Distributed PageRank Computation: An Improved Theoretical Study

Siqiang Luo
The University of Hong Kong
Pokfulam, Hong Kong
sqluo@cs.hku.hk

Abstract

PageRank is a classic measure that effectively evaluates the node importance in large graphs, and has been applied in numerous applications ranging from data mining, Web algorithms, recommendation systems, load balancing, search, and identifying connectivity structures. Computing PageRank for large graphs is challenging and this has motivated the studies of distributed algorithms to compute PageRank. Previously, little works have been spent on the distributed PageRank algorithms with provably desired complexity and accuracy. Given a graph with n nodes and if we model the distributed computation model as the well-known congested clique model, the state-of-the-art algorithm takes $O(\sqrt{\log n})$ communication rounds to approximate the PageRank value of each node in G, with a probability at least $1 - \frac{1}{n}$. In this paper, we present improved distributed algorithms for computing PageRank. Particularly, our algorithm performs $O(\log \log n)$ rounds (a significant improvement compared with $O(\sqrt{\log n})$ rounds) to approximate the PageRank values with a probability at least $1 - \frac{1}{n}$. Moreover, under a reasonable assumption, our algorithm also reduces the edge bandwidth (i.e., the maximum communication message size that can be exchanged through an edge during a communication round) by a $O(\log n)$ factor compared with the state-of-the-art algorithm. Finally, we show that our algorithm can be adapted to efficiently compute another variant of PageRank, i.e., the batch one-hop Personalized PageRanks, in $O(\log \log n)$ communication rounds.

Introduction

Given a graph G of n nodes, and a probability $\alpha \in (0, 1)$, the PageRank vector of a graph is the stationary distribution π of the following specific type of random walk (Sarma, Gollapudi, and Panigrahy 2011): at each step, with probability α the walk restarts from a node chosen uniformly at random; with the remaining probability $1 - \alpha$, the walker jumps to a randomly chosen neighbor v of the current node. PageRank has emerged as a very powerful measure of relative importance of nodes in a network. It was first proposed as the pioneering idea for Google's search engine for document ranking. Since then, PageRank has been applied in extensive applications, such as data mining, distributed networks, Web algorithms, and recommendation systems (Sarma et al. 2015; Florescu and Caragea 2017; Fujiwara et al. 2013; Ahmadi, Kersting, and Sanner 2011; Neumann, Ahmadi, and Kersting 2011; Ponzetto and Strube 2007).

There have been a number of works on distributed PageRank algorithms (Guo et al. 2017; Sarma et al. 2015; Zhu, Ye, and Li 2005). Among them, the algorithm presented in (Sarma et al. 2015) has the best complexity under a fully-distributed model which is a kind of congested clique model. It models the graph G with n independent nodes such that each node communicates with each other over a complete network (i.e., the clique on the n nodes) in order to compute some function of their inputs. Each pair of nodes can communicate at most b bits (which is also known as the edge bandwidth). The computation is performed in synchronous rounds. The algorithm in (Sarma et al. 2015) finishes computing PageRank of an undirected n-node graph G in $O(\sqrt{\log n})$ rounds with a probability at least $1 - \frac{1}{n}$, whereas the edge bandwidth is $b = O(\log^3 n)$ bits.

The congested clique model (definitions may be slightly different in different papers) has received tremendous interest in recently years, and has been studied extensively on many important problems (e.g., (Jurdziński and Nowicki 2018; Drucker, Kuhn, and Oshman 2014; Hegeman et al. 2015; Ghaffari and Parter 2016; Hegeman, Pemmaraju, and Sardeshmukh 2014; Lenzen 2013)). One attractiveness of the congested clique model is that the model is fundamental to other distributed models. In particular, an efficient algorithm based on the congested clique model can be translated into an efficient algorithm on other distributed models such as the k-machine model (Klauck et al. 2015; Luo et al. 2014) and MapReduce model (Hegeman and Pemmaraju 2015; Dean and Ghemawat 2008; Luo et al. 2012).

Our Contributions. In this paper, we present improved distributed PageRank algorithms based on the congested clique model [1]. Particularly, given an undirected connected graph G of n nodes, we show that with a probability at least $1-\frac{1}{n}$, our algorithm finishes computing approximate PageRank vector in $O(\log \log n)$ rounds and the edge bandwidth is $O(\log^2 n)$ bits (under a reasonable degree assumption). This significantly improves over the algorithm presented

[1]In this paper we consider the model that allows each node v to store $O(d(v) \cdot \mathsf{polylog} n)$ bits, where $d(v)$ is the degree of v.

in (Sarma et al. 2015), which costs $O(\sqrt{\log n})$ rounds, entailing an edge bandwidth of $O(\log^3 n)$ bits.

We show that our algorithm can be adapted to compute another variant of the PageRank, i.e., batch one-hop personalized PageRanks (BPPR) (Luo et al. 2019) (The definition can be referred to Section "Extensions to BPPR"). With at least $1 - \frac{1}{n}$ probability, our algorithm handles BPPR in $O(\log \log n)$ rounds and the edge bandwidth is $O(\log^2 n)$ bits (under a reasonable degree assumption). To the best of our knowledge, this is the first distributed algorithm that handles the BPPR in $o(\log n)$ rounds.

Preliminaries

PageRank based on α-Decay Random Walk. Given a graph $G(V, E)$, a reset probability $\alpha \in (0, 1)$, the PageRank (PR) value of a node v, denoted by $\pi(v)$, is the stationary probability of the following α-decay random walk: a random walk is chosen from a node of G uniformly at random, such that at each step of the traversal, it terminates with probability α and, with the other $1 - \alpha$ probability, moves to a randomly selected out-neighbor of the current node. For any node v, the PageRank of v is denoted by $\pi(v)$. In this paper, we focus on undirected graph G of n nodes.

Congested Clique Model. The congested clique model is one of the most well-known models in distributed graph computation. Given a connected n-node undirected graph $G(V, E)$, a congested clique model models the nodes in G as independent nodes who communicate with each other via message passings. A node u can send a message to a node v if u knows the identity (ID) of v. The computation is performed in synchronous rounds, such that the messages that have been sent in round i would be received at the beginning of the round $i + 1$. It defines a bandwidth b. At each round, each pair of nodes (u, v) can exchange at most messages of b bits. In this paper, we assume initially each node knows the IDs of its neighbors in G.

There are several measures for the efficiency of a distributed algorithm based on the congested clique model. Following existing works (e.g., (Hegeman and Pemmaraju 2015; Sarma et al. 2015)), we focus on two measures of an algorithm in this paper: 1) the round complexity, or the number of rounds (asymptotically) that the algorithm takes to finish; 2) the edge bandwidth b. Based on these measures, Algorithm A is considered to be better than algorithm B, if A has a smaller round complexity as well as an edge bandwidth. Note that, typically the local computation within each node is for free, following the assumptions of the standard congested clique model.

Monte Carlo (MC). The Monte Carlo method generates a number, $\Theta(\log n)$, of random walks from every node, and $\pi(v)$ is estimated by the probability of a walk ends at node v. To implement the idea in the congested clique model, initially, each node generates $\Theta(\log n)$ random walk tokens. A node forwards each token with probability $1 - \alpha$ and terminates the token with probability α. From Lemma 2.2 of (Das Sarma, Nanongkai, and Pandurangan 2009) we know that these steps can be implemented in $O(\log^2 n)$ rounds in the congested clique model with a probability at least $1 - \frac{1}{n}$. It can be also shown that it consumes an edge bandwidth of $O(\log^2 n)$ bits.

Length-L Random Walk. The length-L random walk is a random walk *without decay*, but with a fixed length L.

Related Works

In this section we review the closest existing works on distributed computation under the congested clique model. Distributed PageRank computation is studied in (Sarma et al. 2015) and it is the closest literature to this paper. The algorithm presented in (Sarma et al. 2015) improves the round complexity for PageRank computation in undirected graphs over a naive baseline approach. The main idea is to first generate a number of short walks of length $O(\sqrt{\log n})$ from each node. Then, these generated short walks are stitched together, by at most $O(\sqrt{\log n})$ steps, to compose random walks of length $O(\log n)$. It can be shown that the algorithm takes $O(\sqrt{\log n})$ rounds to finish with high probability [2], and the edge bandwidth is $O(\log^3 n)$ bits. To the best of our knowledge, this is the current best theoretical result for PageRank computation in congested clique model.

There are a number of existing algorithms about round complexity of computing a length-L random walk, and fruitful results have been presented for this problem. In (Das Sarma, Nanongkai, and Pandurangan 2009), the authors give a round complexity of $\tilde{O}(L^{\frac{2}{3}} D^{\frac{1}{3}})$ (where D is the diameter of the network) for computing a single random walk of length L, which is among the first algorithms that improve over the baseline linear round complexity $O(L)$. This round complexity is further improved in (Das Sarma et al. 2010) to achieve $\tilde{O}(\sqrt{LD})$ rounds, In this algorithm, the total number of messages passed is $\Omega(m\sqrt{L})$, where m is the number of edges in the graph. Later, the authors of (Sarma, Molla, and Pandurangan 2012) improve the required number of messages to $O(L)$, with the same round complexity guarantee as $\tilde{O}(\sqrt{LD})$. While the results on the distributed computation of a single random walk is closely related to the distributed PageRank algorithms, it is not clear whether the same round complexity (with reasonable bandwidth) can be achieved, as computing PageRank requires multiple random walks to be conducted simultaneously. There can be pressing challenges in extending the algorithm for computing a single random walk to compute multiple random walks in the congested clique model. The reason is that, when multiple random walks are processed at the same time, the message complexity (or bandwidth) can be significantly enlarged, resulting in an unacceptable bandwidth.

To conclude, little works have been done in distributed PageRank algorithms based on the congested clique model besides the works (Sarma et al. 2015). While there are other relevant literatures that address distributed random walks, they can not be easily adapted to an efficient distributed PageRank algorithm based on the congested clique model.

[2]"With high probability" means with a probability at least $1 - \frac{1}{n^c}$ for $c \geq 1$.

Our Solution

In this section we present our improved algorithm. We first present an algorithm called the Radar-Push (RP) algorithm, that improves MC in terms of the edge bandwidth. We then present a further improved algorithm by recursively using the RP algorithm, called the Multiple-round RP (MRP) algorithm. MRP improves the round complexity significantly. Finally, we show how the MRP algorithm can be employed to compute PageRanks.

Basic Idea of Radar Push

The MC algorithm has an edge bandwidth as high as $O(\log^2 n)$ bits [3]. The high edge bandwidth of MC is due to the randomness of the random walk. Particularly, given that a random walk would extend the walk to a randomly chosen neighboring node of the current node, there is a probability that certain nodes be visited by many walks at the same time. These nodes therefore become the hubs of the walks and the hubs have to communicate with their neighbors frequently, resulting in a high edge bandwidth. To address this issue, we need to avoid such hubs and make the generation of the random walks more controllable. Based on this idea, we propose the Radar-Push (RP in short) algorithm such that, instead of randomly extending a walk to a neighboring node, it conducts a deterministic push to all the neighboring nodes. For ease of presentation, let us consider a unit task as follows:

Unit Task: *Generate for each node $v \in V$ a number $d(v)$ of length-L random walks [4], where $d(v)$ is the degree of node v in G.*

Note that, the purpose of introducing the unit task is twofold: 1) It is much easier to present our idea of RP if to handle the unit task. 2) Computing the unit task can be closely related to computing the PageRank (as will be introduced in the following sections). We will explain the Radar-Push algorithm based on this simple task, and also introduce how the computation of PageRank reduces to this simple task.

Let (s, t, i) be a *walk label* to describe the computation state of a length-i walk that starts from node s and currently resides at node t. The main idea of the RP algorithm is that, when a certain random walk (residing at node v) is extended one step further, we do not select a neighbor node of v uniformly at random to extend the walk. Instead, we collect all the random walks which currently reside at v, and *manually* assign their next nodes as evenly as possible. Interestingly, this will not hurt the ending node distribution of a random walk (i.e., the probability of a node becoming the ending node of a walk), and hence the correctness of computing PageRank is guaranteed. The detailed procedure can be referred to the following algorithm.

Algorithm 1.

- **Round 0.** For any node v and each neighbor u of v, v initializes a random walk with the label $(v, u, 1)$, and node v sends the label as a message to node u.
- **Round i ($1 \leq i \leq L-1$).** At the beginning of Round i each node v receives $k = d(v)$ walk labels (will explain shortly why such number of walk labels received) which are $\{(s_1, v, i), (s_2, v, i), \ldots, (s_k, v, i)\}$. We shuffle the labels and increase their lengths by 1, forming a new label list represented as $\{(s_1^*, v, i+1), (s_2^*, v, i+1), \ldots, (s_k^*, v, i+1)\}$. Note that $\{s_1^*, \ldots, s_k^*\}$ is a permutation of $\{s_1, \ldots, s_k\}$. We change the second element in each of the labels to one of $v's$ neighbors, such that each neighbor appears exactly once (this can be done because we have exactly $d(v)$ labels). For example, if $v's$ neighbors are $\{u_1, u_2, \ldots, u_{d(v)}\}$. Then the labels become $\{(s_1^*, u_1, i+1), (s_2^*, u_2, i+1), \ldots, (s_k^*, u_k, i+1)\}$. Node v sends the labels to the nodes indicated by their corresponding second elements in the labels, respectively.
- **Round L.** Each label (s, v, L) that is received corresponds to a generation of a length-L random walk from s to v.
- **Backtrack (optional).** In some cases, we require node s to collect the random walks that are sourced at node s. This can be simply done by sending the walk label back to the source node of the walk.

The above algorithm can be understood as a propagate algorithm that for each node v, there are $d(v)$ walks start at v, and walks are propagated to immediate neighboring nodes, as evenly as possible. There are several interesting designs of the algorithm.

First, we show that the estimator based on RP is not looser than directly using Monte-Carlo (MC), which samples $d(v)$ random walks from node v. There is a crucial difference between RP and MC: in MC, suppose we need to extend p walks from node v to $v's$ neighbors, we select p times for p neighbors uniformly at random and extend the walks to the neighbors respectively. In RP, the walks are extended to $v's$ neighbor evenly. This results in the fact that each node v will receive exactly $d(v)$ walks (under the condition that G is undirected). To explain, at each round there is exactly one message sent through an edge and therefore exactly one message received through the edge. The difference made here is crucial to an improved bandwidth. It means that, at each round each edge conveys at most $O(1)$ labels (and therefore $O(\log n)$ bits as an integer in $[1, n]$ costs $O(\log n)$ bits). Whereas in MC, the message received by each node can be much worse. Consider that in the worst case, all the neighbors of v may send all their messages to v, resulting in a message bottleneck. This results in an unsatisfactory edge bandwidth in MC.

Second, the random walks in RP become dependent with each other, making the correctness analysis much more difficult than MC. To illustrate, consider that node s has two neighbors u_1 and u_2 and we need to extend two walks from s to its neighbors (see Figure 1). In MC, the two walks are independent of each other, i.e., which neighbor the first walk extends to does not affect the other walk. As a result, there are 4 cases shown in Figure 1, each would happen with probability $1/4$. However, in RP, if the first walk extends to u_1, then the second walk must extend to u_2, making them dependent of each other. Then there are only 2 cases (Figure 1 bot-

[3] It can be simply derived based on Lemma 4.1 of (Sarma et al. 2015)

[4] Precisely, we are only interested in the ending node of a walk. We call a walk is generated if we know its ending node.

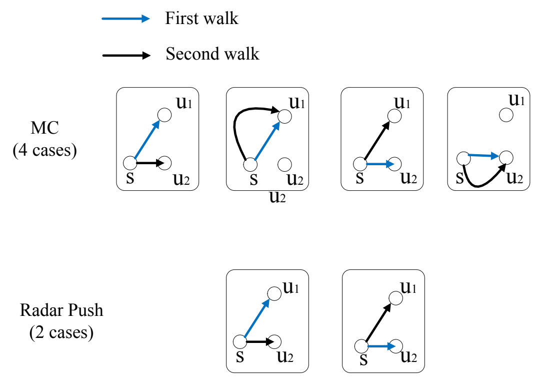

Figure 1: Differences between MC and Radar Push (RP) when extending a random walk.

tom) each with a probability $1/2$. The side-effect of the walk dependency is that, it invalidates the analysis that is used in MC. Therefore we need new analysis to show that the RP gives an unbiased estimator and the estimate does not become looser. To give a formal analysis, we define $\tilde{\Pr}(s,v,L)$ as the RP's estimator of $\Pr(s,v,L)$, the probability that a walk which starts at node s will reach node v at the L-th step, and $\tilde{\Pr}_{mc}(s,v,L)$ is that given by MC. We give the following lemmas to complete the correctness proofs of PR. Particularly, Lemma 1 tells that using RP will not affect the ending node distribution of a random walk, whereas Lemma 2 shows that using RP can even gain a smaller variance for its estimator, compared with that of the MC method.

Lemma 1. *The* $\tilde{\Pr}(s,v,L)$ *is an unbiased estimator of* $\Pr(s,v,L)$.

Proof. We prove by mathematical induction on walk length L. When $L = 1$. Based on Radar-Push, for each neighbor v of s, we have $\mathbb{E}[\tilde{\Pr}(s,v,1)] = \frac{1}{d(s)} = \Pr(s,v,1)$. Lemma holds. Assume that the lemma holds when $L = i$, we consider the case when $L = i+1$. Consider any fixed node v, given the randomness of the shuffle, for any neighboring node u there is $\frac{1}{d(u)}$ probability that a walk is extended to v. Therefore, for any s and v, $\mathbb{E}[\tilde{\Pr}(s,v,i+1)] = \mathbb{E}[\sum_{u\in N(v)} \tilde{\Pr}(s,u,i)\cdot\frac{1}{d(u)}] = \sum_{u\in N(v)} \Pr(s,u,i)\cdot\frac{1}{d(u)} = \Pr(s,v,i{+}1)$. The lemma also holds when $L = i{+}1$. By induction, the lemma holds for all $L \geq 1$. □

Lemma 2. $\mathrm{Var}[\tilde{Pr}(s,v,L)] \leq \mathrm{Var}[\tilde{Pr}_{mc}(s,v,L)]$.

Proof. Let $x(s,v,i)$ be the number of random walks from s that ends at node v at step i for the RP algorithm, whereas $x_{mc}(s,v,i)$ be that for the MC algorithm. Our goal is to show that

$$\mathrm{Var}[x(s,v,L)] \leq \mathrm{Var}[x_{mc}(s,v,L)]$$

Let the neighbors of v be $\{u_1,\ldots,u_{d(v)}\}$. Conditioned on $x(s,u_i,L-1)$ for all i, the probability that v receives a random walk from u_i is $x(s,u_i,L-1)/d(u_i)$. Thus

$$x(s,v,L) = \sum_{1\leq i\leq d(v)} y_i$$

where $y_i = 1$ with probability $x(s,u_i,L-1)/d(u_i)$, and $y_i = 0$ otherwise.

It is easy to show that the conditional variance

$$\begin{aligned}
&\mathrm{Var}[x(s,v,L)|x(s,u_1,L-1),\ldots,x(s,u_{d(v)},L-1)]\\
=&\sum_{1\leq i\leq d(v)} \mathrm{Var}[y_i]\\
=&\sum_{1\leq i\leq d(v)} \mathbb{E}[y_i^2]-(\mathbb{E}[y_i])^2\\
=&\sum_{1\leq i\leq d(v)} \frac{x(s,u_i,L-1)}{d(u_i)}-\frac{x(s,u_i,L-1)^2}{(d(u_i))^2} \qquad (1)
\end{aligned}$$

Taking over the expectation of the variables $x(s,u_i,L-1)$ for all i, we have

$$\begin{aligned}
&\mathrm{Var}[x(s,v,L)]\\
=&\sum_{1\leq i\leq d(v)} \frac{\mathbb{E}[x(s,u_i,L-1)]}{d(u_i)}-\frac{\mathbb{E}[x(s,u_i,L-1)^2]}{(d(u_i))^2} \qquad (2)
\end{aligned}$$

On the other hand, conditioned on $x_{mc}(s,u_i,L-1)$, for all $i = 1,2,\ldots,d(v)$, we can express

$$x_{mc}(s,v,L) = \sum_{1\leq i\leq d(v)} \sum_{1\leq j\leq x_{mc}(s,u_i,L-1)} z_{ij}$$

where $z_{ij} = 1$ with probability $1/d(u_i)$ and $z_{ij} = 0$ otherwise. Given that $\mathrm{Var}[z_{ij}] = 1/d(u_i) - 1/(d(u_i))^2$, and that the variables z_{ij} are independent, we have the conditional variance

$$\begin{aligned}
&\mathrm{Var}[x_{mc}(s,v,L)|\{x_{mc}(s,u_i,L-1)\}_{i\leq d(v)}]\\
=&\sum_{1\leq i\leq d(v)} \frac{x_{mc}(s,u_i,L-1)}{d(u_i)}-\frac{x_{mc}(s,u_i,L-1)}{(d(u_i))^2} \qquad (3)
\end{aligned}$$

Taking over expectation of $x_{mc}(s,u_i,L-1)$, for all i, we have

$$\begin{aligned}
&\mathrm{Var}[x_{mc}(s,v,L)]\\
=&\sum_{1\leq i\leq d(v)} \frac{\mathbb{E}[x_{mc}(s,u_i,L-1)]}{d(u_i)}-\frac{\mathbb{E}[x_{mc}(s,u_i,L-1)]}{(d(u_i))^2}\\
=&\sum_{1\leq i\leq d(v)} \frac{\mathbb{E}[x(s,u_i,L-1)]}{d(u_i)}-\frac{\mathbb{E}[x(s,u_i,L-1)]}{(d(u_i))^2}\\
\geq&\sum_{1\leq i\leq d(v)} \frac{\mathbb{E}[x(s,u_i,L-1)]}{d(u_i)}-\frac{\mathbb{E}[x(s,u_i,L-1)^2]}{(d(u_i))^2}\\
=&\mathrm{Var}[x(s,v,L)] \qquad (4)
\end{aligned}$$

where the inequality holds since $x(s,u_i,L-1)$ are integers. □

To conclude, the RP algorithm handles the unit task by $O(L)$ rounds, entailing an edge bandwidth of $O(\log n)$ bits per round (note that in each round each edge only conveys $O(1)$ integers as messages, and therefore $O(\log n)$ bits edge bandwidth.). This improves over the MC algorithm, which has the same round complexity but with $O(\log^2 n)$ bits edge bandwidth.

Multi-Round Radar Push (MRP)

When $L = 2^r$ (for integer $r \geq 1$), we show that recursively using the RP algorithm effectively reduces the round complexity to $O(\log L)$. The idea is to first compute enough walks of length 2, and then stitch two such walks to generate a longer walk of length $4 = 2 \cdot 2$. Recursively, finally the walks of length 2^r can be stitched by two walks of length 2^{r-1}. To implement this procedure, we consider the following r batches of computation in the congested cliques. In Batch 1 we first generate $2^{r-1} \cdot d(v)$ length-2 walks from node v, for $v \in V$, by using the RP algorithm 2^{r-1} times. The i-th invocation correspondingly generates the length-2 sub-walks, which can be regarded as the sub-walks (of the final length-L walks) containing the i-th jump and $i+1$-th jump for $i = 1, 3, \ldots, 2^r - 1$. We use *step-*$\{a,b\}$ *walk* to refer to the sub-walks from the a-th jump to the b-th jump. In Batch 2 (if $L \geq 4$) we stitch the short walks generated in Batch 1, such that the step-$\{i, i+1\}$ walks and step-$\{i+2, i+3\}$ walks are stitched, for $i = 1, 5, \ldots, 2^r - 3$. This generates length-4 walks, which are denoted by *step-*$\{i, i+3\}$ *walks*. Note that, such walk concatenations are possible because the RP algorithm done by the previous batch guarantees that each node v has $d(v)$ step-$\{i, i+1\}$ walks that end at v, and $d(v)$ step-$\{i+2, i+3\}$ walks that start at v (and this property similarly applies to later batches). The stitches of the walks at each node are similar in spirit to the Radar Push in that the incoming walks of node v are stitched to a random permutation of the outgoing walks. Recursively, in Batch i ($1 \leq i \leq r$) we generate length-2^i walks, and using $r (= O(\log L))$ batches we generate all the length-L walks.

Algorithm 2.

- **Batch** 1**.** Generate $2^{r-1} \cdot d(v)$ length-2 walks from node v, for $v \in V$, by using the RP algorithm for 2^{r-1} times. The i-th invocation correspondingly generates the i-th and $i+1$-th steps (i.e., the i-th and $i+1$-th jumps) in each of the random walks, for $1 \leq i \leq 2^{r-1}$. We call them step-$\{i, i+1\}$ walks, for $i = 1, 3, \ldots, 2^r - 1$.
- **Batch** i **(**$2 \leq i \leq r$**).** In the previous batch we have generated length-2^{i-1} walks, which are the step-$\{1, 2^{i-1}\}$ walks, step-$\{2^{i-1}+1, 2^i\}$ walks, …, step-$\{2^r - 2^{i-1} + 1, 2^r\}$ walks. We stitch two length-2^{i-1} walks together to generate length-2^i short walks, which are the step-$\{1, 2^i\}$ walks, step-$\{2^i+1, 2^{i+1}\}$ walks, …, step-$\{2^r - 2^i + 1, 2^r\}$ walks.

Remarks. The above MRP algorithm can be understood as recursively connecting two shorter walks to longer walks. Consider that a walk W_2 is connected to a walk W_1, we call the ending node of the W_1 (which is also the starting node of W_2) the *connector*.

Complexity. Each batch costs $O(1)$ rounds. In total there are $O(\log L)$ batches and therefore the algorithm takes $O(\log L)$ rounds. As for bandwidth, we use a lemma in (Lenzen 2013) for the analysis.

Lemma 3. *(Lenzen 2013) Assume that each node of the congested clique is given a set of $O(n)$ messages (each of size $O(\log n)$ bits) with fixed destination nodes. Moreover, each node is the destination of $O(n)$ messages from other nodes. Then, it is possible to deliver all messages in $O(1)$ rounds of the congested clique with edge bandwidth of $O(\log n)$ bits.*

Lemma 3 tells that, Batch 1 in Algorithm 2 can be performed in $O(1)$ rounds with an edge bandwidth of $O(L \cdot \log n)$ bits, as each node v receives or sends at most $O(L \cdot d(v)) = O(L \cdot n)$ messages in each round. Note that later batches do not have a per-edge message communication cost higher than Batch 1. Therefore, Algorithm 2 runs in $O(\log L)$ rounds with edge bandwidth of $O(L \cdot \log n)$ bits.

MRP for 2^r-Bounded Unit Task

In PageRank computation it requires that every node generates a random walk, with its length independently drawn from a geometric distribution Geom(α). Therefore, we need to further consider the following task for the preparation of computing PageRank.

2^r**-Bounded Unit Task:** Given $L = 2^r$ for an integer $r > 0$, each node v generates $d(v)$ random walks of lengths $L^v_1, \ldots L^v_{d(v)}$, such that $L^v_i \leq L$ for $1 \leq i \leq d(v)$.

Note that, the aforementioned Algorithm 2 is efficient in generating fixed length random walks from each node. Further extending this algorithm to handle the 2^r-Bounded Unit Task would give us a challenge: how to modify the MRP algorithm so that we can extract the ending node of a length-L^* walk ($L^* \leq L$) in $O(\log L)$ rounds?

To address the challenge, our main idea is to conduct a bisection search to locate the ending node of a length-L^* walk. To see this, observe that each walk generated by MRP can be organized by the hierarchy illustrated by Figure 2. From Figure 2 we see that the walk is recursively divided by the connector and two walks. Note that it is simple to include the connector into the walk label, such that we have four elements in a walk label, respectively indicating the starting node, ending node, the connector as well as the walk length. For a walk label W, we respectively use $W.start$, $W.end$, $W.connect$ and $W.len$ to refer to its four elements. With such an expanded walk label, we can conduct a binary search on the hierarchy from tree-note 0 to locate the tree node that contains the sub-walk containing the desired ending node. For example, suppose we would like to extract the sub-walk of length 5 for the walk in tree-node 0 in Figure 2. Based on the binary search, the search first goes from tree-node 0 to tree-node 2, as 5 is larger than half length of the current random walk. When the search goes to tree-node 2, the current walk length is updated to $8/2 = 4$ and the current searching length is updated to $5 - 8/2 = 1$, meaning we would like to search a sub-walk of length 1 from the current random walk. Similarly, the search further goes to tree-node 5 as the current searching length (which is 1) is not larger than half length of the current walk. And then we update

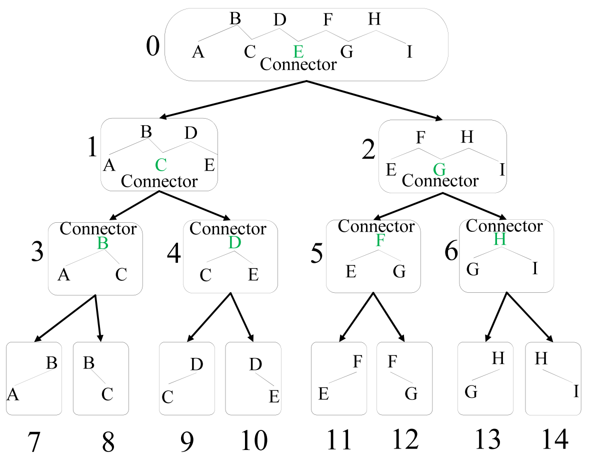

Figure 2: The hierarchical structure of a walk in MRP (connector in green).

the current random walk length to be $4/2 = 2$ and current searching length remains to be 1. The search goes on and finally it goes to tree-node 11, which correctly identifies node F as the ending node of the length-5 sub-walk. Each drilling down of the hierarchy would cost one round in the congested clique model, and therefore it requires only $O(\log L)$ rounds (as the height of the hierarchy is $O(\log L)$) in total to retrieve a length L^*-walk where $L^* \leq L$. The algorithm is described as follows.

Algorithm 3.

- Use Algorithm 2 to generate $d(v)$ length-L random walks that are started at $v, \forall v \in V$. We denote the walks sourced at v by $W_1, \ldots, W_{d(v)}$.
- We subtract the walk W_i $(1 \leq i \leq d(v))$ a sub-walk of length L_i^v (with the same source node) by the following method:

  • **Round** 0**.** Consider each walk $W_i = (v, u, c, L)$, which is a length-L walk from node v to node u with connector c. The current walk length $cur\_walk\_len = L$, and the current searching length $cur\_search\_len = L_i^v$. Node u, the holder of W_i, would either send messages to the connector c or itself or node v, depending on the following three conditions (in a way similar to the binary search in the hierarchy of the current random walk.)

  [Case 1.] If $cur\_search\_len < cur\_walk\_len/2$, then node u sends a *sub-walk message*, which is a message that facilitates the sub-walk search, to the connector c, such that the sub-walk message is represented by $(v, u, v, c, cur\_walk\_len/2, L_i^v)$. Essentially, this sub-walk message tells node c that we need to get a sub-walk of length L_i^v (the last element in the message), and the current walk length should be updated to $cur\_walk\_len/2$ (the fifth element in the message). The other elements in the sub-walk message have the following meaning: the first two elements record the start/end node of the original walk of length L; the next two elements record the start/end node of the current sub-walk.

  [Case 2.] If $cur\_search\_len > cur\_walk\_len/2$, then node u sends a sub-walk message to node u itself, with the sub-walk message represented as $(v, u, c, u, cur\_walk\_len/2, L_i^v - cur\_walk\_len/2)$.

  [Case 3.] If $cur\_search\_len = cur\_walk\_len/2$, send a message to node v saying that a walk ending at node c is generated.

  • **Round** j $(0 < j \leq r)$**.** For each received sub-walk message $(s_1, t_1, s_2, t_2, cur\_walk\_len, cur\_search\_len)$ from round $j-1$, we do the following two steps.

  [Step 1 of Round j.] In this step we aim to identify the current sub-walk from s_2 to t_2. This can be done by searching the walk labels and the connection information [5] holding at node s_2. The identified sub-walk W must satisfy: 1) $W.start = s_2$; 2) $W.end = t_2$; 3) $W.len = cur\_walk\_len$.

  [Step 2 of Round j.] In this step we conduct the same binary search as that in Round 0 based on the identified walk W.

Edge Bandwidth. Note that the additional sub-walk extraction steps included in Algorithm 3 does not affect the edge bandwidth. To see this, observe that during each round the connector would not receive more sub-walk messages than the walks (of the same lengths) it has stitched. That means, during the sub-walk extraction, each connector will send/receive at most the number of messages that it has sent/received in Algorithm 2. Therefore, the edge bandwidth remains the same as Algorithm 2, which is $O(L \log n)$ bits for the unit task.

Computing PageRank using MRP

In this section we show that the MRP algorithm (Algorithm 3) can be used to efficiently compute PageRank. We first give the number of random walks that need to be sampled using MC (see Lemma 5).

Lemma 4 (Chernoff Bound)**.** *Let $X = \sum_{i=1}^m X_i$ be the sum of m independent random variables, where $X_i = 1$ with probability p_i and $X_i = 0$ with probability $1 - p_i$. Let $\mu = \mathbb{E}[X]$. Then for all $1 > \delta > 0$,*

$$\Pr[|X - \mu| \geq \delta\mu] \leq 2e^{-\frac{\delta^2 \mu}{3}}$$

Let $\pi(u)$ be the PageRank of node u. Fix an error bound $\epsilon \in (0, 1)$, and a failure probability p_f, we show that with probability at least $1 - p_f$, we can estimate the PageRank of each node with a multiplicative error ϵ.

Suppose we initiate $k \cdot d(s)$ random walks from each node s with RP, where k is an integer, and estimate $\pi(u)$ by $\hat{\pi}(u) = \frac{1}{n} \sum_{s \in V} \frac{c(s)}{k \cdot d(s)}$, where $c(s)$ is the number of random walks that start at s and end at u. By Lemma 1, $\hat{\pi}(u)$ is an unbiased estimator of $\pi(u)$, i.e., $\mathbb{E}[\hat{\pi}(u)] = \pi(u)$.

[5] We assume that each node stores the walk labels of incoming walks and outgoing walks, as well as how they are connected to generate longer walks. Since each node conducts $O(L)$ walk connections and the information for each connection costs a storage of $O(d(v) \log n)$ bits for a node v, it requires each node v to store $O(d(v) L \log n)$ bits.

Lemma 5. *For $k = \frac{3\log(2/p_f)}{\epsilon^2\alpha}$, we have*

$$\Pr[|\hat{\pi}(u) - \pi(u)| > \epsilon \cdot \pi(u)] \leq p_f.$$

Proof. Observe that the $k \cdot d(s)$ random walks initiated from node s can be grouped into k sets, each of size $d(s)$. We only require that the random walks within the same group are sent to different neighbors of s. Hence initiating $k \cdot d(s)$ random walks from node s is equivalent to repeating k independent MRP, in which each node s initiates $d(s)$ random walks.

Let n_i be the number of random walks that end at node v, in the i-th run of MRP. In Lemma 1, we have shown that $\frac{n_i}{n}$ is an unbiased estimator of $\pi(v)$; in Lemma 2, we have shown that the variable has a better concentration than the estimator given by the MC model. Let $\tilde{n}_i$ be the estimator given under the MC model, which is the summation of n random variables x_j, each with $\mathbb{E}[x_j] = \pi(v)$. Then we have

$$\begin{aligned}
&\Pr\left[\left|\tfrac{1}{k}\textstyle\sum_{i=1}^{k}\tfrac{n_i}{n} - \pi(v)\right| > \epsilon\pi(v)\right]\\
\leq &\Pr\left[\left|\tfrac{1}{k}\textstyle\sum_{i=1}^{k}\tfrac{\tilde{n}_i}{n} - \pi(v)\right| > \epsilon\pi(v)\right]\\
= &\Pr\left[\left|\textstyle\sum_{i=1}^{kn} x_i - kn\cdot\pi(v)\right| > \epsilon kn\cdot\pi(v)\right]\\
\leq &2\cdot\exp(\tfrac{\epsilon^2}{3}\cdot kn\cdot\pi(v))\\
\leq &2\cdot\exp(\frac{\epsilon^2\alpha}{3}\cdot k) = p_f. \qquad (\text{due to } \pi(v) \geq \frac{\alpha}{n})
\end{aligned}$$

□

In Lemma 5, let $p_f = \frac{1}{n}$, then $k = \frac{3\log(2n)}{\epsilon^2\alpha}$. Therefore, one can sample $O(\frac{\log n}{\epsilon^2\alpha}\cdot d(u))$ walks from each node u to achieve the approximation guarantee given by Equation 5.

As pointed out in (Sarma et al. 2015), sampling α-decay random walks is equivalent to generating a random walk of length L where L follows a geometric distribution Geom(α). That means, to compute the PageRank, we can perform $O(d(v)\cdot\frac{\log n}{\epsilon^2\alpha}) = O(d(v)\log n)$ times random walks from each node v, where the length of the walks are respectively drawn from Geom(α) independently. Note that, the length of the random walk with a length larger than $4\log_{\frac{1}{1-\alpha}} n == \frac{4\log n}{\log\frac{1}{1-\alpha}} = O(\log n)$ has probability $(1-\alpha)^{4\log_{\frac{1}{1-\alpha}} n} = \frac{1}{n^4}$.

In total we need to perform $O(\sum_{v\in V} d(v)\cdot\frac{\log n}{\epsilon^2\alpha}) = O(n^2\log n)$ random walks. Hence, by union bound, the case that there are at least one random walks with length larger than $4\log n$ would happen with probability at most

$$O\left(\frac{1}{n^4}\cdot n^2\log n\right) < \frac{1}{n}$$

when n is enough large.

Therefore, with a probability at least $1-\frac{1}{n}$, every random walk has a length at most $4\log_{\frac{1}{1-\alpha}} n = O(\log n)$. To compute the PageRank, therefore, we can invoke $O(\frac{\log n}{\epsilon^2\alpha})$ runs of MRP (Algorithm 3), each runs with walk length $L = O(\log n)$ to generate $d(v)$ random walks from each node v. As such, there are $O(\frac{\log n}{\epsilon^2\alpha})$ instances of MRP and that they are merged together for concurrently processing in the same rounds. This ensures that it only requires $O(\log\log n)$ [6] rounds to finish with a probability at least $1-\frac{1}{n}$. Since MRP entails a $O(L\cdot\log n) = O(\log^2 n)$ bits edge bandwidth, invoking $O(\frac{\log n}{\epsilon^2\alpha})$ MRP algorithms at the same time will result in an edge bandwidth of $O(\log^3 n)$ bits. Next, we show that this edge bandwidth can be improved to $O(\log^2 n)$, if we assume that the maximum degree d_m is at most $\frac{n}{\log n}$.

To explain, if $d_m \leq \frac{n}{\log n}$, then in the current processing of the $O(\frac{\log n}{\epsilon^2\alpha})$ instances of MRP algorithm, in each round each node v sends/receives $O(\frac{\log n}{\epsilon^2\alpha}\cdot d(v)L) = O(\frac{\log n}{\epsilon^2\alpha}\cdot\frac{n}{\log n}L) = O(nL)$ messages, i.e., $O(n\log n)$ messages with high probability. By Lemma 3, with high probability, the operations can be guaranteed to be finished in $O(1)$ rounds, with an edge bandwidth of $O(\log^2 n)$ bits. To conclude, with a probability at least $1-\frac{1}{n}$, the MRP algorithm computes the PageRank in $O(\log\log n)$ rounds with an edge bandwidth of $O(\log^2 n)$ bits in the congested clique model. This algorithm is summarized as follows.

Algorithm 4 (Compute PageRank).

- Invoking $O(\frac{\log n}{\epsilon^2\alpha})$ instances of Algorithm 3 simultaneously. All together, each node v generates $N_v = O(\frac{\log n}{\epsilon^2\alpha}\cdot d(v))$ random walks.
- Estimate $\pi(v)$ by $\frac{1}{|V|}\cdot\sum_{\text{walk from } s \text{ to } v}\frac{1}{N_s}$.

Extensions to BPPR

Very recently, there is another variant of PageRank, called BPPR, that finds interesting applications in recommendation systems (Luo et al. 2019). BPPR is defined based on the Personalized PageRank (PPR), which is defined as follows.

Definition 1 (PPR). *Given a pair of nodes (s,t), the PPR value of node t with respect to s, denoted by $\pi(s,t)$, is defined as the probability that an α-decay random walk from s would terminate at t.*

BPPR (Batch One-Hop Personalized PageRanks) is defined as follows.

Definition 2 (BPPR). *Given a threshold δ, an error bound ϵ, and a failure probability p_f, an approximate batch one-hop PPR query returns an estimated PPR $\hat{\pi}(s,v)$ for every node pair (s,v) such that $s\in V$ and v is an out-neighbor of s, such that for all $\pi(s,v)\geq\delta$,*

$$|\pi(s,v) - \hat{\pi}(s,v)| \leq \epsilon\cdot\pi(s,v) \qquad (5)$$

holds with a probability at least $1-p_f$.

We show that our algorithm can be applied to compute BPPR very naturally and efficiently. To explain, in (Luo et al. 2019) there is a lemma (summarized as Lemma 6) says that it can effectively approximate the BPPR by letting each node v generate $O(d(v)\log n)$ α-decay random walks.

Lemma 6. *Let node s generate $\omega = O(\frac{d(s)\log n}{\alpha\cdot(1-\alpha)\epsilon^2})$ random walks. If there are $c(v)$ random walks end at a neighboring*

[6]Careful readers may notice that we omit the factors about ϵ and α in complexities as we treat them as given constants. However, it is fairly convenient to add those factors back to the complexities.

node v of s, then $\tilde{\pi}(s,v) = \frac{c(v)}{\omega}$ is an unbiased estimator of $\pi(s,v)$, such that Equation 5 holds with a probability at least $1 - \frac{1}{n}$, for $\delta = 0$.

Lemma 6 interestingly points out that, to compute BPPR, the number of random walks started at node v should be proportional to $d(v)$. Precisely, the number of walks started at node v is $O(d(v)\log n)$. This naturally suits the unit tasks performed by MRP, and it is easy to see that computing BPPR requires the same complexities as those of computing the PageRank, which is, with a probability at least $1 - \frac{1}{n}$, the algorithm finishes in $O(\log\log n)$ rounds with an edge bandwidth of $O(\log^2 n)$ under the condition that $d_m \leq \frac{n}{\log n}$. The algorithm is described as follows.

Algorithm 5 (Compute BPPR).

- Invoking $O(\frac{\log n}{\epsilon^2\alpha(1-\alpha)})$ runs of Algorithm 3 simultaneously. All together, each node v generates $N_v = O(\frac{\log n}{\epsilon^2\alpha(1-\alpha)} \cdot d(v))$ random walks.
- Estimate $\pi(s,v)$ by $\sum_{\text{walk from } s \text{ to } v} \frac{1}{N_s}$.

Conclusion

In this paper we present improved distributed PageRank algorithms based on a kind of congested clique model. Particularly, given an undirected connected graph G of n nodes, we propose an algorithm, called MRP, that finishes computing approximate PageRank vector in $O(\log\log n)$ rounds with a probability at least $1 - \frac{1}{n}$. This significantly improves over the state-of-the-art approaches. We also present solutions to optimize the edge bandwidth as well as to extend the techniques for computing the BPPR, which is another variant of batch Personalized PageRanks that has found applications in practice.

Acknowledgements

I sincerely thank Dr. Xiaowei Wu for his contributions on Lemma 2 and Lemma 5, and his comments that greatly improved the manuscript.

References

Ahmadi, B.; Kersting, K.; and Sanner, S. 2011. Multi-evidence lifted message passing, with application to pagerank and the kalman filter. In *IJCAI*, 1152–1158.

Das Sarma, A.; Nanongkai, D.; Pandurangan, G.; and Tetali, P. 2010. Efficient distributed random walks with applications. In *PODC*, 201–210.

Das Sarma, A.; Nanongkai, D.; and Pandurangan, G. 2009. Fast distributed random walks. In *PODS*, 161–170.

Dean, J., and Ghemawat, S. 2008. Mapreduce: simplified data processing on large clusters. *Communications of the ACM* 51(1):107–113.

Drucker, A.; Kuhn, F.; and Oshman, R. 2014. On the power of the congested clique model. In *PODC*, 367–376.

Florescu, C., and Caragea, C. 2017. A position-biased pagerank algorithm for keyphrase extraction. In *AAAI*, 4923–4924.

Fujiwara, Y.; Nakatsuji, M.; Shiokawa, H.; Mishima, T.; and Onizuka, M. 2013. Fast and exact top-k algorithm for pagerank. In *AAAI*, 1106–1112.

Ghaffari, M., and Parter, M. 2016. MST in log-star rounds of congested clique. In *PODC*, 19–28.

Guo, T.; Cao, X.; Cong, G.; Lu, J.; and Lin, X. 2017. Distributed algorithms on exact personalized pagerank. In *SIGMOD*, 479–494.

Hegeman, J. W., and Pemmaraju, S. V. 2015. Lessons from the congested clique applied to mapreduce. *Theoretical Computer Science* 608:268–281.

Hegeman, J. W.; Pandurangan, G.; Pemmaraju, S. V.; Sardeshmukh, V. B.; and Scquizzato, M. 2015. Toward optimal bounds in the congested clique: Graph connectivity and MST. In *PODC*, 91–100.

Hegeman, J. W.; Pemmaraju, S. V.; and Sardeshmukh, V. B. 2014. Near-constant-time distributed algorithms on a congested clique. In *DISC*, 514–530.

Jurdziński, T., and Nowicki, K. 2018. MST in O(1) rounds of congested clique. In *SODA*, 2620–2632.

Klauck, H.; Nanongkai, D.; Pandurangan, G.; and Robinson, P. 2015. Distributed computation of large-scale graph problems. In *SODA*, 391–410.

Lenzen, C. 2013. Optimal deterministic routing and sorting on the congested clique. In *PODC*, 42–50.

Luo, S.; Luo, Y.; Zhou, S.; Cong, G.; and Guan, J. 2012. Disks: a system for distributed spatial group keyword search on road networks. *PVLDB* 5(12):1966–1969.

Luo, S.; Luo, Y.; Zhou, S.; Cong, G.; Guan, J.; and Yong, Z. 2014. Distributed spatial keyword querying on road networks. In *EDBT*, 235–246.

Luo, S.; Xiao, X.; Lin, W.; and Kao, B. 2019. Efficient batch one-hop personalized pageranks. In *ICDE*.

Neumann, M.; Ahmadi, B.; and Kersting, K. 2011. Markov logic sets: Towards lifted information retrieval using pagerank and label propagation. In *AAAI*, 447–452.

Ponzetto, S. P., and Strube, M. 2007. Deriving a large scale taxonomy from wikipedia. In *AAAI*, volume 7, 1440–1445.

Sarma, A. D.; Molla, A. R.; Pandurangan, G.; and Upfal, E. 2015. Fast distributed pagerank computation. *Theoretical Computer Science* 561:113–121.

Sarma, A. D.; Gollapudi, S.; and Panigrahy, R. 2011. Estimating pagerank on graph streams. *Journal of the ACM* 58(3):13:1–13:19.

Sarma, A. D.; Molla, A. R.; and Pandurangan, G. 2012. Near-optimal random walk sampling in distributed networks. *arXiv preprint arXiv:1201.1363*.

Zhu, Y.; Ye, S.; and Li, X. 2005. Distributed pagerank computation based on iterative aggregation-disaggregation methods. In *CIKM*, 578–585.