the destination node at the end of the first phase). Let C be the sampled coupon and v be the destination node of C. The source s then sends a 'token' to v and s deletes the coupon C (so that C will not be sampled again next time at s, otherwise, randomness will be destroyed). The process then repeats. That is, the node v currently holding the token samples one of the coupons it distributed in Phase 1 and forwards the token to the destination of the sampled coupon, say v'. Nodes v, v' are called 'connectors' — they are the endpoints of the short walks that are stitched. A crucial observation is that the walk of length λ used to distribute the corresponding coupons from s to v and from v to v' are independent random walks. Therefore, we can stitch them to get a random walk of length 2λ. We therefore can generate a random walk of length $3\lambda, 4\lambda, \ldots$ by repeating this process. We do this until we have completed a length of at least $(\log n/\epsilon - \lambda)$. Then, we complete the rest of the walk by doing the naive random walk algorithm. We show that Phase 2 finishes in $O(\frac{\log n}{\lambda\epsilon})$ rounds with high probability (cf. Lemma 5).

In the third phase we count the number of visits of all the random walks to a node. As we have discussed, we have to create many short walks of length λ from each node. All short walks may not be used to make the long walk of length $\log n/\epsilon$. We show a technique to count all the used short walks' visits to different nodes. Remember that after completion of Phase 2, all the $\Theta(n \log n)$

Algorithm 2. IMPROVED-PAGERANK-ALGORITHM

Input (for every node): Length $\ell = \frac{\log n}{\epsilon}$ of each walk, reset probability ϵ, short walk length $\lambda = \sqrt{\log n}$ and number of walks $K = c \log n$ (where $c = \frac{2}{\delta' \epsilon}$ and δ' is defined in Section 3.2).
Output: PageRank of each node.

Phase 1: (Each node v performs $d(v)\eta = d(v)\log^3 n/\epsilon$ random walks of length $\lambda = \sqrt{\log n}$. At the end of this phase, there are $d(v)\log^3 n/\epsilon$ (not necessarily distinct) nodes holding a 'coupon' containing the ID of v.)

1: **for** each node v **do**
2: Construct $d(v)\eta = d(v)\log^3 n/\epsilon$ messages containing its ID and also the desired walk length of $\lambda = \sqrt{\log n}$. We will refer to these messages created by node v as 'coupons created by v'.
3: **end for**
4: **for** $i = 1$ to λ **do**
5: This is the i-th round. Each node v does the following: Consider each coupon C held by v which is received in the $(i-1)$-th round. If the coupon C's desired walk length is at most i, then v keeps this coupon (v is the desired destination). Else, {v generates a random number $r \in [0, 1]$. If $r < \epsilon$, terminate the coupon C and keep the coupon as then v itself is the destination. Else, pick a neighbor u uniformly at random for the coupon C and forward C to u after incrementing counter}. Note that v does this for every coupon simultaneously in the i-th round.
6: **end for**
7: Each destination node sends its ID to the source node, as it has the source node's ID now.

Phase 2: (Stitch short walks by token forwarding. Stitch $\Theta(\ell/\lambda)$ walks, each of length λ)

1: The source node s creates a message called "token" which contains the ID of s. (Note that for simplicity we are showing the stitching from one source node but this has to be done for each node in the network in parallel.)
2: The algorithm will forward the token around and keep track of a set of connectors, denoted by CON. Initially, $CON = \{s\}$.
3: **while** Length of walk completed is at most $\ell - \lambda$ **do**
4: Let v be the node that is currently holding the token.
5: v samples one of the coupons distributed by v uniformly at random from the unused set of coupons. Let v' be the destination node of the sampled coupon, say C.
6: v sends the token to v' and deletes the coupon C.
7: $CON = CON \cup \{v'\}$
8: **end while**
9: Walk naively until ℓ steps are completed (this is at most another λ steps).
10: A node say w, holding the token having the ID of s is final destination of $\ell = \log n/\epsilon$ length PageRank random walk. $CON = CON \cup \{w\}$

Phase 3: (Counting the number of visits of short walks to a node)

1: Each node v maintains a counter ζ_v to keep track of the number of visits of walks.
2: **for** each walk completed in Phase 2 **do**
3: Start from each connector node in CON except the source node s.
4: Trace the random walk in reverse (in parallel) up to the source node of the corresponding short walk. (Recall that each connector node is the destination of some short walk).
5: Count the number of visits during this reverse tracing and add to ζ_v.
6: **end for**
7: Each node v outputs its PageRank π_v as $\frac{\zeta_v \epsilon}{cn \log n}$.

long walks ($\Theta(\log n)$ from each node) have been stitched. During stitching (i.e., in Phase 2), each connector node (which is also end point of the short walk) should remember the source node of the short walk. Now start from the each connector node and do a walk in reverse direction (i.e., retrace the short walk backwards) to the source node in parallel. During the reverse walk, simply count the visit to nodes. It is easy to see that this will take at most $O(\lambda)$ rounds with high probability (cf. Lemma 6). Now we analyze the running time of our algorithm Improved-PageRank-Algorithm. The compact pseudo code is given in Algorithm 2.

4.2 Analysis

First we are interested in the value of η i.e., how many coupons (short walks) do we need from each node to successfully answer all the stitching requests. Notice that it is possible that $d(v)\eta$ coupons are not enough (if η is not chosen suitably large): We might forward the token to some node v many times in Phase 2 and all coupons distributed by v in the first phase may be deleted. (In other words, v is chosen as a connector node many times, and all its coupons have been exhausted.) If this happens then the stitching process cannot progress. To fix this problem, we use an easy upper bound of the number of visits to any node v of a random walk of length ℓ in an undirected graph: $d(v)\ell$ times. Therefore each node v will be visited as a connector node at most $O(d(v)\ell)$ times with high probability. This implies that each node does not have to prepare too many short walks.

The following lemma bounds the number of visits to every node when we do $\Theta(\log n)$ walks from each node, each of length $\log n/\epsilon$ (note that this is the maximum length of a long walk, whp).

Lemma 2. *If we perform $\Theta(\log n)$ random walks of length $\log n/\epsilon$ from each node, then no node v is visited more than $O(\frac{d(v)\log^3 n}{\epsilon})$ times with high probability.*

Proof. Suppose we perform so many long walks in parallel. In other words, we can say that each node performing one walk of length $\Theta(\log^2 n/\epsilon)$. The bound on the number of visits to each node follows because in each round a node v can get only at most $d(v)$ walks in expectation (since we have an undirected graph) and hence $O(d(v)\log n)$ whp (via Chernoff bound). Since long walk length is $\Theta(\log^2 n/\epsilon)$, so total number of visits is $O(d(v)\log^3 n/\epsilon)$ whp. □

It is now clear from the above lemma (cf. Lemma 2) that $\eta = O(\log^3 n/\epsilon)$ i.e., each node v has to prepare $O(d(v)\log^3 n/\epsilon)$ short walks of length λ in Phase 1. Now we show the running time of algorithm (cf. Algorithm 2) using the following lemmas.

Lemma 3. *Phase 1 finishes in $O(\frac{\lambda}{\epsilon})$ rounds with high probability.*

Proof. It is known from the Lemma 2 that in Phase 1, each node v performs $O(d(v)\log^3 n/\epsilon)$ walks of length λ. Initially each node v starts with

$O(d(v)\log^3 n/\epsilon)$ coupons (or messages) and each coupon takes a random walk according to the PageRank transition probability. Let $\eta = O(\log^3 n/\epsilon)$. We now prove that after any given number of steps j ($j \leq \lambda$), the expected number of coupons at node any v is still $d(v)\eta$. This is because at each step any node v can send (as well as receive) $d(v)$ messages in expectation. The number of messages we started at any node v is proportional to its degree $d(v)$. Therefore, in expectation the number of messages at any node remains same. Thus in expectation the number of messages, say X that want to go through an edge in any round is at most 2η (from both end points). Using Chernoff bound we get, $\Pr[X \geq 4\eta \log n] \leq 2^{-4\log n} = n^{-4}$. It follows that the number of messages that want to go through any edge in any round is at most $4\eta \log n = O(\log^4 n/\epsilon)$ with high probability. Hence there will be at most $O(\log^5 n/\epsilon)$ bits whp at any edge per round (as one message is $\log n$ bits). Since we consider CONGEST(polylog n) model, we can extend all walk's length from i to length $i+1$ in $O(1/\epsilon)$ rounds whp. Therefore, for walks of length λ it takes $O(\lambda/\epsilon)$ rounds whp as claimed. □

Lemma 4. *One time stitching in parallel from each node always finishes within* $O(1)$ *rounds.*

Proof. Each node knows all of its short walk's (or coupon's) destination address. Each time when a (source or connector) node wants to stitch, it randomly chooses one of its unused coupons (created in Phase 1). Then it contacts the destination node (holding the coupon) through *direct* communication and informs it as the next connector node (or stitching point). Since the network allows polylog n congestion, this will finish in constant rounds. □

Lemma 5. *Phase 2 finishes in* $O(\frac{\log n}{\lambda\epsilon})$ *rounds.*

Proof. Phase 2 is for stitching short walks of length λ to get the long walk of length $O(\log n/\epsilon)$. Therefore it needs to stitch approximately $O(\log n/\lambda\epsilon)$ times. Since each time stitching can be done in constant rounds (cf. Lemma 4), Phase 2 finishes in $O(\frac{\log n}{\lambda\epsilon})$ rounds. □

Lemma 6. *Phase 3 finishes in* $O(\lambda)$ *rounds with high probability.*

Proof. Each short walk is of length λ. Phase 3 is simply tracing back the short walks. So it is easy to see we can perform all the reverse walks in parallel in $O(\lambda)$ rounds (same as the time to do all the short walks in parallel in Phase 1). Due to Lemma 3 and the fact that each node can communicate a polylog n number of bits in every round, we can say that Phase 3 finishes in $O(\lambda)$ rounds with high probability. □

Now we are ready to show the main result of this section.

Theorem 2. *The* Improved-PageRank-Algorithm *(cf. Algorithm 2) computes the PageRank accurately and with high probability finishes in* $O(\frac{\sqrt{\log n}}{\epsilon})$ *rounds.*

Proof. The algorithm Improved-PageRank-Algorithm consists of three phases. We have calculated above the running time of each phase separately. Now we want to compute the overall running time of the algorithm by combining these three phases and by putting appropriate value of parameters. By summing up the running time of all three phases, we get from Lemmas 3, 5 and 6 that the total time taken to finish the Improved-PageRank-Algorithm is $O(\frac{\lambda}{\epsilon} + \frac{\log n}{\lambda\epsilon} + \lambda)$ rounds with high probability. Choosing $\lambda = \sqrt{\log n}$, gives the required bound as $O(\frac{\sqrt{\log n}}{\epsilon})$ whp. □

5 A Faster Algorithm for Directed Graphs

We extend the Improved-PageRank-Algorithm of Section 4 to directed graphs. Recall that it follows from Section 3 that it is enough to approximate PageRank vector if each node performs $c \log n$ PageRank random walk of length $\log n/\epsilon$, where $c = 2/\delta'\epsilon$ is a constant. The basic idea of the algorithm is similar as above i.e., create some short walks from each node in parallel and later stitch them to get long walks and then count the number of visits of all the long walks to different nodes. However, the main difficulty in an directed graph is to bound the number of visits of random walks to any node. This is because, in a directed graph we do not have a suitable upper bound on PageRank (unlike the case of an undirected graph). There could be large discrepancy between *indegree* and *outdegree* of a node on a directed graph (in shorthand we use *indeg* and *outdeg* respectively). Therefore, for any node whose indeg and outdeg ratio is large enough, it is very likely that many random walk coupons will pass over those nodes in every round. In the similar way, there can be a large congestion on those nodes if we want to perform a large number of short walks from each node. Hence it is difficult to derive a similar faster algorithm as Algorithm 2 in the CONGEST model for directed graphs. Hence, in this section, we adopt the LOCAL distributed computing model [16] where message size restriction is removed, i.e., nodes can communicate any number of bits in each round. (Our algorithm will need a polynomial number of bits to be processed and sent by a node in each round.) Even in the LOCAL model, it is not obvious how to perform a ℓ length random walk in less than ℓ rounds when $\ell < D$, the diameter of the network. Because in LOCAL model, a trivial solution of any distributed computation problem is to collect all the information of the network to a single node and compute the solution locally. Clearly this will take diameter (D) time. Since we are interested to performing random walks of length $\log n/\epsilon$ which can be much less than the diameter (in general), our algorithm gives a non-trivial result in LOCAL model also. We discuss below our algorithm for directed graphs using the same approach as in Section 4.

5.1 Description of Our Algorithm

It is now clear that only Phase 1 of Algorithm 2 is problematic. We want to modify the Phase 1 of the previous algorithm. First we consider an upper bound

on the number of times any node is visited if we perform $c \log n$ random walks of length $\log n/\epsilon$ from each node. We assume the trivial upper bound that any node v will be visited at most $\frac{cn \log^2 n}{\epsilon}$ times with high probability (since total $cn \log n$ walks of length $\log n/\epsilon$). This bound also trivially holds for number of visits as a connector node. This implies that we have to create $\frac{cn \log^2 n}{\epsilon}$ short walks of length λ from each node in Phase 1. It is easy to see that this can be done in $O(\lambda)$ rounds in the LOCAL model (cf. Lemma 7). The other two phases of the algorithm namely, Phase 2 (stitching short walks) and Phase 3 (counting number of visits) can be done by the same approach as in Section 4. We note that Phase 2 and Phase 3 can be done in almost the same running time without considering *direct* communication in LOCAL model.

5.2 Analysis

Lemma 7. *Phase 1 takes $O(\lambda)$ rounds for performing $\frac{cn \log^2 n}{\epsilon}$ walks of length λ from each node v.*

Proof. We are interested in performing $\frac{cn \log^2 n}{\epsilon}$ random walks of length λ from each node. In the LOCAL model, every node can send or receive any number of messages through an edge in each round. Congestion is not an issue here. Therefore at any round i, each node holding any number of coupons can forward them to randomly chosen outgoing neighbors (in parallel). This will take one round only. Thus by walking in naive way for λ rounds in parallel, all short walks can extend their length to λ, i.e. every coupon will reach to the destination node after λ rounds. So it will finish in $O(\lambda)$ rounds. □

Lemma 8. *Phase 2 finishes in $O(\frac{\log n}{\lambda \epsilon})$.*

Proof. Since Phase 2 is the same as in Algorithm 2, the proof follows from the Lemma 5 above. □

Lemma 9. *Phase 3 finishes in $O(\lambda)$ rounds with high probability.*

Proof. The Phase 3 is also same as in Algorithm 2. The proof follows from the Lemma 6 above. □

Theorem 3. *The algorithm computes the PageRank accurately on directed graph and with high probability finishes in $O(\sqrt{\frac{\log n}{\epsilon}})$ rounds in LOCAL model.*

Proof. The algorithm for computing PageRank on directed graph also comprises of three phases. Combining the running time of these three phases from above we get the total time taken to finish the algorithm: $O(\lambda + \frac{\log n}{\epsilon\lambda} + \lambda)$ rounds with high probability. Choosing $\lambda = \sqrt{\frac{\log n}{\epsilon}}$, gives the required bound as $O(\sqrt{\frac{\log n}{\epsilon}})$. □

6 Conclusion

We presented fast distributed algorithms for computing PageRank, a measure of fundamental interest in networks. Our algorithms are Monte-Carlo and based on the idea of speeding up random walks in a distributed network. Our faster algorithms take time only sub-logarithmic in n which can be useful in large-scale, resource-constrained, distributed networks, where running time is especially crucial. Since they are based on random walks, which are lightweight, robust, and local, they can be amenable to self-organizing and dynamic networks.

References

1. Andersen, R., Chung, F., Lang, K.: Local graph partitioning using pagerank vectors. In: FOCS, pp. 475–486 (2006)
2. Avrachenkov, K., Litvak, N., Nemirovsky, D., Osipova, N.: Monte carlo methods in pagerank computation: When one iteration is sufficient. SIAM J. Number. Anal. 45(2), 890–904 (2007)
3. Bahmani, B., Chakrabarti, K., Xin, D.: Fast personalized pagerank on mapreduce. In: SIGMOD Conference, pp. 973–984 (2011)
4. Bahmani, B., Chowdhury, A., Goel, A.: Fast incremental and personalized pagerank. PVLDB 4, 173–184 (2010)
5. Berkhin, P.: A survey on pagerank computing. Internet Mathematics 2(1), 73–120 (2005)
6. Bianchini, M., Gori, M., Scarselli, F.: Inside pagerank. ACM Trans. Internet Technol. 5(1), 92–128 (2005)
7. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. In: Seventh International World-Wide Web Conference (WWW 1998), pp. 107–117 (1998)
8. Cook, M.: Calculation of pagerank over a peer-to-peer network (2004)
9. Das Sarma, A., Nanongkai, D., Pandurangan, G., Tetali, P.: Efficient distributed random walks with applications. In: PODC, pp. 201–210 (2010)
10. Grolmusz, V.: A note on the pagerank of undirected graphs. CoRR, abs/1205.1960 (2012)
11. Iván, G., Grolmusz, V.: When the web meets the cell: using personalized pagerank for analyzing protein interaction networks. Bioinformatics 27(3), 405–407 (2011)
12. Langville, A.N., Meyer, C.D.: Survey: Deeper inside pagerank. Internet Mathematics 1(3), 335–380 (2003)
13. Mitzenmacher, M., Upfal, E.: Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press, New York (2005)
14. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab (1999)
15. Pandurangan, G., Khan, M.: Theory of communication networks. In: Algorithms and Theory of Computation Handbook, 2nd edn. CRC Press (2009)
16. Peleg, D.: Distributed computing: a locality-sensitive approach. SIAM, Philadelphia (2000)
17. Perra, N., Fortunato, S.: Spectral centrality measures in complex networks. Phys. Rev. E 78, 36107 (2008)

18. Sankaralingam, K., Sethumadhavan, S., Browne, J.C.: Distributed pagerank for p2p systems. In: Proceedings of the 12th International Symposium on High Performance Distributed Computing, pp. 58–68 (June 2003)
19. Sarma, A.D., Gollapudi, S., Panigrahy, R.: Estimating pagerank on graph streams. In: PODS, pp. 69–78. ACM (2008)
20. Shi, S., Yu, J., Yang, G., Wang, D.: Distributed page ranking in structured p2p networks. In: Proceedings of the 2003 International Conference on Parallel Processing (2003)
21. Wang, J., Liu, J., Wang, C.: Keyword Extraction Based on PageRank. In: Zhou, Z.-H., Li, H., Yang, Q. (eds.) PAKDD 2007. LNCS (LNAI), vol. 4426, pp. 857–864. Springer, Heidelberg (2007)