Distributively Computing Random Walk Betweenness Centrality in Linear Time

Qiang-Sheng Hua, Ming Ai, Hai Jin, Dongxiao Yu, Xuanhua Shi
Services Computing Technology and System Lab/Big Data Technology and System Lab/Cluster and Grid Computing Lab,
School of Computer Science and Technology, Huazhong University of Science and Technology, P.R. China.

***Abstract*—Betweenness centrality of a node represents its influence over the spread of information in the network. It is normally defined as the ratio of the number of shortest paths passing through the node among all shortest paths. However, the spread of information may not just pass through the shortest paths which is captured by a new measure of betweenness centrality based on random walks [1]. The random walk betweenness centrality of a node means how often it is traversed by a random walk between all pairs of other nodes. In this paper, we propose an $O(n \log n)$ time distributed randomized approximation algorithm for calculating each node's random walk betweenness centrality with an approximation ratio $(1-\epsilon)$ where n is the number of nodes and ϵ is an arbitrarily small constant between 0 and 1. Our distributed algorithm is designed under the widely used $\mathcal{CONGEST}$ model, where each edge can only transfer $O(\log n)$ bits in each round. To our best knowledge, this is the first distributed algorithm for computing the random walk betweenness centrality. Moreover, we give a non-trivial lower bound for distributively computing the exact random walk betweenness centrality under the $\mathcal{CONGEST}$ model, which is $\Omega(\frac{n}{\log n} + D)$ where D is the network diameter. This means exactly computing random walk betweenness cannot be done in sublinear time.**

I. Introduction

In order to quantify the importance of a node in the network, various centraility indices have been proposed and they are playing an important role in network analysis [2]. Among these centrality indices, the study of betweenness centrality has garnered an increasing attention due to its wide applications [3] and its inherent high computational complexity [4]. A node's betweenness centrality value can be roughly regarded as its influence over the spread of information in the network. There are mainly two types of definitions of a node's betweenness centrality: One is called the shortest path based betweenness centrality (abbreviated as shortest path betweenness) [5] which only accounts for the spread of information along the shortest paths between each pair of nodes; If the information is spread not just on the shortest paths but via random walks, this is called the random walk based betweenness centrality (abbreviated as random walk betweenness) [1]. The random walk betweenness was first proposed by Newman in 2005 and since then it has been widely employed in the network analysis community [6].

Formally, the *shortest path betweenness* of v (denoted as $C_B(v)$) is defined as $C_B(v) = \sum_{s \neq t \neq v} \sigma_{st}(v)/\sigma_{st}$ where σ_{st} indicates the number of shortest paths from s to t and $\sigma_{st}(v)$ indicates the number of shortest paths from s to t passing through v.

The state-of-the-art centralized algorithm to compute shortest path betweenness is the Brandes' Algorithm [4], which can calculate all the nodes' shortest path betweenness centralities in $O(nm)$ time where n is the number of nodes and m is the number of edges in the unweighted graph. The time complexity could be $O(n^3)$ which is unacceptable for large graphs with hundreds of millions of nodes. In our previous work, we have proposed an $O(n)$ time distributed approximation algorithm to compute the shortest path betweenness with approximation ratio $(1 \pm \frac{1}{n^c})$ (c is a constant) under the $\mathcal{CONGEST}$ model where each edge can only transfer $O(\log n)$ bits in each round [5]. We have also proved that the lower bound of distributively computing shortest path betweenness is $\Omega(\frac{n}{\log n} + D)$ where D is the network diameter, indicating the proposed distributed algorithm for shortest path betweenness is nearly optimal.

As shown in the definition of the shortest path betweenness, it only considers the shortest paths for flowing the information. However, when a node starts propagating the information, it might not know the shortest path between a source and a destination which makes it difficult to decide which node for routing. In addition, as indicated in [7], in most networks, the information indeed is not spread only along the shortest paths. Taking Fig. 1 as an example, nodes A and B have high shortest path betweenness centralities, since every shortest path between the two groups passes through node A and node B. On the other hand, node C does not lie on any shortest path between the two groups, so it has a low shortest path betweenness centrality. However, in most realistic situations, node C would play an important role in information flows. It is possible that information does not only flow through the shortest paths but also through other paths.

In respond to the above observation, Newman proposed a new measure of betweenness centrality based on random walks, i.e., the information is flowing along the random walks [1] instead of restricting on only shortest paths. Roughly speaking, the random walk betweenness of some node i is the total number of times the random walk from the source node s to the destination node t passing through it, averaged over the random walks for all pairs s and t.

Since computing random walk betweenness needs to know all possible paths between two arbitrary nodes, it is harder than computing shortest path betweenness. In [1], Newman proposed an $O((n+m)n^2)$ time algorithm to compute random walk betweenness by using matrix operations. The time complexity could be $O(n^4)$, making it unacceptable for large

1063-6927/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDCS.2017.287

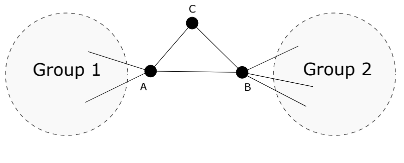

Fig. 1. Nodes A and B have high shortest path betweenness centralities in this graph but node C does not.

graphs. In our previous paper [5], we have shown it is possible to distributively computing shortest path betweenness in linear time. Thus, a natural question is **if we can also devise a linear time distributed algorithm to compute random walk betweenness.** In this paper, we answer this question positively but using a completely different method from [5].

Notice that the trivial method that asking a designated node to collect all the other nodes' neighbors information and then letting the node calculate the betweenness centrality values locally cannot get efficient algorithms, as this method needs $O(m)$ time under the $\mathcal{CONGEST}$ model.

In summary, our contributions are as follows:

1) We propose an $O(n \log n)$ time distributed approximation algorithm to compute random walk betweenness centralities of all nodes with an approximation ratio $(1-\epsilon)$ where ϵ is an arbitrarily small constant between 0 and 1. To our best knowledge, this is the first non-trivial distributed algorithm for computing the widely used random walk betweenness centrality.
2) We also prove an $\Omega(\frac{n}{\log n} + D)$ lower bound for exactly computing random walk betweenness under the $\mathcal{CONGEST}$ model, no matter whether the algorithm computing it is deterministic or randomized. This result means that exactly computing random walk betweenness cannot be done in sublinear time.

The remainder of this paper is orgnized as following: The related works are given in section II. The system model and the problem definition are given in section III. In section IV, we introduce the matrix expressions of random walk betweenness first proposed by Newman. The challenges of designing a distributed random walk betweenness algorithm are introduced in section V. In section VI, we propose our distributed algorithms under the $\mathcal{CONGEST}$ model. We analyse the correctness and the efficiency of our algorithms in section VII. In section VIII, a non-trivial lower bound of distributively computing random walk betweenness is proposed. We conclude the paper in section IX.

II. Related Works

In this section, we will first introduce three centrality indices closely related with random walk betweenness, including network-flow betweeness centrality, pagerank and α-current flow betweenness centrality; then we will introduce recent works on distributed random walk algorithms.

A. Network-Flow Betweenness Centrality

The network-flow betweenness centrality was proposed by Freeman *et al.* in [8]. A node's network-flow betweenness is defined as the amount of network flow through it when the maximum flow is propagated from the source node s to the destination node t, averaged over all pairs of s and t. Like random walk betweenness, the network-flow betweenness does not only take the shortest paths into account, but also the other paths. However, as seen in the definition, if a node wants to calculate the network-flow betweenness, it must "know" the ideal route, i.e., the maximum flow needs to be known. But in reality, the information is not always propagated by the ideal route.

Since the maximum flow problem can be calculated in $O(m^2)$ time by using the augmenting path method in [9], all nodes' network-flow betweenness centralities can be calculated in time $O(nm^2)$. For distributed algorithms, we can use the very recent result in [10] to solve the approximate maximum flow problem in time $O((D+\sqrt{n}) \cdot n^{o(1)})$, and then to compute the approximate network-flow betweenness centralities for all nodes in time $O((D+\sqrt{n}) \cdot n^{1+o(1)})$.

B. Pagerank

A similar definition with random walk betweenness is pagerank, which was proposed by Larry Page *et al.* in [11]. The pagerank of all nodes is calculated by the *stationary distribution* vector of the following random walk: with probability ϵ, the random walk starts at a uniformly selected node, and with probability $(1-\epsilon)$, the random walk flows from the current node to a random neighbor.

A typical centralized algorithm to compute pagerank is as following: Each node holds N random walks starting at it and these random walks have stop probability ϵ. Then each node estimates its own pagerank as a fraction of N random walks ending at it (cf. Algorithm 2 in [12]). As for distributed algorithms, Sarma *et al.* proposed a distributed algorithm to compute PageRank in $O(\log n/\epsilon)$ time w.h.p (with probability greater than $1 - 1/n^c$ where $c \geq 1$) under the $\mathcal{CONGEST}$ model, where ϵ is the reset probability used in PageRank computation which is a fixed constant [13].

Comparing to the definition of random walk betweenness centralities whose random walk lengths are infinite, the lengths of random walks in pagerank computation are much smaller. We can easily find the expectation lengths of the random walks are $1/\epsilon$. Thus, the problem to compute the random walk betweenness is more difficult than to compute pagerank. And we cannot simply use the distributed algorithms for computing pagerank to calculate the random walk betweenness centrality.

C. α-Current Flow Betweenness Centrality

The random walk betweenness can be also called the current flow betweenness due to its analogy to current flow [1]. In [14], the authors proposed a variant of current flow betweenness called α-current flow betweenness centrality. In this new measure, random walk betweenness is given a parameter α indicating that for all random walks starting at a node, only the fraction of α of them do random moves to its neighbor(s), which can bring down the high cost of computing random walk betweenness.

For computing the α-current flow betweenness, the authors in [14] proposed an $O(m \log n\epsilon^{-2} \log \epsilon / \log \alpha)$ time centralized approximation algorithm within an absolute error of ϵ with arbitrarily high fixed probability. For distributed algorithms, since the definition of the α-current flow betweenness is in the spirit of pagerank, we can use the techniques in [13] to distributively compute α-current flow betweenness in time $O(\log n/(1-\alpha))$.

D. Distributed Algorithms for Random Walk

There are some distributed algorithms for computing random walk which is to output the destination node ID when performing a given l length random walk from a starting node under the $\mathcal{CONGEST}$ model. The authors in [15] gave a distributed algorithm with time $\tilde{O}(\sqrt{lD})$ where $\tilde{O}$ means a polylog(n) factor is hidden and D is the network diameter. The key insight of this algorithm is that instead of performing the long length random walks, it performs many short length random walks simultaneously. Then "stitching" the short random walks together to get the destination node ID. They further extended their algorithm into performing k independent random walks in time $\tilde{O}(\sqrt{klD}+k)$, and proved the lower bound of performing l-length random walk is $\Omega(\sqrt{l})$. In [16], Nanongkai *et al.* further proved an unconditional lower bound of $\Theta(l)$ length random walks in networks with diameter D is $\Omega(\sqrt{Dl}+D)$ when $D \leq l \leq (n/(D^3 \log n))^{1/4}$, which indicates the upper bound in [15] is optimal.

Compared with the random walk betweenness problem, the distributed algorithms for the random walk problem cannot be easily utilized due to the following reasons: (1) The random walk problem only asks to output the destination node ID, but the random walk betweenness problem asks each node to count the number of times the random walk passing through it. (2) The lengths of random walks in random walk betweenness computation can be infinite so that we cannot divide them into short lengths random walks.

III. System Model and Problem Definition

A. System Model

Given an undirected graph $G=(V,E)$, where V ($|V|=n$) denotes the set of nodes and E ($|E|=m$) means the set of edges. Each node can be represented by an $O(\log n)$-bit size unique identifier (ID). The node $u \in V$ can only directly communicate with its neighbor $v \in V$ where $\{u,v\} \in E$. If $u \in V$ needs to communicate with $v \in V$ where $\{u,v\} \notin E$, the message must be propagated along the nodes of a path from $u \in V$ to $v \in V$.

In the system, we assume synchronous communications and each node performs the distributed algorithm on its own. The communication happens on the synchronized and discrete pulses and the time between two successive pulses is denoted as a *round*. In each round, each node can only send an $O(\log n)$-bit size message to its neighbors. In addition, the system limits that only a constant number of messages can be transmitted on each edge in each round, i.e., the bandwidth of each edge is also $O(\log n)$. This communication model is called $\mathcal{CONGEST}$ and it has been widely recognized in the distributed computing community [17]. Compared with the $\mathcal{LOCAL}$ model where each edge can transmit unbounded size of messages, designing low time complexity distributed algorithms under the $\mathcal{CONGEST}$ model faces a big challenge.[1] Here the *time complexity* of an algorithm means the number rounds the algorithm used when all nodes terminate. Note that the time complexity does not take each node's local computation time into account.

B. Problem Definition

Consider about a "message", which could be any kind of information generated by a node s in a network, it intends to go to a target node t, but it does not know where t is. So it only performs random moves until it finds itself at target node t. Thus, in each round, the "message" randomly chooses an adjacent node with uniform probability from its current node. This is the so called "Random walk".

The random walk betweenness centrality of node i is the expected net number of times the random walk passes it for a node pair s,t, averaged over all s,t pairs. By "net" we mean if a walk passes through a node and then passes back through it later in the opposite direction, the two cancel out and there is no contribution to the betweenness. This is necessary since we need to guarantee that a node cannot increase its random walk betweenness by simply let a random walk pass it back and forth.

Besides proposing the random walk betweeness measure, Newman also presented matrix expressions of this new measure to facilitate its calculations [1]. Our distributed algorithms will be built upon these matrix expressions. However, we need to stress that devising an efficient distributed algorithm on the matrix expressions will be a non-trivial task where the details will be given in sections V and VI.

IV. Matrix Expressions of Random Walk Betweenness

Given a network whose adjacency matrix is denoted by $\mathbf{A}$, where:

$$A_{ij} = \begin{cases} 1 & \text{there is an edge between } i \text{ and } j, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Note that $\sum_j A_{ij} = d(i)$ where $d(i)$ is the degree of i. Consider an "absorbing random walk" starting from s and will terminate when it arrives at a target t. Before the absorting random walk terminates, it does random moves around the network. If the random walk finds itself on node j at some point, then the probability it will appear on node i is given by:

$$M_{ij} = \frac{A_{ij}}{d(j)}, \text{ for } j \neq t. \tag{2}$$

In matrix notation, we can write $\mathbf{M} = \mathbf{A}\mathbf{D}^{-1}$, where $\mathbf{D}$ is diagonal matrix with $\mathbf{D}_{ii} = d(i)$.

[1]Most distributed problems can be easily solved in $O(D)$ rounds in the $\mathcal{LOCAL}$ model since a node can collect the whole graph information in $O(D)$ rounds. This is not true under the $\mathcal{CONGEST}$ model.

In Eq. 2, the only exception is when $j = t$, the probability will be 0, since when random walk arrives at t, it will be absorbed (this random walk will stop at t). So $M_{it} = 0$ for all i. Thus, we can remove the t-th row from matrix $\mathbf{M}$. And the t-th column can also be removed without affecting any transitions between other nodes. Let $\mathbf{M}_t$ be the matrix with all these elements removed, and similarly for $\mathbf{A}_t$ and $\mathbf{D}_t$.

Let $\mathbf{M}_t^r$ denote the r-th power of matrix $\mathbf{M}_t$. Now for a random walk starting from s, the probability it finds itself at j after r steps is given by $[\mathbf{M}_t^r]_{js}$, the probability it moves to a random adjacent node i in next step is denoted by $d(j)^{-1}[\mathbf{M}_t^r]_{js}$. The total probabilities from j to i is computed by summing over all values of r from 0 to ∞, i.e., the length of random walk is from 0 to ∞. So we get the total probabilities each node is visited by the random walks starting from s as vector $\mathbf{R}$:

$$\mathbf{R} = \mathbf{D}_t^{-1} \cdot (\mathbf{I} - \mathbf{M}_t)^{-1} \cdot \mathbf{s} \quad = (\mathbf{D}_t - \mathbf{A}_t)^{-1} \cdot \mathbf{s} \tag{3}$$

where each element s_i in the source vector $\mathbf{s}$ is defined by:

$$s_i = \begin{cases} 1 & \text{when i = s,} \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

But we find there is a node t which is 0 (since random walks arriving at t are absorbed) is missing in the vector $\mathbf{R}$. To represent this, we add the t-th row and column back to the resulting matrix $(\mathbf{D}_t - \mathbf{A}_t)$ with values all equaling 0. We denote the resulting matrix as $\mathbf{T}$. Let $V_i^{(st)}$ denote the difference between the total probabilities of random walks arriving at i from source s and target t, then $V_i^{(st)}$ is given in terms of elements in $\mathbf{T}$ by:

$$V_i^{(st)} = T_{is} - T_{it}. \tag{5}$$

Now the difference between the total probabilities of a random walk through j and i for source s and target t is given by the absolute difference $|V_i^{(st)} - V_j^{(st)}|$, and this difference can also be regarded as the "net" flow in the random walk betweenness centrality's definition. So the total "net" flows passing through i for source s and target t is:

$$\begin{aligned} I_i^{(st)} &= \frac{1}{2}\sum_j A_{ij}|V_i^{(st)} - V_j^{(st)}| \\ &= \frac{1}{2}\sum_j A_{ij}|T_{is} - T_{it} - T_{js} + T_{jt}| \text{ for } i \neq s, t. \end{aligned} \tag{6}$$

As shown by Eq. 6, this expression does not satisfy the situation when the computing node is the source or the target. But these nodes also have full fraction of random walks which is one unit, and they can be written as:

$$I_s^{(st)} = 1, I_t^{(st)} = 1. \tag{7}$$

Then the random walk betweenness centrality of i is defined as the average of the total probabilities that random walk passing through i over all possible source/target pairs:

$$b_i = \frac{\sum_{s<t} I_i^{(st)}}{\frac{1}{2}n(n-1)}. \tag{8}$$

The notations used in this paper are briefly summarized in Table I.

TABLE I
NOTATIONS AND THEIR DEFINITIONS

Notation	Definition
$\mathbf{A}$	The adjacency matrix of network (Eq.2).
$d(i)$	The degree of node i.
$\mathbf{M}$	The transition matrix of network (Eq.2).
$\mathbf{D}$	The diagonal matrix where $D_{ii} = d(i)$.
$\mathbf{M}_t$	The transition matrix $\mathbf{M}$ without the t-th row and column.
$\mathbf{A}_t$	The adjacency matrix $\mathbf{A}$ without the t-th row and column.
$\mathbf{D}_t$	The diagonal matrix $\mathbf{D}$ without the t-th row and column.
$\mathbf{M}_t^r$	The r-th power of matrix $\mathbf{M}_t$.
$\mathbf{s}$	The source vector. (Eq.4)
$\mathbf{T}_t$	The resulting matrix of $(\mathbf{D}_t - \mathbf{A}_t)^{-1}$.
$\mathbf{T}$	Adding back the t-th row and column to $\mathbf{T}_t$.
$\mathbf{R}$	The resulting matrix defined by Eq.3.
$V_i^{(st)}$	Node i's probabilities difference value for pair (s,t) (Eq.5).
$I_i^{(st)}$	Total "net" flow through i for pair (s,t) (Eq.6 and Eq.7).
b_i	Random walk betweenness centrality of node i (Eq.8).
$\|\|\mathbf{A}\|\|_1$	$\|\|\mathbf{A}\|\|_1$ the 1-norm of a matrix $\mathbf{A}$.
ξ_v^s	The count number of visits from source s on v.

V. Challenges of Distributively Utilizing the Matrix Expressions

Although the matrix expressions can be used to facilitate the calculation of random walk betweenness in a centralized manner [1], designing an efficient distributed algorithm under the $\mathcal{CONGEST}$ model is still a challenging task:

1) It is hard to calculate the inversion of matrix in Eq.3. The reason is computing the inversion of matrix $(\mathbf{I} - \mathbf{M}_t)$ needs to sum up the $\mathbf{M}_t^r$ where r is from 0 to ∞, meaning the length of random walk is from 0 to ∞. If the length of random walk can be ∞, i.e., the random walk will never stop so that we cannot precisely count how many times a node is passed through by the random walk. Meanwhile, if we use the trivial algorithm mentioned before, which asks a designated node to collect all the other nodes' neighbors information and then let the node calculate the betweenness centrality values locally, the time complexity is as high as $O(m)$ which is too time consuming.
2) In the matrix expression, we need to calculate the probability a random walk from a given node to another node. If we transfer the probability value, there will be two issues: First, in each "step" (we call a random walk moves from a node to one of its neighbors as "one step"), each node needs to transfer all the probabilities that random walks from the other nodes to it, which are $n - 1$ messages, to its neighbors. Comparing to the time of doing random walk which is just $O(1)$ in each step, the probabilities transferring operation needs $O(n)$ time, which is too high. Second, the value of the probability can be very small, but each edge can only transfer $O(\log n)$ bits each round, leading to the value cannot be transferred precisely. So we decide to use the random walk process to simulate the procedure of

calculating probabilities. But we need to solve the issue that how many random walks are needed to simulate the probability.

We will show how to slove these challenges in the following sections VI and VII.

VI. Distributed Random Walk Betweenness Algorithms

In this section, we will first give the intuitive ideas and the algorithms overview, and then we will give the detailed algorithms.

A. Algorithms Overview

In the matrix expressions for computing the random walk betweenness, we find that by Eq. 5, the total probabilities ($V_i^{(st)}$) that the random walk passes through a node i for source s and target t can be calculated by the difference between the total probabilities (T_{is}) that a random walk starting at s ending at i and the total probabilities (T_{it}) that a random walk starting at t ending at i. So for each source s, we just need to hold one random walk and let it do random moves around the graph. And for each node, it just needs to compute the total probabilities that the random walks pass through it starting from the other nodes. Thus it obviates the need to compute $O(n^2)$ random walks for each pair (s, t) (cf. Eq.8).

In response to the first challenge, instead of performing infinite length random walks, in our algorithm, we impose a bounded length constraint l (decided later) on the random walks. The main idea of our algorithm is as following. Since it is identical to choose any node as a target node (a random walk will terminate when it arrives at the target node), we just need to choose one node as target t rather than choosing every node as a target. This process is the same as randomly removing the t-th row and the t-th colmun in the matrix experssion. After that, each node i can compute the total probabilities from i to s ($\mathbf{T}_{is}$). In response to the second challenge, we calculate the probability by the fraction between the times that random walks starting at s passing through node i and the total number of K (decided later) random walks starting at s. After each node i computes $\mathbf{T}_{is}$ for all s, it can compute its own $V_i^{(st)}$ by Eq.5. Then, each node can compute the total "net" flows through it by Eq.6. Finally, they can compute their own random walk betweenness by collecting all their neighbor j's $V_j^{(st)}$ by Eq.8.

B. Detailed Algorithms

Our algorithms are broken into two phases, which are shown in Algorithm 1 (the counting phase) and Algorithm 2 (the computing phase) respectively. As shown in Algorithm 1, each node counts the number of times the random walks starting at each source passing through it. In Algorithm 2, each node computes its own random walk betweenness using the counts it gets in the counting phase.

In Algorithm 1, first, every node v maintains a parameter ξ_v^s for each source s, where ξ_v^s indicates the number of times the random walks starting at s passing through v. Then randomly choose a node t as a target, random walks will be absorbed when arriving at it (line 2). Each node holds K random walks for target t, and each random walk has a parameter $length$ which initially is l (line 3). Each random walk knows which node is its source. In each round, each random walk goes to a random neighbor independently with the probability $1/d$ where d is the degree of the node. If there is more than one random walk needed to be sent to the same neighbor, randomly choose one of them and send it to the neighbor (line 6). If that random walk arrives at t, then it will terminate, otherwise the length of it decreases by one. When the length of a random walk becomes zero, it will terminate too (lines 7-9). When a node v is visited by a random walk whose source is s, the count number of visits of source s (i.e. ξ_v^s) increases by 1 (line 10). Since by Eq.6, the "net" flow from a node to its neighbor is calculated by the difference between them, we do not need to record the random walk's direction. When all random walks terminate, every node will hold the number of visits starting at each node. Then each node sends all its count numbers to its neighbors, and uses the count it receives to compute the random walk betweenness (Algorithm 2).

In Algorithm 2, each node first divides all counts it stores by its node degree (line 1). Then each node transfers all the counts to its neighbors (line 2). After that, random walk betweenness can be computed by each node (lines 3-4).

Now the remaining issue is to decide both the l and the K values where l is the random walk length and K is the number of random walks each node should perform. This will be elaborated in the next section.

VII. Algorithm Analysis

In this section, we will analyse the correctness and the efficiency of our algorithms. Specifically, we will first prove when $l = O(n)$, the multiplicative error will be a constant $(1 - \epsilon)$ where ϵ is an arbitrarily small constant between 0 and 1. Then we will prove that when the number of random walks starting at each node is $K = O(\log n)$, each node can estimate its random walk betweenness w.h.p. After proving all the above issues, we will show that our algorithm can compute the random walk betweenness of all nodes in $O(n \log n)$ time w.h.p.

A. Correctness Analysis

We first prove the number of random walks is decreasing during the execution of the algorithm, since the random walks will be absorbed by the target node.

Lemma 1. *Given a random walk starting at a random node s, it will be absorbed when it arrives at node t. Then after D rounds, the total probabilities at each node except for t are less than 1.*

Proof. Suppose the random walk starts at node s. Since the random walk moves to a random neighbor in each round, after D rounds, the probability from s to t is larger than 0, i.e., the total probabilities at the other nodes are smaller than 1. □

Algorithm 1 Each node counts the number of random walks passing through it

```
Input: Number of nodes $n$
Output: For each node $v$ and each source $s \neq t$, the number
    of visits of random walks $\xi_v^s$ starting at $s$.
1: Each node $v$ maintains a variable $\xi_v^s$ to count how many
   times the random walk starting at $s$ passing through it.
   Initially, $\xi_v^s = 0$ indicates there is no random walk starting
   at $s$ has passed through it.
2: Randomly choose a target node $t$.
3: Each node $s$ maintains $K$ random walks starting at it,
   where each random walk maintains $source = s$ and a
   parameter $length = l$ (See Theorem 1 and Theorem 3 in
   section VII for detailed $l$ and $K$ values).
4: while Some random walk does not terminate do
5:     Each node $s$ maintaining at least one random walk
       does the following in parallel:
6:     For each random walk of node $u$, choose a random
       neighbor $v$, then send it to node $v$. If there is more than one
       random walk needed to be sent to $v$, just send a random
       walk to $v$ randomly.
7:     for Each node $u$ receiving random walk $RW$ do
8:         if $u$ is the target node $t$, or $RW.length == 0$ then
9:             The random walk $RW$ terminates.
10:        else
11:            $s = RW.source$
12:            $\xi_v^s = \xi_v^s + 1$
13:        end if
14:        $u$ maintains $RW$.
15:        $RW.length = RW.length - 1$
16:    end for
17: end while
```

Algorithm 2 Compute random walk betweenness

```
Input: For each source $s$, each node $v$ knows how many times
    the random walks strating at $s$ passing through itself ($\xi_v^s$).
Output: Each node $i$ computes its own random walk be-
    tweenness $b_i$.
1: For each node $v$, dividing all the counts on it by its degree
   $d(v)$, i.e., for each source $s$, $\xi_v^s = \xi_v^s / d(v)$.
2: Each node $v$ sends the updated counts to its neighbors
   $N(v)$.
3: Each node $i$ computes $I_i^{(st)}$ as $I_i^{(st)} =
   \frac{1}{2}\sum_{j \in N(i)} |\xi_i^s - \xi_i^t - \xi_j^s + \xi_j^t|$ for all $s, t \neq i$. If
   $i == s$ or $i == t$, $I_i^{(st)} = 1$.
4: Each node $i$ computes its own betweenness centrality $b_i$
   as $b_i = \frac{\sum_{s<t} I_i^{st}}{\frac{1}{2}Kn(n-1)}$.
```

Then in the following Theorem 1, we will prove the approximation ratio is $(1-\epsilon)$ when $l = O(n)$, which gives a solution to the first challenge in section V.

Theorem 1. *Given a random walk starting at a random node s, it will be absorbed when it arrives at node t. Then after $O(n)$ rounds, the remaining fraction of the random walks is at most ϵ in our algorithm.*

Proof. Let $\mathbf{M}$ be a transition matrix of random walk, where $[\mathbf{M}]_{ij}$ expresses the random walk will move to node i when its current position is at node j. We should remove the t-th row and the t-th column of $\mathbf{M}$, since the random walk is absorbed by t when it arrives at t. Denote $\mathbf{M}_t$ as the matrix with these elements removed. Now we assume a random walk starts at s, and denote $[\mathbf{M}_t^r]_{js}$ as the probability at node j after r rounds. Denote $||\mathbf{A}||_1$ as the 1-norm of a matrix $\mathbf{A}$, which is the maximum absolute column sum of the matrix, i.e.:

$$||\mathbf{A}||_1 = \max_{1 \le j \le n} \sum_{i=1}^{n} |A_{ij}|.$$

By Lemma 1 we know:

$$||\mathbf{M}_t^D||_1 < 1, \tag{9}$$

since after D rounds, the total probabilities at each node except for t are less than 1. Denote λ and $\mathbf{x}$ as the eigenvalue and the eigenvector of a matrix, respectively, i.e., $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$. And we have:

$$\mathbf{A}^k\mathbf{x} = \lambda^k\mathbf{x}, \tag{10}$$

which means the eigenvalue of the matrix $\mathbf{A}^k$ is the k-th power of matrix $\mathbf{A}$'s eigenvalue.

By Eq.9 and Eq.10, the matrix's maximum eigenvalue is not greater than its 1-norm. We know the maximum eigenvalue of $\mathbf{M_t}$ is less than one, i.e., all eigenvalues of the matrix $\mathbf{M}_t$ are less than one. Write the matrix $\mathbf{M}_t$ as jordan canonical form:

$$\mathbf{M}_t = \mathbf{P}\mathbf{J}\mathbf{P}^{-1}, \tag{11}$$

where $\mathbf{P}$ is a constant matrix and $\mathbf{J}$ is a block diagonal matrix like:

$$\mathbf{J} = \begin{bmatrix} J_1 & & \\ & \ddots & \\ & & J_k \end{bmatrix},$$

and the block J_i is a square matrix of the form:

$$\mathbf{J}_i = \begin{bmatrix} \lambda_i & 1 & & \\ & \lambda_i & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_i \end{bmatrix},$$

where λ_i is i-th eigenvalue of $\mathbf{M}_t$. So after k rounds, the probability at each node except for t is:

$$\begin{aligned} \mathbf{M}_t^k &= \underbrace{\mathbf{P}\mathbf{J}\mathbf{P}^{-1} \cdot \cdots \cdot \mathbf{P}\mathbf{J}\mathbf{P}^{-1}}_{\text{k}} \\ &= \mathbf{P}\mathbf{J}^k\mathbf{P}^{-1}. \end{aligned} \tag{12}$$

So without loss of generality (w.l.o.g), we will find the maximum element in matrix $\mathbf{J}^k$, and let it equal $\lambda\epsilon$. To find the maximum element in $\mathbf{J}^k$, w.l.o.g., suppose $\mathbf{J}$ has only one $(n-1)\times(n-1)$ block $\mathbf{J}_{\max}$ and $k \geq n-1$. And we have:

$$\mathbf{J}^k = \begin{bmatrix} \lambda_{\max}^k & \binom{k}{1}\lambda_{\max}^{k-1} & \cdots & \binom{k}{n-2}\lambda_{\max}^{k-n+2} \\ & \lambda_{\max}^k & \binom{k}{1}\lambda_{\max}^{k-1} & \ddots \\ & & \ddots & \binom{k}{1}\lambda_{\max}^{k-1} \\ & & & \lambda_{\max}^k \end{bmatrix},$$

where $\binom{k}{x} = \prod_{i=1}^{x} \frac{k+1-i}{i}$ is the binomial coefficients.

Since $\binom{k}{x} \leq (\frac{ek}{x})^x$:

$$\begin{aligned}\binom{k}{x}\lambda^{k-x} &\leq (\frac{ek}{x})^x \cdot \lambda^{k-x} \\ &= (\frac{ek}{x\lambda})^x \lambda^k.\end{aligned} \tag{13}$$

To find the maximum value in $\mathbf{J}^k$, we first compute the maximum value in function $f(x) = (\frac{ek}{x\lambda})^x\lambda^k$, where $0 \leq x \leq n-2$. Notice that $f(x)$ is monotonically increasing when $0 \leq x \leq n-2$, so when $x = n-2$, the function gets its maximum $(\frac{ek}{(n-2)\lambda})^{(n-2)}\lambda^k$. To prove the remaining fraction is not greater than ϵ, we let the maximum value equal $\lambda\epsilon$, i.e., we find the value k to let function $f(k) = (\frac{ek}{(n-2)\lambda})^{(n-2)}\lambda^k = \lambda \cdot \epsilon$. Using the common log on both sides gives:

$$\begin{aligned}&\ln((\frac{ek}{(n-2)\lambda})^{(n-2)}\lambda^k) \\ &= (n-2)(1+\ln k - \ln(n-2) - \ln\lambda) + k\ln\lambda \\ &= \ln\epsilon + \ln\lambda.\end{aligned} \tag{14}$$

where λ and ϵ are constants. So we can get $k = O(n)$, i.e., after $O(n)$ rounds, the remaining fraction of random walks is at most ϵ.

□

Theorem 2. *The relative error of random walk betweenness computed by our algorithms is $(1-\epsilon)$.*

Proof. Since after l rounds, which we have proved to be $O(n)$ in Theorem 1, the random walks that have not been absorbed will do random moves until they are absorbed, and we have proved that the fraction of remaining random walks is at most ϵ. So what we do not take into account is the ϵ part of the total random walks, i.e., the relative error is equal to $(1-\epsilon)/1 = (1-\epsilon)$. □

Finally, in the following Theorem 3, we will prove when each node holds $O(\log n)$ random walks, i.e., $K = O(\log n)$, the algorithm will get its result w.h.p. This gives a solution to the second challenge in section V.

Theorem 3. *Given a source node s, if there are $O(\log n)$ random walks starting at it, then each node will get the expected number of times that a random walk staring at s passing through it w.h.p.*

Proof. Denote the random walks starting at s passing through i as X, the expected number of times a random walk starting at s passing through node i is $E(X)$. By the two-sided Chernoff's Bound (e.g. in [18], Corollary 4.6), for any X

$$P[|X - E(X)| \geq \delta E(X)] \leq 2\exp(-\frac{\delta^2}{3}E(X)).$$

Notice that $E(X) = cK$ where c is a constant and K is the number of random walks starting at each node, and δ is an arbitrary constant between 0 and 1. So let $K = c\log n/(\frac{\delta^2}{3}) = O(\log n)$, we get:

$$P[|X - E(X)| \geq \delta E(X)] \leq 2\exp(-c\log n) = 2n^{-c}.$$

It indicates when $K = O(\log n)$, we can simulate the probability from a node to another node w.h.p. □

Theorem 4. *Our algortihms satisfy the $\mathcal{CONGEST}$ model.*

Proof. In Algortihm 1 and Algorithm 2, each message contains $O(\log n)$ bits, and each edge in each round only transfer $O(1)$ messages. So Algortihms 1 and 2 satisfy the $\mathcal{CONGEST}$ model. □

B. Efficiency Analysis

In this subsection, we will analyse the time complexities of our algorithms.

Lemma 2. *Algorithm 1 requires $O(n\log n)$ time.*

Proof. From Theorem 1 and Theorem 3, in Algorithm 1, each node holds $K = O(\log n)$ random walks, and each random walk has length of $l = O(n)$. So the total time of Algorithm 1 is $O(Kn + l) = O(n\log n)$. □

Lemma 3. *Algorithm 2 requires $O(n)$ time.*

Proof. In Algorithm 2, each node holds one count for each source, so the total number of counts is $O(n)$. In each round, each node sends one count to its neighbors, so it takes $O(n)$ time to transfer all the counts to its neighbors. Thus, Algorithm 2 takes $O(n)$ time. □

Theorem 5. *Each node can compute its own approximated random walk betweenness with an approximation ratio $(1-\epsilon)$ in $O(n\log n)$ time.*

Proof. It is obvious from Lemma 2, Lemma 3 and Theorem 2. □

VIII. Lower Bound

In this section, we will give the lower bound of computing random walk betweenness centralities of all nodes under the $\mathcal{CONGEST}$ model, where each edge can only transfer $O(\log n)$ bits in each round. The key insight of the proof is to find the lower bound of deciding the random walk betweenness of a node is z (explained later) or larger than z.

Theorem 6. *For any network with n nodes, any distributed randomized algorithm A computing exact random walk betweenness of any node requires at least $\Omega(\frac{n}{\log n} + D)$ time under the $\mathcal{CONGEST}$ model.*

To prove Theorem 6, we need to give some existing results in "Communication Complexity" (abbreviated as cc) and its use in the lower bound proof for distributed graph problems. First, we give the definition of "Communication Complexity", which in general is to ask how many bits are needed to solve a distributed function computing problem.

Definition 1. *(Two-Party Communication Complexity [19]) Given two players Alice and Bob where Alice has an arbitrary input a and Bob has an arbitrary input b. They intend to compute a function h with an error probability β. Denote the set of two party algorithms using public randomness for computing the function as A_β. Given an algorithm $A \in A_\beta$, we denote the number of bits that Alice and Bob need to exchange on inputs a and b using algorithm A as $R_\beta^{cc-public}(A(a,b))$. And we define:*

$$R_\beta^{cc-public}(h) = \min_{A \in A_\beta} R_\beta^{cc-public}(A(a,b))$$

as the minimum number of bits required to be exchanged by any algorithm for computing h.

We follow a similar approach as in [20], [5] to prove the lower bound for distributed graph problems via the two-party communication complexity.

Definition 2. *(Cut [20]) Given a graph $G = (V, E)$, a cut (G_a, G_b, C_k) is a partition of G into two disjoint subgraphs $G_a = (V_a, E_a)$ and $G_b = (V_b, E_b)$ and a cut $C_k \subseteq E$ such that $V = V_a \dot{\bigcup} V_b$ and $E = E_a \dot{\bigcup} E_b \dot{\bigcup} C_k$, where $\dot{\bigcup}$ means the disjoint union of two sets. The cut C_k contains $c_k := |C_k|$ edges whose two endpoints are in V_a and V_b respectively.*

After introducing the "cut" definition, we now give the connection between the two-party communicaton complexity and the time complexity of distributed graph problems in the following:

Theorem 7. *([20]) Let $B \geq 1$ and f be any function on graphs and f' be the function derived from f. We have*

$$\frac{R_\beta^{cc-public}(f')}{2c_k \cdot B} \leq R_\beta^{dc-public}(f),$$

where $R_\beta^{cc-public}(f')$ represents the two-party communication complexity for computing function f', $R_\beta^{dc-public}(f)$ represents the distributed time complexity for computing function f, c_k is the number of edges in the cut C_k, and B is the number of bits each edge can transfer in each round, which is $O(\log n)$ in the $\mathcal{CONGEST}$ model.

Definition 3 (Sparse Disjointness Problem)**.** *There are two subsets x and y of set $[1, \cdots, n]$, each of which has size k. Then function $DISJ_n^k(x, y)$ equals 1 iff $x \cap y = \emptyset$.*

To prove Theorem 6, we construct a graph illustrated in Fig.2. Given an even number M whose value will be decided later, we first create $2M$ nodes L_1,L_2,$\cdots$,L_M and R_1,R_2,$\cdots$,R_M. We add an edge between each pair L_i and R_i where $i \in 1, 2, \cdots, M$. Then we choose N subsets X_1,X_2,$\cdots$,X_N from $L = \{L_1, L_2, \cdots, L_M\}$, and N subsets Y_1,Y_2,$\cdots$,Y_N from $R = \{R_1, R_2, \cdots, R_M\}$ where $|X_i| = |Y_i| = M/2$.

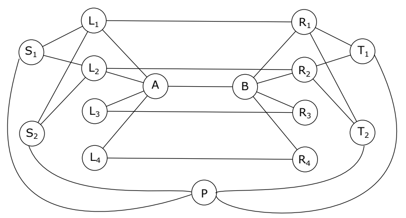

Fig. 2. The graph construction for the lower bound of computing random walk betweenness. In this figure, we set $M = 4$ and $N = 2$. Whether the random walk betweenness of P is z or larger than z depends on the whether the set X is disjoint to the set Y, i.e., each S_i is equal to all T_j. For example, in this figure, S_1 is equal to T_1 and T_2, and S_2 is equal to T_1 and T_2, so $X \cap Y = \emptyset$ and the random walk betweenness of P is the minimum.

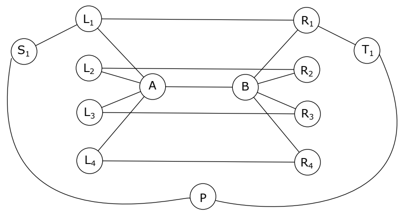

Fig. 3. A special case for $N = 1$, i.e., there is only one node S_1 in X and only one node T_1 in Y. And S_1 is only connected to L_1 and T_1 is only connected to R_1, indicating $S_1 = T_1$.

Denote X as the union of the subsets X_i and Y as the union of the subsets Y_i. For each subset X_i (Y_i), we denote it by a node S_i (T_i) in the graph. When each subset X_i contains an element in L, we create an edge between S_i and the corresponding node L_j. When each subset Y_i does ***NOT*** contain the element in R, we create an edge between T_i and the corresponding node R_j. Since the cardinality of each subset X_i (Y_i) is $M/2$, $M/2$ edges will be added between each S_i (T_i) and L (R).

Denote $S_i = T_j$ iff S_i is connected to those $L_p \in L$ whose corresponding nodes $R_p \in R$ are connected with T_j. Taking Fig.2 as an example, we say $S_1 = T_2$ since S_1 is connected to L_1 and L_2 whose corresponding nodes R_1 and R_2 are connected with T_2. In the middle of the graph, we create two nodes A and B, which are connected to each other. The node A (B) also connects with all nodes L (R). Finally, for each node S_i (T_i), we create an edge from it to a node P.

Lemma 4. *With a graph constructed above, we can verify that the random walk betweenness b_P of node P as:*

$$b_P = \begin{cases} z & X \cap Y = \emptyset \\ > z & otherwise. \end{cases}$$

Proof. Denote random walk betweenness of P is z when $X \cap Y = \emptyset$. We first prove that the random walk betweenness of P gets the minimum value when $S_i = T_i$ (i.e. $X \cap Y = \emptyset$) in the special case where $N = 1$, i.e., there is only one node S_1 (T_1) in the set X (Y), and there is only one edge between X and L (Y and R).

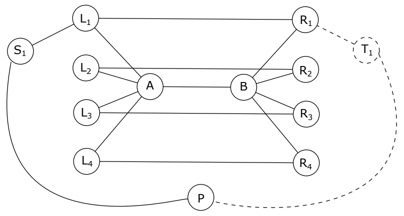

Fig. 4. The graph with the target node T_1 removed. In order to compute the random walk betweenness of P, the target node T_1 can be removed since the random walks will be absorbed when they arrive at T_1.

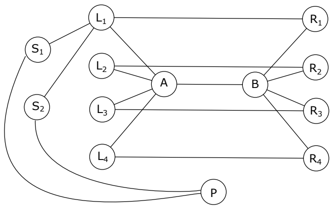

Fig. 5. A special case that there is one node S_1 in X connecting to node L_1, and we want to add a new node S_2 into set X connecting to an arbitrary node in L.

Lemma 5. *b_P gets the minimum value when $S_i = T_i$ in the special case where $N = 1$ and there is only one edge between X and L, so does Y and R.*

Proof. As shown in Fig.3, there is only one node S_1 (T_1) in set X (Y). W.l.o.g., we connect the node S_1 with L_1. So T_1 is connected to R_1 if $S_1 = T_1$, or to other node R_i where $i \neq 1$ in R. Then we will show that the random walk betweenness of P gets the minimum value when T_1 is connected to R_1.

We will use the definition to compute the random walk betweenness of P. First, we remove the node T_1 and the edges connected to it from the graph as shown in Fig.4. To compute the random walk betweenness of P, we need to sum the probabilities the random walks reach it and its neighbors from all the other nodes, then get the result by using Eq.8. Since after T_1 is removed, the random walks from all the other nodes to P must pass through S_1. And the degree of S_1 is 2, so the probability the random walks reach S_i is as twice as they reach P. By Eq.6, we can compute the random walk betweenness by the probabilities that the random walks reach S_1.

Comparing the situation T_1 connecting to R_1 with the situation T_1 connecting to other node R_i rather than R_1, the differences are the degrees of node R_1 and R_i. In the first situation, the degrees of R_1 and R_i are 3 and 2 respectively. And in the second situation, the degrees of R_1 and R_i are 2 and 3 respectively. In these two situations, notice that the probabilities the random walks reaching S_1 are different if they pass through node R_1 or R_i on its journey. And if the random walks do not pass through R_1 or R_i, the probabilities they reach S_1 are the same. Thus we only need to consider the random walks starting from R_1 and R_i, ending at S_1.

Consider the graph without T_1 and all edges connecting to it. Assume a random walk starting at R_1, then the probabilities it appears at node L_1 and node B are both $1/2$ (we do not take the influence of T_1 into account). If a random walk starts at R_i where $i \neq 1$, the probabilities it appears at node L_i and node B are both $1/2$. Since the probabilities at B are both $1/2$ in the next step, we only need to compare the probability from L_1 to S_1 with the probability from L_i to S_1. Since all the paths ending at S_1 must pass through node L_1, except for those whose source is P or S_1, the probability from L_1 to S_1 must be larger than the probability from L_i to S_1. So the probability a random walk from R_1 to S_1 is larger than it from R_i ($i \neq 1$) to S_1.

Now consider the situation in Fig.4. If node T_1 is connected to node R_1, then the probability from R_1 to T_1 is $1/3$, i.e., the probability that a random walk starting at R_1 is absorbed by T_1 is $1/3$. Thus the probability that a random walk from R_1 to S_1 decreases by $1/3$. Similarly, if T_1 is connected to R_i, then the probability from R_i to S_1 decreases by $1/3$. Comparing these two situations, the probability a random walk reaches S_1 decreases more when T_1 is connected to R_1, since the probability from R_1 to S_1 is larger than R_i to S_1. Therefore, node P's random walk betweenness is the minimum when T_1 is connected with R_1, i.e., when $S_1 = T_1$. □

If there is some edge from X to L and we add a new node S_i into X, then we prove that the random walk betweenness of P will have the minimum value when S_i is connected to L_p which had edge(s) with X before the addition.

Lemma 6. *If we add a node S_i to set X where X is not an empty set, the random walk betweenness of P will be the minimum when S_i is connected to a node $L_p \in L$ which had edge(s) with X before the addition.*

Proof. As shown in Fig.5, w.l.o.g., we assume there has been a node $S_1 \in X$ connecting to $L_1 \in L$, and we add a node S_2 into X. To make the random walk betweenness of P be the minimum, we will prove S_2 will be added to connect with L_1.

Assume there is a random walk starting at each node. Since the graph is symmetric, the sum probabilities the random walks ending at each node in L is identical, except for the random walks starting at S_1, S_2 and P. Since we have shown we need to compute the probabilities to S_1, S_2 in Lemma 5, we will compare the probabilities to them under the two situations: S_2 is connected to L_1, and S_2 is connected to other node $L_i \in L$ where $i \neq 1$.

Suppose the sum of probabilities that the random walks ending at each node in L, except for whose source is S_1, S_2 or P, are all equal to p. The probability to S_1 equals $p/3$ before S_2 is added. If S_2 is connected to L_1, p will not change since the random walks whose source is not S_1, S_2 and P do not pass through S_1 or S_2 before they reach L_i. The probability to S_1 becomes $p/4$ since the degree of L_1 becomes 4, and the probability to S_2 is also $p/4$. Then the sum of probabilities to S_1 and S_2 is $p/2$. If S_2 is connected to L_i where $i \neq 1$, the probability to S_1 is unchanged ($p/3$), and the probability to S_2 is also $p/3$. Then the sum of probabilities to S_1 and S_2 is $2p/3$, which is larger than the situation when S_2 is connected to L_1. So the random walk betweenness of P will be the minimum when S_i is connected to a node $L_p \in L$ which had edge(s) with X before its addition. $\square$

Combining Lemma 5 and Lemma 6, we can deduce that the random walk betweenness of P gets the minimum value when all $S_i = T_j$, i.e., $X \cap Y = \emptyset$. Assume that we have only one node S_1 in set X where S_1 is connected $M/2$ nodes in L. If we want to get the the minimum random walk betweenness of P after adding one node to Y, we need to add a node in Y which is connected to the corresponding nodes in R by Lemma 5. And if we want to get the the minimum random walk betweenness of P after adding one node into X, we need to add a node which is connected to those connected nodes in L by Lemma 6. So the random walk betweenness of P gets the minimum value when $X \cap Y = \emptyset$.

$\square$

In order to determine the value of M and N, we notice that when $M = O(\log N)$, $\binom{M}{M/2} \geq N^2$. So the number of nodes in the network $n = 2N + 2M + 3 = O(N)$.

Theorem 8. *Suppose Alice and Bob both have a set consisting of N numbers in range $\{1, 2, \ldots, N^2\}$, if they want to evaluate whether their sets are disjoint, they need to exchange at least $\Omega(N \log N)$ bits.*

Proof. From Definition 3, a disjoint function $DISJ^N_{N^2}(x, y)$ can be derived where x and y are both subsets of $\{1, 2, ..., N^2\}$ with size N. Then we can obtain $R^{cc-public}_\beta(DISJ^N_{N^2}) = \Omega(N \log N)$ from [21]. As a result, Alice and Bob need to exchange $\Omega(N \log N)$ bits to evaluate whether x and y are disjoint. $\square$

Corollary 1. *If Alice has a set $X = \{X_1, X_2, \cdots, X_N\}$ and Bob has a set $Y = \{Y_1, Y_2, \cdots, Y_N\}$ where both X_i ($|X_i| = M/2$) and Y_i ($|Y_i| = M/2$) is a subset of $\{1, 2, ..., N^2\}$. In our graph construction, $M = O(\log N)$. If there are any X_i and Y_j satisfying $X_i \cap Y_j \neq \emptyset$, then $X \cap Y \neq \emptyset$. Now if Alice and Bob want to evaluate whether $X \cap Y = \emptyset$, they need to exchange $\Omega(N \log N)$ bits.*

Proof. Since both subsets X_i and Y_i satisfy $|X_i| = M/2$ and $|Y_i| = M/2$, we can encode them as a $M/2$ length binary sequence where each bit in the sequence indicates whether the corresponding node is in the subsect or not. For example, in Fig. 2, the subset X_1 which corresponds to the node set $\{L_1, L_2\}$ can be encoded as 1100, and the subset Y_1 which corresponds to the node set $\{R_3, R_4\}$ can be encoded as 0011. Since both sets X and Y have N subsets X_i, the total number of bits of both X and Y are $O(N \log N)$. Then using Theorem 8, which indicates that $R^{cc-public}_\beta(DISJ^N_{N^2}) = \Omega(N \log N)$, we know Alice and Bob need to exchange at least $\Omega(N \log N)$ bits. $\square$

Finally, we will prove that the lower bound of computing random walk betweenness is $\Omega(\frac{n}{\log n} + D)$.

Proof of Theorem 6. Given a graph construction like Fig.2, we denote the random walk betweenness of P as z when $X \cap Y = \emptyset$. If there is a distributed algorithm to compute random walk betweenness of P, Alice and Bob can simulate the algorithm under the construction by using the set disjointness problem. From Theorem 7 we know that

$$\frac{R^{cc-public}_\beta(f')}{2c_k \cdot B} \leq R^{dc-public}_\beta(f),$$

where $R^{cc-public}_\beta(f')$ represents the two-party communication complexity for computing function f', and $R^{dc-public}_\beta(f)$ means the distributed time complexity for computing function f. Notice that in our graph construction like Fig.2, $c_k = M$, and $R^{cc-public}_\beta(f') = \Omega(n/\log n)$ by Corollary 1 ($n = O(N)$). Since $B = O(\log n)$ (the $\mathcal{CONGEST}$ model), and the time that a message flows from one node to another node is at most $O(D)$, the time complexity to distributively compute the random walk betweenness of node P is:

$$R^{dc-public}_\beta(f) \geq \Omega(\frac{n \log n}{2 \log n \cdot \log n} + D) = \Omega(\frac{n}{\log n} + D).$$

$\square$

IX. Conclusion

In this paper, by carefully examining the matrix expressions to compute the random walk betweenness centralities, we find the two challenges to distributively utilize the matrix expressions under the widely recognized $\mathcal{CONGEST}$ model where each edge can only transfer $O(\log n)$ bits in each round where n is the number of nodes in the network. Then we propose the first distributed algorithm that overcomes these two challenges. Our distributed approximation algorithm takes $O(n \log n)$ time and the approximation ratio is $(1-\epsilon)$ where ϵ is an arbitrarily small constant from 0 and 1. To our best knowledge, this is the first non-trivial distributed approximation algorithm for computing the random walk betweenness of all nodes. Note that exactly computing random walk betweenness in a distributed manner might entail a node to collect all the other nodes' neighbors information whose time complexity will be $O(m)$ where m is the number of edges. Besides the linear time distributed algorithm, we also prove the first non-trivial $\Omega(\frac{n}{\log n}+D)$ lower bound for distributively computing random walk betweenness centralities of all nodes where D is the diameter of the graph.

Acknowledgement

This work was supported in part by the National Natural Science Foundation of China under Grant No. 61572216, 61433019, U1435217, the National Key Research and Development Plan (No. 2016YFB0200502), and the Fundamental Research Funds for the Central Universities, HUST:No. 0118210126, 0180210118.

References

[1] M. E. J. Newman. A measure of betweenness centrality based on random walks. *Social Networks*, 27(1):39–54, 2005.

[2] Stanley Wasserman and Katherine Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

[3] Dimitrios Prountzos and Keshav Pingali. Betweenness centrality: algorithms and implementations. In *ACM SIGPLAN Notices*, volume 48. ACM, 2013.

[4] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.

[5] Qiang-Sheng Hua, Haoqiang Fan, Ming Ai, Lixiang Qian, Yangyang Li, Xuanhua Shi, and Hai Jin. Nearly optimal distributed algorithm for computing betweenness centrality. In *36th IEEE International Conference on Distributed Computing Systems, ICDCS*, pages 271–280, 2016.

[6] Nicolas Kourtellis, Tharaka Alahakoon, Ramanuja Simha, Adriana Iamnitchi, and Rahul Tripathi. Identifying high betweenness centrality nodes in large social networks. *Social Netw. Analys. Mining*, 3(4):899–914, 2013.

[7] Karen Stephenson and Marvin Zelen. Rethinking centrality: Methods and examples. *Social Networks*, 11(1):1–37, 1989.

[8] Linton C Freeman, Stephen P Borgatti, and Douglas R White. Centrality in valued graphs: A measure of betweenness based on network flow. *Social networks*, 13(2):141–154, 1991.

[9] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows - theory, algorithms and applications*. Prentice Hall, 1993.

[10] Mohsen Ghaffari, Andreas Karrenbauer, Fabian Kuhn, Christoph Lenzen, and Boaz Patt-Shamir. Near-optimal distributed maximum flow: Extended abstract. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC*, pages 81–90, 2015.

[11] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[12] Konstantin Avrachenkov, Nelly Litvak, Danil Nemirovsky, and Natalia Osipova. Monte carlo methods in pagerank computation: When one iteration is sufficient. *SIAM Journal on Numerical Analysis*, 45(2):890–904, 2007.

[13] Atish Das Sarma, Anisur Rahaman Molla, Gopal Pandurangan, and Eli Upfal. Fast distributed pagerank computation. *Theor. Comput. Sci.*, 561:113–121, 2015.

[14] Konstantin Avrachenkov, Nelly Litvak, Vasily Medyanikov, and Marina Sokol. Alpha current flow betweenness centrality. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 106–117. Springer, 2013.

[15] Atish Das Sarma, Danupon Nanongkai, Gopal Pandurangan, and Prasad Tetali. Efficient distributed random walks with applications. In *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC*, pages 201–210, 2010.

[16] Danupon Nanongkai, Atish Das Sarma, and Gopal Pandurangan. A tight unconditional lower bound on distributed randomwalk computation. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC*, pages 257–266, 2011.

[17] David Peleg. Distributed computing a locality sensitive approach. *SIAM Monographs on discrete mathematics and applications*, 5, 2000.

[18] Michael Mitzenmacher and Eli Upfal. *Probability and computing - randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[19] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, New York, NY, USA, 1997.

[20] Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. Networks cannot compute their diameter in sublinear time. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1150–1162, 2012.

[21] Mert Saglam and Gábor Tardos. On the communication complexity of sparse set disjointness and exists-equal problems. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 678–687, 2013.