Nearly Optimal Distributed Algorithm For Computing Betweenness Centrality

Qiang-Sheng Hua*, Haoqiang Fan†, Ming Ai*, Lixiang Qian*, Yangyang Li*, Xuanhua Shi*, Hai Jin*
* *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.*
† *Institute for Interdisciplinary Information Science, Tsinghua University, P.R. China*

***Abstract*—In this paper, we propose an $O(N)$ time distributed algorithm for computing betweenness centralities of all nodes in the network where N is the number of nodes. Our distributed algorithm is designed under the widely employed $\mathcal{CONGEST}$ model in the distributed computing community which limits each message only contains $O(\log N)$ bits. To our best knowledge, this is the first linear time deterministic distributed algorithm for computing the betweenness centralities in the published literature. We also give a lower bound for distributively computing the betweenness centrality under the $\mathcal{CONGEST}$ model as $\Omega(D+N/\log N)$ where D is the diameter of the network. This implies that our distributed algorithm is nearly optimal.**

I. Introduction

Centrality is an important measure used in complex network analysis to quantify the relative importance of a node within the network [1], [2]. Various centrality indices have been proposed in the literature [3], including graph centrality, closeness centrality, stress centrality and betweenness centrality. All these centrality measures are closely related with the shortest paths in the network. Roughly speaking, the node's graph centrality is the inverse of its longest distance to the other nodes; the node's closeness centrality is the inverse of the sum of its distances to all the other nodes; the node's stress centrality denotes the number of shortest paths passing through this node; the node's betweenness centrality is the ratio of the number of shortest paths passing through the node over the total number of shortest paths.

Given an undirected and unweighted graph $G = (V, E)$ where V ($|V| = N$) denotes the set of nodes and E ($|E| = M$) denotes the set of edges, we use $d(v, u)$ to denote the length of the shortest path between $v \in V$ and $u \in V$. In addition, we use σ_{st} to denote the number of shortest paths between s and t, and $\sigma_{st}(v)$ to denote the number of shortest paths between s and t passing node v. Then the formal definitions of the above centrality indices are given below.
The *closeness centrality* of v:

$$C_C(v) = \frac{1}{\sum_{t\in V} d(v,t)}. \tag{1}$$

The *graph centrality* of v:

$$C_G(v) = \frac{1}{\max_{t\in V} d(v,t)}. \tag{2}$$

The *stress centrality* of v:

$$C_S(v) = \sum_{s\neq t\neq v} \sigma_{st}(v). \tag{3}$$

The *betweenness centrality (BC)* of v[1]:

$$C_B(v) = \sum_{s\neq t\neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}. \tag{4}$$

Motivated by the fast-growing need for computing these centrality indices on large-scale graphs, our task is to compute them with a low time complexity. For a node v, computing the graph centrality $C_G(v)$ or the closeness centrality $C_C(v)$ would be much easier than computing the stress and betweenness centralities since the former two can be computed by solving the single source shortest path (*SSSP*) problem which takes $O(M + N)$ time[2]. However, computing the stress centrality or the betweenness centrality for a node v is much more complicated. The state-of-the-art Brandes algorithm [3] can compute *BC* in $O(MN)$ time[3]. Even worse, [5] proved that computing *BC* needs $\Omega(MN)$ time by using the path comparison based methods.

Turning to the distributed algorithm side, very recent results [6], [7], [8] show that *APSP* can be computed in $O(N)$ time[4] meaning the graph or the closeness centralities can be also computed in linear time. The remaining question is, ***can we also design a linear time distributed algorithm for betweenness centrality?*** In this paper, we give an affirmative answer. Moreover, our distributed algorithm is based on the extensively used $\mathcal{CONGEST}$ model which limits each sent message only contains $O(\log N)$ bits.

The remainder of this paper is organized as follows. We give the related work in Section II. The system model and problem definition are explained in Section III. We discuss the details of the Brandes algorithm in Section IV. The challenges for implementing a distributed version of the Brandes algorithm under the $\mathcal{CONGEST}$ model are discussed in Section V. In Section VI, we will show how to

[1]We did not consider the random-walk based betweenness centrality proposed in [4]. Distributively computing this centrality will be our future work.
[2]The time means the number of centralized steps.
[3]The stress centrality can also be computed in a similar way.
[4]The time means the number of rounds, c.f. subsection III-A.

1063-6927/16 $31.00 © 2016 IEEE
DOI 10.1109/ICDCS.2016.89

employ the floating point arithmetic in computing *BC* and how to bound the accumulated errors. Then our linear time distributed algorithms and their analyses are introduced in Section VII and Section VIII, respectively. The lower bound for distributively computing the betweenness centrality is given in Section IX. We conclude the paper and give some future works in Section X.

II. Related Work

As already mentioned, the fastest centralized algorithm for computing *BC* is the Brandes algorithm [3]. The key insight of this algorithm is it finds a recursive formulation of *BC* making it possible to compute *BC* by solving $O(N)$ times single source short paths (*SSSP*). This algorithm needs $O(N+M)$ space and runs in $O(N+M)$ time on unweighted graphs and $O(NM + N^2 \log N)$ time on weighted graphs. Comparing with the previous $O(N^3)$ time [9] algorithm on unweigheted graphs, it achieves a big improvement on the sparse graphs where $m = O(N)$. In [5], the authors prove computing *BC* needs $\Omega(MN)$ time by any path comparison based methods. The authors in [10] build a connection between the *APSP*, the *Diameter* and the *BC* problems, showing that if either problem can be computed in $O(N^{3-\alpha})$ (α is a small constant) time, then the other two can also be computed in $O(N^{3-\beta})$ (β is a small constant) time.

Based on the Brandes algorithm, many algorithms have been devoted to the approximation algorithms. In [11], [12], the *BC* values are approximated by extrapolating from the random subset instead of computing the contribution of all the other vertices. It needs $\Omega(\log N(N/\delta)/\epsilon^2)$ samples to guarantee that all estimates are within ϵ from the real value with probability at least $1-\delta$. In [13], an adaptive sampling algorithm is proposed to approximate the high-*BC* nodes, it performs an *SSSP* computation on all other vertices.

Since the *SSSP* and *APSP* can be used to compute centrality indices, we need to briefly review typical distributed algorithms for them. First, the distributed Breadth-First-Search(*BFS*) algorithm [14] on the unweighted graphs can be used to solve *SSSP*. This only needs $O(D)$ time where D is the network diameter. For distributively computing *APSP*, a series of recent research [6], [7], [8] show that *APSP* can be solved under the $\mathcal{CONGEST}$ model in time $O(N)$. In the PhD Thesis [15], Holzer also briefly discussed how to design distributed algorithms to approximate closeness and betweenness centralities. Their algorithms are based on sampling and the result is given without algorithm details. For weighted graphs, distributed *BFS* algorithm does not apply. Instead, Nanongkai [16] proposed an elegant randomized method to compute $(1+\epsilon)$-approximation APSP in time $O(\epsilon^{-2} N \log^2 N)$. In [17], the authors proposed a deterministic $(1+\epsilon)$-approximation APSP in time $O(\epsilon^{-2} N \log N)$, which imporved the result in [16] with a factor of $\Theta(\log N)$.

III. System model and problem definition

A. System Model

In this paper, we will use the classical distributed computing model that has been widely employed in the distributed computing community [14]. We model the network as an undirected graph $G = (V, E)$, where V and E represent the nodes and edges, respectively. Each node is represented by a unique identifier (ID) with $O(\log N)$ bits. The node $v \in V$ can only communicate directly with its neighbor $u \in V$ where $\{u, v\} \in E$. If $v \in V$ requires to communicate with $u \in V$ where $\{u, v\} \notin E$, the message must be passed along the nodes of some path from $v \in V$ to $u \in V$.

As for the communication message, we employ the frequently used $\mathcal{CONGEST}$ model. This model limits that the message sent from each node to its neighbors is bounded in $O(\log N)$ bits. Compared with the $\mathcal{LOCAL}$ model that the message size could be unbound, taking the $\mathcal{CONGEST}$ model poses a big challenge for designing efficient distributed algorithms. For example, in $\mathcal{LOCAL}$ model, most of distributed problems can be solved locally because each node can collect all the graph's topology in time D. However, in $\mathcal{CONGEST}$ model, each node $v \in V$ cannot obtain all the $\Gamma_k(v)$ (v's k-hop neighbors) in k rounds. So a problem which can be solved easily in $\mathcal{LOCAL}$ model may not be solved easily in $\mathcal{CONGEST}$ model.

In our system, we use synchronous communications where all nodes wake up simultaneously and start the distributed algorithm on their own. The communication occurs on the discrete and globally synchronized pulses. The time between two successsive pulses is called a *round*. In each round, each node is allowed to send a message to its neighbors. We further assume the communication channel is reliable.

We mainly use the *time complexity* to measure the efficiency of the distributed algorithms. An algorithm's *time complexity* is the number of the communication rounds until all nodes terminate. The *time complexity* has no restriction of local computation, every node can perform local computation in each round and it has no influence on the *time complexity* of distributed computing.

B. Problem Definition

After giving the system model, our research problem is to design a distributed algorithm for computing the betweenness centralities of all nodes in the undirected and unweighted graph under the $\mathcal{CONGEST}$ model.

IV. The Centralized Brandes Algorithm

Since our distributed algorithm can be seen as a distributed version of the centralized Brandes algorithm, we need to first give the algorithm details. Some notations used in this algorithm and our distributed algorithms are listed in Table I.

We first define the set of predecessors of u on the shortest paths from s:

$$P_s(u) = \{w|(w,u) \in E, d(s,u) = d(s,w) + d(w,u)\} \quad (5)$$

where $d(s,u)$ is the length of shortest path from node u to node s in G. Then the number of shortest paths from s to t can be calculated as:

$$\sigma_{st} = \sum_{w \in P_s(t)} \sigma_{sw}. \quad (6)$$

Pair dependency of a pair $s,t \in V$ on $v \in V$ is defined as:

$$\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}}. \quad (7)$$

Brandes defines the *dependency* of $s \in V$ on $v \in V$ as:

$$\delta_{s\cdot}(v) = \sum_{t \in V} \delta_{st}(v). \quad (8)$$

In [3], a recursive formulation of $\delta_{s\cdot}(v)$ is found as:

$$\delta_{s\cdot}(v) = \sum_{w: v \in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot (1 + \delta_{s\cdot}(w)). \quad (9)$$

We can compute the *BC* by Eq.(4) and Eq.(8) as:

$$C_B(v) = \sum_{s \neq v \in V} \delta_{s\cdot}(v) \quad (10)$$

By Eq.(6), we can compute the number of shortest paths from node $s \in V$ to $v \in V$ using the Breadth-First-Search (abbreviated as *BFS*) algorithm on node $s \in V$. To compute a source node s's dependency $\delta_{s\cdot}(v)$ on node $v \in V$, we can traverse the nodes in a non-increasing order of distances from $s \in V$, then we calculate the dependency recursively by Eq.(9). Finally, we can calculate the betweenness centralities by Eq.(10).

The pseudo codes of the Brandes algorithm are shown in $Algorithm$ 1. Firstly, the algorithm performs Breadth-First-Search on each node s (lines 1-19). When a node w is visited, the algorithm computes its distance $d(s,w)$ to s (lines 9-12), counts the number of shortest paths σ_{sw} (lines 14-15), and then records its predecessors with respect to source node s (line 16). Secondly, for each node v, the algorithm computes its dependency $\delta_{s\cdot}(v)$ by Eq.(9) (lines 22-24). Finally, the $C_B(v)$ of v can be computed by Eq.(10) (lines 26-28).

In summary, we can see that the Brandes algorithm computes the betweenness centralities of all nodes by two steps:

Step 1: Counting the number of all pairs shortest paths (lines 1-18).

Step 2: Calculating the dependency of each node using Eq.(9), then accumulating the dependencies to compute the betweenness centralities (lines 19-29).

Table I
NOTATIONS AND THEIR DEFINITIONS

Notation	Definition
$BFS(v)$	The BFS tree rooted in node v.
$d(v,u)$	The distance between nodes v and u.
σ_{st}	The number of shortest paths from s to t.
$\sigma_{st}(v)$	The number of shortest paths from s to t passing v.
$\delta_{st}(v)$	The pair dependency of pair (s,t) on node v.
$\delta_{s\cdot}(v)$	The dependency of a node s on the node v.
$\psi_s(v)$	The ratio of the $\delta_{s\cdot}(v)$ over σ_{sv}.
$P_s(v)$	The predecessors of v on the shortest paths from s.
$R_s(v)$	All descendants of v on shortest paths from source s.
T_0	The global clock used for synchronous execution.
$\Gamma_k(v)$	Node v's neighbors where $u \in \Gamma_k(v)$ has $d(v,u) \leq k$.

Algorithm 1 Brandes algorithm [3]

```
Input: Graph $G = (V,E)$
Output: $C_B(v)$ for each $v \in V$
 1: for $s \in V$ do
 2:     initialize $S = \emptyset; Q = \emptyset; \sigma_{ss} = 1; d(s,s) = 0$
 3:     initialize $P_s(w) = \emptyset$, $w \in V$
 4:     initialize $\sigma_{st} = 0$; $d(s,t) = -1$, $t \neq s \in V$
 5:     enqueue $s \to Q$
 6:     while $Q$ is not empty do
 7:         dequeue $v \leftarrow Q$
 8:         push $v \to S$
 9:         for $v$'s neighbor $w$ do
10:             if $d(s,w) < 0$ then
11:                 enqueue $w \to Q$
12:                 $d(s,w) \leftarrow d(s,v) + 1$
13:             end if
14:             if $d(s,w) = d(s,v) + 1$ then
15:                 $\sigma_{sw} \leftarrow \sigma_{sw} + \sigma_{sv}$
16:                 append $v \to P_s(w)$
17:             end if
18:         end for
19:     end while
20:     $\delta_{s\cdot}(v) \leftarrow 0, v \in V$
21:     while $S$ is not empty do
22:         pop $w \leftarrow S$
23:         for $v \in P_s(w)$ do
24:             $\delta_{s\cdot}(v) \leftarrow \delta_{s\cdot}(v) + \frac{\sigma_{sv}}{\sigma_{sw}} \cdot (1 + \delta_{s\cdot}(w))$
25:         end for
26:         if $v \neq s$ then
27:             $C_B(v) \leftarrow C_B(v) + \delta_{s\cdot}(v)$
28:         end if
29:     end while
30: end for
```

V. Challenges For Distributively Implementing The Brandes Algorithm

In this section, we will explain the two challenges for implementing a distributed version of the Brandes algorithm under the harsh $\mathcal{CONGEST}$ model.

As shown above, the Brandes algorithm has two steps. **Step 1** can be done by building N *BFS* trees for each node, we can implement it easily by the algorithm in [6] on $\mathcal{CONGEST}$ model. In **Step 2**, for calculating the dependency $\delta_{s\cdot}(v)$ of s on v, we need to compute the dependency of s on w where w has the longest distance from s at first, then w sends the dependency $\delta_{s\cdot}(w)$ to u where $u \in P_s(w)$. Third, we compute $\delta_{s\cdot}(u)$ by Eq.(9). These calculations will be run repeatedly until we have all the values for computing $\delta_{s\cdot}(v)$. In Brandes algorithm, in order to calculate $C_B(v)$, we need to calculate the dependencies of all nodes on v except the dependency of v on v. In this case, it may happen that a node w needs to send both $\delta_{s_1\cdot}(w)$ and $\delta_{s_2\cdot}(w)$ to a node v belonging to $P_{s_1}(w)$ and $P_{s_2}(w)$ at the same time. Even worse, if the node w needs to send $O(N)$ messages in the same time to a node, this violates the message size limit of the $\mathcal{CONGEST}$ model since each message contains $O(\log N)$ bits. So **Step 2** in the Brandes algorithm is hard to implement in a distributed way under the stringent $\mathcal{CONGEST}$ model. Since the key issue in this step is to aggregate the dependency values for all nodes, we call this challenge as the "Aggregation Challenge". Note that, although the efficient algorithm in [6] can be used to fulfill the task of **Step 1**, it cannot be easily utilized to overcome the "Aggregation Challenge" (**Step 2** of the Brandes algorithm), i.e., just reversing the *BFS* construction in [6] cannot solve **Step 2** easily. For example, in Figure 1(d), node v_5 is a leaf in $BFS(v_4)$. If we use convergecast in $BFS(v_4)$, v_5 will send a message to its parent v_4 at first. But for counting $\delta_{v_4\cdot}(v_5)$, v_5 must wait until v_2 has sent a message to it. So algorithm in [6] cannot be easily used in **Step 2**.

On the other hand, since the $\mathcal{CONGEST}$ model requires that each message can only contain $O(\log N)$ bits, the values sent by each node must be $O(N)$. However, the number of shortest paths from s to t, i.e., σ_{st}, could be as large as $O((N/D)^D)$. We call this challenge as the "Large Value Challenge".

The main idea of our distributed algorithm is to first solve the "Large Value challenge" such that each message will contain only $O(\log N)$ bits, and then we will focus on solving the "Aggregation Challenge".

VI. Solution For the Large Value Challenge

As mentioned above, the number of shortest paths from s to t (σ_{st}) could be as large as $O((N/D)^D)$ which is exponential, we will use the floating point arithmetic to tackle this issue. Although it will incur some calculation errors, and these errors may also accumulate in computing the betweenness centralities, we will show these accumulated errors will be bounded to a small value (depending on the network size and diameter) in our distributed algorithm.

A. Floating Point Arithmetic

Since all these σ_{st} values are positive, we only consider positive values. Given a positive number a, we can represent it as $a = y2^x$, where $x \in \mathcal{Z}, y \in [0, 1]$. We use $2L$=$O(\log N)$ bits to store the values of x and y. Note that all the numbers represented by $2L$ bits is $\mathcal{A}$=$\left\{\frac{u}{2^L}2^v \mid -2^L+1 \le v \le 2^L-1, 0 \le u \le 2^L-1\right\}$.

Now we first consider the relative error introduced by storing the numbers in floating point format.

Lemma 1. *Given a postive number $b \in \mathcal{B} = (2^{-2^L+1+L}, 2^{2^L-1})$, we can get the b's ceil estimation value a such that the relative error between a and b can be controlled by*

$$|\frac{a}{b} - 1| \le 2^{-L+1}. \tag{11}$$

Proof: Write b as $b = y2^L2^{x-L}$, set the estimated value $\hat{y} \in [0, 2^L-1]$ as $\lceil y2^L \rceil$. Choose x such that $|y| > \frac{1}{2}$. Then the estimated value $a = \hat{y}2^{x-L}$, and we can get

$$|\frac{a}{b} - 1| = |\frac{\hat{y} - 2^L y}{y2^L}|. \tag{12}$$

Notice that $y2^L \ge 2^{L-1}$, and $|\hat{y} - 2^L y| \le 1$. So we have

$$|\frac{a}{b} - 1| = |\frac{\hat{y} - 2^L y}{y2^L}| \le \frac{1}{2^{L-1}}. \tag{13}$$

■

B. Bounding the Accumulated Errors

When we use the floating point number to approximate the exponential value such as the number of shortest paths, the accumulated errors incurred will be bounded.

First, in order to reduce error in the computation process, we define a new notation $\psi_s(v) = \delta_{s\cdot}(v)/\sigma_{sv}$. Then, we can rewrite the Eq.(9) as

$$\psi_s(v) = \sum_{w:v \in P_s(w)} (1/\sigma_{sw} + \psi_s(w)). \tag{14}$$

By using the above formula, the node will send value $\psi_s(v)$ instead of $\delta_{s\cdot}(v)$ used in the Brandes algorithm.

We define the set of all the descendants of v on shortest paths from source s as $R_s(v)$.

Lemma 2. *For each node v,*

$$\psi_s(v) = \begin{cases} 0 & R_s(v) = \emptyset \\ \sum_{q:q \in R_s(v)} 1/\sigma_{sq} & R_s(v) \ne \emptyset \end{cases}. \tag{15}$$

Proof: The proof is by induction on node v. When v does not have any descendant on s,

$$\psi_s(v) = 0.$$

When v has only one descendant q on s, by Eq.(14) we have:

$$\psi_s(v) = 1/\sigma_{sq}.$$

Now, assuming the inductive hypothesis $\psi_s(w) = \sum_{q:q\in R_s(w)} 1/\sigma_{sq}$, we compute the $\psi_s(v)$ by Eq.(14) where v is the predecessor of w on s:

$$\psi_s(v) = \sum_{w:v\in P_s(w)} (1/\sigma_{sw} + \sum_{q:q\in R_s(w)} 1/\sigma_{sq}).$$

Because node q is the descendant of w on s, and v is the predecessor of w, we have:

$$\psi_s(v) = \sum_{q:q\in R_s(v)} 1/\sigma_{sq}.$$

■

And the estimated value $\hat{\psi}_s(v)$ is

$$\hat{\psi}_s(v) = \sum_{w:w\in R_s(v)} 1/\hat{\sigma}_{sw}. \tag{16}$$

The set $R_s(v)$ is used for computing both $\hat{\psi}_s(v)$ and $\psi_s(v)$. We suppose $\psi_s(v) > 2^{-2^L+1+L}$. In addition, by Lemma 1, $\hat{\sigma}_{sv} > \sigma_{sv}$ since $\hat{\sigma}_{sv}$ is the ceil estimated value of σ_{sv}. So the value $\hat{\sigma}_{sv}$ satisfies $\sigma_{sv} < \hat{\sigma}_{sv} < (1+\eta)\sigma_{sv}$, where η is a relative error $O(2^{-L})$. Furthermore, we get inequality $\sigma_{sw} < \hat{\sigma}_{sw} < (1+\eta)\sigma_{sw}$, $w \in R_s(v)$. Then each estimated value $1/\hat{\sigma}_{sw}$ satisfies $\frac{1}{(1+\eta)\sigma_{sw}} < \frac{1}{\hat{\sigma}_{sw}} < \frac{1}{\sigma_{sw}}$, and we get

$$(\frac{1}{1+\eta}) \sum_{w:w\in R_s(v)} \frac{1}{\sigma_{sw}} < \sum_{w:w\in R_s(v)} \frac{1}{\hat{\sigma}_{sw}} < \sum_{w:w\in R_s(v)} \frac{1}{\sigma_{sw}}. \tag{17}$$

Consider Eq.(15) and Eq.(16), we get

$$(\frac{1}{1+\eta})\psi_s(v) < \hat{\psi}_s(v) < \psi_s(v) \tag{18}$$

where η is a relative error $O(2^{-L})$.

Before calculating the betweenness centrality of v, the value of $\delta_{s\cdot}(v)$ should be computed again because we just send $\psi_s(v)$. So node v needs to receive a message containing σ_{sw} and $\psi_s(w)$. After all messages have been received, Eq.(14) need to multiply σ_{sv}. As $\hat{\sigma}_{sv}$ is stored locally and it is an estimated value, it affects the final value by inequality $\sigma_{sv} < \hat{\sigma}_{sv} < (1+\eta)\sigma_{sv}$, then we get another inequality.

$$\begin{aligned}(\frac{1}{1+\eta})\sigma_{sv} \sum_{w:w\in R_s(v)} \frac{1}{\sigma_{sw}} &< \hat{\sigma}_{sv} \sum_{w:w\in R_s(v)} \frac{1}{\hat{\sigma}_{sw}} \\ &< (1+\eta)\sigma_{sv} \sum_{w:w\in R_s(v)} \frac{1}{\sigma_{sw}}\end{aligned} \tag{19}$$

where η is a relative error $O(2^{-L})$.

Theorem 1. *By using the floating point arithmetic, the betweenness centrality $C_B(v)$ for each node v only has the relative error $O(\eta)$.*

Proof: Consider Eq.(18), Eq.(19) and $\psi_s(v) = \delta_{s\cdot}(v)/\sigma_{sv}$, then the estimated value $\hat{\delta}_{s\cdot}(v)$ satisfies

$$(\frac{1}{1+\eta})\delta_{s\cdot}(v) < \hat{\delta}_{s\cdot}(v) < (1+\eta)\delta_{s\cdot}(v). \tag{20}$$

Because $C_B(v) = \sum_{s\neq v\in V} \delta_{s\cdot}(v)$, so we can get inequality

$$(\frac{1}{1+\eta})C_B(v) < \hat{C}_B(v) < (1+\eta)C_B(v). \tag{21}$$

Then, the relative error of $C_B(v)$ is $O(\eta)$, where η is $O(2^{-L})$. ■

Corollary 1. *Set $L = O(\log N)$, then the betweenness centrality $C_B(v)$ for each node v only has relative error $O(N^{-c})$ (c is a constant number).*

Proof: We have proved that each node v has relative error $O(\eta)$, where η is $O(2^{-L})$. Since $L = O(\log N)$, the relative error is $O(N^{-c})$. ■

C. Messages Containing Exponential Values

In this subsection, we will further discuss when one node needs to send the message containing the exponential values to the other node in our distributed algorithms (c.f. the following section), how to make sure it only sends $O(\log N)$ bits and how to recover these exponential values when calculating the betweenness centrality.

As described above, we can store σ_{sv} and $\psi_s(v)$ in floating point format with a relative error $O(N^{-1})$ even if the values are $O((N/D)^D)$. These exponential values can be replaced by $a = y\cdot 2^x$, and we just send x and y with $2L$ bits where L is $O(\log N)$. When a node has received x and y, it needs to do a reverse operation. After all messages have been received by node v, we do one more operation $\psi_s(v) \times \sigma_{sv}$ since $\psi_s(v)$ is actually $\delta_{s\cdot}(v)/\sigma_{sv}$. Finally, we sum all $\delta_{s\cdot}(v)$ for each v belonging to different sources. In summary, the potential exponential values sent in each message, such as the number of shortest paths σ_{st} between s and t and the $\psi_s(t)$, are all denoted by floating point arithmetic with $2L$ ($L = O(\log N)$) bits.

VII. The Distributed Algorithm

To implement the Brandes algorithm on the $\mathcal{CONGEST}$ model, our distributed algorithm also has two corresponding phases: the counting phase and the aggregation phase. In the counting phase, we mainly borrow the algorithm from [6] to implement **Step 1** in the Brandes algorithm. In the aggregation phase, we implement **Step 2** in the Brandes algorithm such that each node can efficiently aggregate all the necessary values in Eq. (9) and Eq. (14) to compute the betweenness centralities for all nodes.

We describe counting phase as $Algorithm$ 2. As mentioned, the main idea of the algorithm are largely borrowed from existing work. Concretely, $Algorithm$ 2 is modified from the efficient distributed algorithm for computing *APSP* in [6]. We modify these algorithms by further computing parameters like the number of shortest paths between each pair s and t (σ_{st}) and the predecessors for each node v on the shortest paths from s ($P_s(v)$).

We build a BFS tree in the graph rooted in a randomly selected vertex at first. In $Algorithm$ 2 we perform Depth-First-Search(*DFS*) in *BFS* tree, when one node is visited in BFS tree, it performs *BFS* to calculate the shortest paths from it. So after $Algorithm$ 2, we can get the all pairs shortest paths information. In Brandes algorithm, when a node v needs to aggregate data in $BFS(v)$, every node u sends its message in order by the distance $d(v,u)$, the longer the $d(v,u)$ is, the earlier the node u sends the message to nodes in $P_v(u)$. So we coordinate each node's sending time in the aggregation phase to avoid the message collisions. The aggregation phase is described in $Algorithm$ 3. We will prove the algorithm's correctness in the next section.

```
Algorithm 2 Counting
Input: $BFS(u_0)$
Output: Each node $v$ gets $L_v, D$
 1: Run a DFS process in $BFS(u_0)$ at $T_0$
 2: if a node $s$ is visited at first time then
 3:     $DFS$ waits one time slot
 4:     Start a $BFS(s)$
 5:     Get the $BFS(s)$ starting time $T_s$
 6: end if
 7: for each node $v$ in $BFS(s)$ do
 8:     $L_v \leftarrow \emptyset$
 9:     $v$ records $T_s, s$
10:     for $v$ receives messages from neighbor $u$ do
11:         if $d(s,v) < 0$ then
12:             $d(s,v) \leftarrow d(s,u) + 1$
13:         end if
14:         if $d(s,v) = d(s,u) + 1$ then
15:             $\sigma_{sv} \leftarrow \sigma_{sv} + \sigma_{su}$
16:             $P_s(v) \leftarrow P_s(v) \bigcup u$
17:         end if
18:         $v$ sends $d(s,v)$, $\sigma_{sv}$ to its neighbors
19:     end for
20:     $L_v \leftarrow L_v \bigcup (s, T_s, d(s,v), \sigma_{\text{sv}}, P_s(v))$
21: end for
22: Broadcast the diameter $D$ to all nodes
```

By performing $Algorithm$ 2, we obtain the diameter of graph, the lengths and numbers of all pairs shortest paths. $Algorithm$ 2 performs Depth-First-Search (DFS) in *BFS*(u_0) from root u_0 (line 1). When a node s is visited in $BFS(u_0)$ for the first time at T_s, DFS should wait a time slot, and node s records T_s and builds a *BFS*(s) (lines 2-6). Each node v in $BFS(s)$ records its source s, distance $d(v,s)$, the number of shortest paths σ_{vs}, time T_s and its predecessors $P_s(v)$ on shortest path from s (lines 7-16). Recording the maximum $d(v,s)$ $\forall v, s \in V$ as diameter D, and letting each node know D (line 22).

```
Algorithm 3 Computing Betweenness Centrality
Input: $D, L_u = (s, T_s, d(s,u), \sigma_{su}, P_s(u))$
Output: $C_B(u)$
 1: Resetting the global clock as $T_0$
 2: for each node $u$ do
 3:     Compute $T_s(u) \leftarrow T_s + D - d(s,u)$
 4:     Initialize $\psi_s(u) \leftarrow 0$
 5:     Initialize $C_B(u) \leftarrow 0$
 6: end for
 7: while $T_0 \leq$ maximum $(T_s) + D$ do
 8:     if $u$ receives $(1/\sigma_{sv} + \psi_s(v))$ then
 9:         $\psi_s(u) \leftarrow \psi_s(u) + (1/\sigma_{sv} + \psi_s(v))$
10:     end if
11:     if $T_0 = T_s(u)$ then
12:         $u$ sends $1/\sigma_{su} + \psi_s(u)$ to node $w \in P_s(u)$
13:     end if
14:     $T_0 \leftarrow T_0 + 1$
15: end while
16: for $u \in V \neq s$ do
17:     $\delta_{s\cdot}(u) = \psi_s(u) \cdot \sigma_{su}$
18:     $C_B(u) \leftarrow C_B(u) + \delta_{s\cdot}(u)$
19: end for
```

In $Algorithm$ 3, we compute the *BC* of each node. We reset the global time as T_0 equalling to the time DFS in $Algorithm$ 2 starts to avoid the time conflict with the DFS process (line 1). Each node u computes its sending time to its predecessors in $P_s(u)$ as $T_s(u) = T_s + D - d(s,u)$ to avoid message collisions (two messages sent on the same edge), and initialize the $\psi_s(u) = 0$, $C_B(u) = 0$ (lines 2-6). When T_0 equals $T_s(u)$, node u will send the value of $1/\sigma_{su} + \psi_s(u)$ to its predecessor $w \in P_s(u)$ (lines 11-12). When a node u receives $1/\sigma_{su} + \psi_s(u)$ from its neighbor v, it calculates the $\psi_s(v)$ by Eq. (14) (lines 8-9). Each node u (excluding node s) computes its dependency $\delta_{s\cdot}(u) = \psi_s(u) \cdot \sigma_{su}$, and its *BC* $C_B(u)$=$C_B(u) + \delta_{s\cdot}(u)$ (lines 16-18), these calculations are done locally on each node.

Since coordinating each node's message sending time in $Algorithm$ 3 plays an important role for avoiding message collisions, we will give the calculation details based on the graph in Figure 1. For example, in Figure 1(a), we first reset the global time because we run $Algorithm$ 3 after all BFS trees have been built. To calculate the *BC*, the dependency of v_1 on $\forall v \neq v_1$ is needed. In order to calculate the dependency of v_1 on $\forall v \neq v_1$, v_4 needs to send its message to v_3, v_5 at first because v_4 is farthest from v_1. So we calculate the message sending time of v_4 as $T_{v_1}(v_4) = T_{v_1} + D - d(v_1, v_4) = 0 + 3 - 3 = 0$ (line 3) where $T_{v_1}, d(v_1, v_4)$ are received from $Algorithm$ 2 (line 5). We can calculate the other nodes' sending time in $BFS(v_1)$ listed in Figure 1(a). Similarly, we can calculate the sending time of v_4 in $BFS(v_2)$ as $T_{v_2}(v_4) = T_{v_2} + D - d(v_2, v_4) = 2 + 3 - 2 = 3$

in Figure 1(b). The sending time of v_4 in $BFS(v_3)$ is $T_{v_3}(v_4) = T_{v_3}+D-d(v_2, v_4) = 4+3-1 = 6$ in Figure 1(c). In Figure 1(d), v_4 is the root of $BFS(v_4)$, so it does not need to send in $BFS(v_4)$. In Figure 1(e), v_4's sending time in $BFS(v_5)$ is $T_{v_5}(v_4) = T_{v_5}+D-d(v_5, v_4) = 8+3-1 = 10$.

When all nodes receive the messages, they will calculate the dependencies and the *BC* on their own. Still taking the graph in Figure 1 as an example, the dependency of v_1 on v_2 is $\delta_{v_1\cdot}(v_2) = \sigma_{v_1v_2}\cdot(1/\sigma_{v_1v_3}+\psi_{v_1}(v_3)+1/\sigma_{v_1v_5}+\psi_{v_1}(v_5))$ by Eq. (14). Here $\psi_{v_1}(v_5) = 1/\sigma_{v_1v_4} + \psi_{v_1}(v_4) = \frac{1}{2}$, and $\psi_{v_1}(v_3) = 1/\sigma_{v_1v_4} + \psi_{v_1}(v_4) = \frac{1}{2}$. So $\delta_{v_1\cdot}(v_2) = 1\cdot(1+\frac{1}{2}+1+\frac{1}{2}) = 3$. Similarly, we can compute the dependencies of the other nodes on v_2 and will achieve the same $C_B(v_2)$ value as the centralized Brandes algorithm which is $C_B(v_2) = (\delta_{v_1\cdot}(v_2)+\delta_{v_3\cdot}(v_2)+\delta_{v_4\cdot}(v_2)+\delta_{v_5\cdot}(v_2))/2 = \frac{7}{2}$.

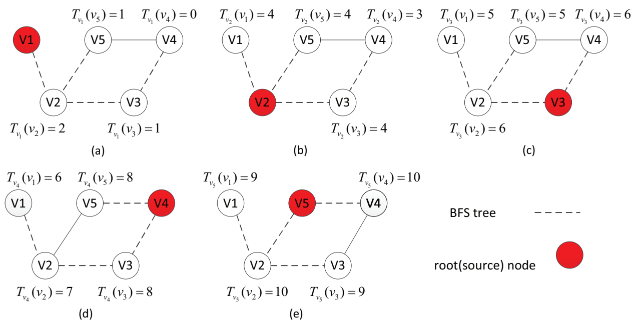

Figure 1. Calculating each node's sending time in each BFS tree. (a) Each node's sending time in $BFS(v_1)$. (b) Each node's sending time in $BFS(v_2)$. (c) each node's sending time in $BFS(v_3)$. (d) Each node's sending time in $BFS(v_4)$. (e) Each node's sending time in $BFS(v_5)$.

VIII. Algorithm Analysis

In this section, we will give both the correctness and efficiency analyses for our algorithms. In the correctness analysis, we will prove all of our algorithms satisfy the $\mathcal{CONGEST}$ model; whereas in the efficiency analysis, we will prove that our proposed distributed algorithms can compute the betweenness centralities of all nodes in $O(N)$ rounds where N is the number of nodes in the network.

A. Correctness Analysis

Lemma 3. *Algorithm 2 satisfies the* $\mathcal{CONGEST}$ *model.*

Proof: The main difference of $Algorithm$ 2 from the distributed *APSP* algorithm in [6] lies in lines 14-18. The algorithm in [6] has proved that it satisfies the $\mathcal{CONGEST}$ model. In addition, all the values sent in lines 14-18 can be packed into $O(\log N)$ bits. So $Algorithm$ 2 satisfies the $\mathcal{CONGEST}$ model. ■

Before we prove $Algorithm$ 3 also satisfies the $\mathcal{CONGEST}$ model, we first give the important Lemma 4.

Lemma 4. *In Algorithm 3, a node u in line 12 will never send more than one messages to its predecessor $w \in P_s(u)$ and $w \in P_t(u)$ simultaneously.*

Proof: Suppose a node s needs to aggregate all its $BFS(s)$'s data at T_s. Then the node u will send its own data to its predecessor in $P_s(u)$ at time $T_s(u) = T_s+D-d(s, u)$. Now, assuming a node t needs to aggregate all its $BFS(t)$'s data at $T_t > T_s$. In our algorithm it means that $T_t \geq T_s + d(s, t) + 1$, since it needs at least $d(s, t)$ time from s to t. So we can prove that $T_t(u) = T_t + D - d(u, t) \geq T_s + d(s, t) + 1 + D - d(u, t) \geq T_s + 1 + D - d(u, s) > T_s + D - d(u, s) = T_s(u)$ where the second to the last inequality is based on the triangle inequality. ■

In order to give an intuitive feeling of this proof, we also take the graph in Figure 1 as an example. If the dependencies of v_1 on $\forall v \neq v_1$ and the dependencies of v_2 on $\forall v \neq v_2$ are required to be computed at the same time, the v_4 will send $1/\sigma_{v_1v_4}+\psi_{v_1}(v_4)$ and $1/\sigma_{v_2v_4}+\psi_{v_2}(v_4)$ to its predecessors in $P_{v_1}(v_4) = \{v_3, v_5\}$ and $P_{v_2}(v_4) = \{v_3, v_5\}$ respectively at first (line 12), because v_4 is farthest from v_1 and v_2. In $Algorithm$ 3, we set $T_{v_1}(v_4) = T_{v_1} + D - d(v_1, v_4)$, and $T_{v_2}(v_4) = T_{v_2} + D - d(v_2, v_4)$ (line 3). Since $T_{v_2} - T_{v_1} = d(v_1, v_2)+1$ (lines 3-5 in $Algorithm$ 2), $T_{v_1}(v_4) < T_{v_2}(v_4)$. In conclusion, node v_4 cannot send messages to v_3 and v_5 simultaneously.

Lemma 5. *Algorithm 3 satisfies the* $\mathcal{CONGEST}$ *model.*

Proof: From Lemma 4, a node cannot send more than one messages to the same node at the same time. In addition, the value of $1/\sigma_{su} + \psi_s(u)$ (lines 11-12) can be described by $O(\log N)$ bits. So $Algorithm$ 3 satisfies the $\mathcal{CONGEST}$ model. ■

Theorem 2. *Our algorithms satisfy the* $\mathcal{CONGEST}$ *model.*

Proof: This is obvious from the above Lemma 3, and Lemma 5. ■

B. Efficiency Analysis

Lemma 6. *Algorithm 2 requires $O(N)$ time.*

Proof: In $Algorithm$ 2, line 1 requires $O(N)$ time. Lines 2-21 require $O(D)$ time. Line 22 requires $O(D)$ time. So $Algorithm$ 2 requires $O(N + D + D) = O(N)$ time. ■

Lemma 7. *Algorithm 3 requires $O(N)$ time.*

Proof: In $Algorithm$ 3, lines 1-6 require $O(1)$ time by computing locally. Lines 7-13 require $O(N)$ time. Because each node u's sending time $T_{su} = T_s + D - d(s, u)$ (line 3), we need to find the last sending time of all nodes. From Lemma 6, we have $T_s = O(N)$ in $Algorithm$ 2. Since $d(s, u) \geq 0$, we know the last sending time of all nodes is $O(N)$ (line 8). Lines 16-18 require $O(1)$ time by computing locally. So $Algorithm$ 3 requires $O(N)$ time.

■

Theorem 3. *Our algorithms compute betweenness centrality in $O(N)$ time.*

Proof: This is obvious from the above Lemma 6, Lemma 7. ■

IX. Lower Bound

In this section, we will first give a lower bound construction for distributively computing diameter. Based on this construction, we then give a lower bound construction for distributively computing betweenness centrality. Recall that we employ the $\mathcal{CONGEST}$ model where there are at most $O(\log N)$ bits traversing each edge in a communication round. Moreover, by taking a further look at the diameter lower bound construction in [18], we have analyzed their lower bound is $\Omega(\frac{N}{x \log N} + x)$ where x is the diameter of the graph. This lower bound decreases with the increase of x when $x \leq \sqrt{\frac{N}{\log N}}$. In our diameter lower bound construction, we will show that deciding whether the network's diameter is x or $x+2$ does not become easier as x increases.

A. Lower Bound of Diameter

In this section, we prove the lower bound of deciding whether the diameter of the network is x or $x+2$. We assume $x \geq 8$. Our construction is illustrated in Figure 2. Let m be an even integer to be determined later. We first create $2m$ nodes $L_{1..m}$[5] and $L'_{1..m}$, and add paths of length $x-6$ between corresponding node pairs. Then we choose n subsets $X_{1..n}$ (denote X as the set of subsets) of $M = \{1, 2, \ldots, m\}$ where the cardinality of each subset X_j is $m/2$. For each subset X_j, we create a node S_j and add an edge (L_i, S_j) for all $i \in X_j$. For each node S_j, we add two more nodes S'_j and S''_j such that they are linked to S_j to form a chain of length 2.

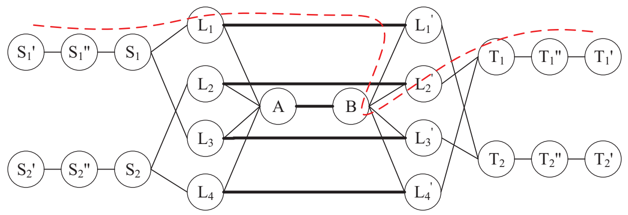

Figure 2. Construction for lower bound of computing diameter. The thick lines represent paths of length $x-6$. We set $m = 4$ and $n = 2$ such that $\binom{m}{m/2} \geq n^2$. The diameter of the graph is $x+2$ or x depending on whether there are X_i and Y_j that correctly match with each other. For example, in this figure, $X_1 = Y_1$, $X_2 = Y_2$, so S_1 and T_1 cannot reach each other without going backwards. The path marked by dash line is one of the shortest paths from S'_1 to T'_1 with $d(S'_1, T'_1) = x+2$.

On the right side of Figure 2, we choose another family of subsets $Y_{1..n}$ such that each subset has cardinality $m/2$. Then we create nodes T_j, T'_j and T''_j in the same way. The difference is that an edge (L'_i, T_j) is created for all $i \notin Y_j$ (this construction is different from the construction of left side, where an edge (L_i, S_j) exists for all $i \in X_j$). In the middle of Figure 2, two points A, B are added and connected to $L_{1..m}$ and $L'_{1..m}$ respectively. A path of length $x-6$ is also created between nodes A and B. Then we get Lemma 8.

Lemma 8. *The distance between S'_i and T'_j satisfies*

$$d(S'_i, T'_j) = \begin{cases} x & X_i \neq Y_j \\ x+2 & X_i = Y_j \end{cases} \tag{22}$$

and the diameter of the graph can be verified as

$$D = \begin{cases} x & X \cap Y = \emptyset \\ x+2 & \textit{otherwise} \end{cases}. \tag{23}$$

Proof: Let the eccentricity of a node be its maximum distance to other nodes $\text{ecc}(v) = \max_u d(u, v)$.

First consider node A. Its distance to B is $d(A, B) = x-6$. On the left side, we have $d(A, L_i) = 1$, $d(A, S_i) = 2$, $d(A, S''_i) = 3$, $d(A, S'_i) = 4$. Similar arguments apply to B. We conclude that $\text{ecc}(A) = \text{ecc}(B) = x-6+4 = x-2$.

For $v \in L \cup L' \cup S \cup T$, their eccentricity is bounded by $\text{ecc}(v) \leq \min\{\text{ecc}(A) + d(A, v), \text{ecc}(B) + d(B, v)\} \leq x-2+2 = x$.

Now consider S'_i and T'_j. As S_i and T_j represent X_i and Y_j, so when $X_i \neq Y_j$, the shortest path from S'_i to T'_j is $S'_i \to S''_i \to S_i \to L_p \to L'_p \to T_j \to T''_j \to T'_j$ where $p \in X_i \cap (M - Y_j)$ and the length is x. If $X_i = Y_j$, we cannot travel from S_i to T_j using only left-to-right edges. So the distance is at least $x+2$. This is witnessed by the path $S'_i \to S''_i \to S_i \to L_p \to A \to B \to L'_q \to T_j \to T''_j \to T'_j$ where $p \in X_i$, $q \in M - Y_j$.

As for the distances between nodes in $S' \cup S''$, the distance cannot exceed 8. And $\text{ecc}(S''_i) = \text{ecc}(S'_i) - 1$. The same argument applies to the right side.

Notice that X and Y are sets of X_i and Y_j, so we conclude that

$$D = \max_{i,j} d(S'_i, T'_j) = \begin{cases} x & X \cap Y = \emptyset \\ x+2 & X \cap Y \neq \emptyset \end{cases}.$$

■

In order to satisfy the sparse set disjointness model[19], we set $m = O(\log n)$ so that $\binom{m}{m/2} \geq n^2$. It follows that $N = O(n)$ and $D \in \{x, x+2\}$. Our bound is based on communication complexity theory [20].

Definition 1 (Communication Complexity)**.** *Let X, Y, Z be arbitrary finite sets and let $f : X \times Y \to Z$. There are two players, Alice and Bob, who wish to evaluate $f(x, y)$, for some inputs $x \in X$ and $y \in Y$. Hence, they need to communicate with each other according to some fixed protocal P. The communication complexity of f is the minimum cost of P, over all protocols P that computer*

[5] $L_{1..m}$ represents nodes $L_1, L_2, \cdots, L_m$, so do the subsequent $L'_{1..m}$, $F_{1..n}$, $T_{1..n}$, and $S_{1..n}$ notations.

f. We use $D(f)$ to denote the deterministic communication complexity.

Definition 2 (Sparse Set Disjointness)**.** *Two sets x and y, each of which is a subset of $[1, ..., n]$ of size k ($0 \leq k \leq n/2$). Function $DISJ_n^k(x, y)$ is defined to be 1 iff $x \cap y = \emptyset$.*

Theorem 4. *If two parties each having n numbers in range $\{1, 2, \ldots, n^2\}$ want to deterministically decide whether their sets are disjoint, at least $\Omega(n \log n)$ bits need to be exchanged. This lower bound also holds for randomized algorithms.*

Proof: From Definition 2, we can derive a sparse set disjointness problem $DISJ_{n^2}^n(x, y)$, where x, y are subsets of $[1, 2, ..., n^2]$ of size n respectively. Then we get $D(DISJ_{n^2}^n) = \log \binom{n^2}{n}$ [20], and [19] shows that the lower bound on the randomized complexity of this problem is $\Omega(n \log n)$. So, $\Omega(n \log n)$ bits need to be exchanged to decide whether x and y are disjoint. ■

Corollary 2. *Two sets X and Y, each of them has n subset X_i or Y_i which is a subset of $1, 2, ..., n^2$ of size $m/2$, where $m = \log n$ in our model. The $X \cap Y \neq \emptyset$ if there is a X_i equal to Y_j, $i, j \in [1, n]$. Then deciding whether X and Y are disjoint needs $\Omega(n \log n)$ bits even for randomized algorithms.*

Proof: We divide the n^2 numbers into n groups and each group is a subset of size $m/2$. Subset X_i and Y_i can be encoded as a number by lexicographical order of elements. For example, in Figure 2, The subset X_1 can be encoded as 01010 that each bit represents an element of subset. So we get two sets X and Y, each of them has n numbers. Then we can use Theorem 4 to get a low bound of $\Omega(n \log n)$. ■

Theorem 5. *Deciding whether the diameter of the network is x or $x + 2$, for $x \geq 8$ and $x = O(N/\log N)$ takes at least $\Omega(D + N/\log N)$ time. This bound does not decrease as x increases.*

Proof: If there is a distributed protocol to compute network diameter, the two parties can simulate the protocol on the network to solve the sparse set disjointness problem. In Figure 2, there must be at least $\Omega(N \log N)$ bits flowing from the left side to the right side. This takes $\Omega(N/\log N)$ time because only $m+1 = O(\log N)$ messages can be sent between the two sides and each message can carry $\log N$ bits. So at least $\Omega(D + N/\log N)$ time is needed. In order for the construction to work, the minimum diameter x need to satisfy all the conditions. Consider there is a constant distance between nodes in $S' \cup S''$, so we have to guarantee that $x \geq 8$. And there are $x \log n$ nodes between two sides, $O(n)$ nodes of two sides. The sum of them equal to N. As we need to guarantee that $N = O(n)$, hence $x \log n \leq n$ and we get $x = O(N/\log N)$. ■

B. Lower Bound of Betweenness Centrality

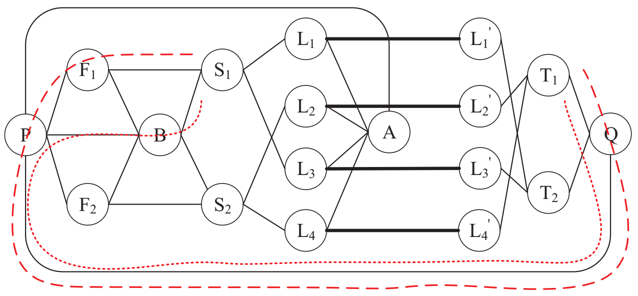

Figure 3. Construction for betweenness centrality. This is similar to the construction in Figure 2. S_i goes through F_i on its way to T_j if and only if it cannot go directly through $L_{1..m}$ and $L'_{1..m}$. A, B, P, Q are placed so that $C_B(F_i)$ is either 1 or 1.5. In this figure, we only consider the pairs of nodes that have the shortest paths passing node F_1. There are two shortest paths from S_1 to T_1, one is marked by dash line and another is marked by dotted line. Similarly, we can find two shortest paths from S_1 to P and two shortest paths from S_1 to Q. So $C_B(F_1)$ is 1.5.

The construction for betweenness centrality is more complicated. As depicted in Figure 3, $L_{1..m}$,$L'_{1..m}$,$S_{1..n}$,$T_{1..n}$ are constructed in the same way as previous construction except that the distance between L_i and L'_i is one. Nodes $F_{1..n}$ are added and connected to $S_{1..n}$. correspondingly. We also add four more nodes P,Q,A,B such that they are connected to $F_{1..n}$, $T_{1..n}$, $L_{1..m}$ and $S_{1..n}$ respectively. Then we have Lemma 9.

Lemma 9. *The betweenness centrality of F_i can be verified as*

$$C_B(F_i) = \begin{cases} 1 & X_i \notin X \cap Y \\ 1.5 & \textit{otherwise} \end{cases}. \tag{24}$$

Proof: The analysis is a bit more complicated. It is easy to see that nodes in $L \cup L' \cup T \cup \{A, B\}$ do not need to go through F to reach to each other. As for other nodes, the only chance that a shortest path contains F_i is that one of the endpoint is S_i.

$$C_B(F_i) = \delta_{S_i P}(F_i) + \delta_{S_i Q}(F_i) + \sum_j \delta_{S_i T_j}(F_i).$$

Observe that

$$d(S_i, T_j) = \begin{cases} 3 & X_i \neq Y_j \\ 4 & X_i = Y_j \end{cases}.$$

On the path from S_i to T_j when $X_i = Y_j$, the node can go through $S_i \rightarrow F_i \rightarrow P \rightarrow Q \rightarrow T_j$ or $S_i \rightarrow B \rightarrow P \rightarrow Q \rightarrow T_j$. This implies

$$\delta_{S_i T_j}(F_i) = \begin{cases} 0 & X_i \neq Y_j \\ 0.5 & X_i = Y_j \end{cases}.$$

So we conclude that

$$C_B(F_i) = \begin{cases} 1 & X_i \notin X \cap Y \\ 1.5 & \text{otherwise} \end{cases}.$$

■

Theorem 6. *Computing betweenness centrality to* 0.499 *relative error needs at least* $\Omega(D + N/\log N)$ *time even for randomized algorithms.*

Proof: We use essentially the same communication complexity argument. Two parties wish to solve $\mathrm{DISJ}_n^{n^2}$. They construct the left and right side of the graph according to their sets and simulate the distributed betweenness centrality algorithm. Notice that computing to relative error 0.499 suffices to distinguish 1 from 1.5. So $\Omega(N \log N)$ bits need to flow across the cut. Each time only $O(\log N \log N)$ bits can be sent over the cut, so the time complexity is at least $\Omega(D + N/\log N)$. ■

X. Conclusion

In this paper, we have proposed a linear time ($O(N)$) distributed algorithm for computing the betweenness centralities for all nodes in the graph under the widely used $\mathcal{CONGEST}$ model. This algorithm can be seen as a distributed version of the prominent Brandes algorithm, although we need to overcome the two nontrivial challenges to make it work under the $\mathcal{CONGEST}$ model. To our best knowledge, this is the first linear time distributed algorithm that distributively compute the betweenness centralities of all nodes. In addition, we also give the first non-trivial lower bound for distributively computing betweenness centralities as $\Omega(D + N/\log N)$ where D is the diameter. Note that our distributed algorithm only works on the unweighted graph. For weighted graphs, there are no efficient distributed algorithms for computing betweenness centralities. But the idea in [16] which adds virtual nodes in the weighted edges might also work for approximately computing betweenness centralities. Finally, designing an $O(D + N/\log N)$ time distributed algorithm that matches the lower bound will be also an interesting future work.

Acknowledgment

We thank the reviewers for their valuable comments to improve the presentation this paper. This work was supported in part by the National Natural Science Foundation of China under Grant No. 61572216 and the National 863 Hi-Tech Research and Development Program under Grant No. 2014AA01A301.

References

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

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

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

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

[5] Shiva Kintali. Betweenness centrality: Algorithms and lower bounds. *arXiv preprint arXiv:0809.1906*, 2008.

[6] Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *Proc. PODC*, 2012.

[7] Christoph Lenzen and David Peleg. Efficient distributed source detection with limited bandwidth. In *Proc. PODC*, 2013.

[8] David Peleg, Liam Roditty, and Elad Tal. Distributed algorithms for network diameter and girth. In *Proc. ICALP*, 2012.

[9] Riko Jacob, Dirk Koschützki, Katharina Lehmann, Leon Peeters, and Dagmar Tenfelde-Podehl. Algorithms for centrality indices. *Network Analysis*, 2005.

[10] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, apsp and diameter. In *Proc. SODA*, 2015.

[11] Ulrik Brandes and Christian Pich. Centrality estimation in large networks. *International Journal of Bifurcation and Chaos*, 17(07):2303–2318, 2007.

[12] David Eppstein and Joseph Wang. Fast approximation of centrality. *J. Graph Algorithms Appl.*, 8:39–45, 2004.

[13] David A Bader, Shiva Kintali, Kamesh Madduri, and Milena Mihail. Approximating betweenness centrality. In *WAW*. Springer, 2007.

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

[15] Stephan Holzer. *Distance Computation, Information Dissemination, and Wireless Capacity in Networks, Diss. ETH No. 21444*. PhD thesis, ETH Zurich, Zurich, Switzerland, 2013.

[16] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proc. STOC*, 2014.

[17] Christoph Lenzen and Boaz Patt-Shamir. Fast partial distance estimation and applications. In *Proc. PODC*, 2015.

[18] Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. Networks cannot compute their diameter in sublinear time. In *Proc. SODA*, 2012.

[19] Mert Saglam and Gábor Tardos. On the communication complexity of sparse set disjointness and exists-equal problems. In *Proc. FOCS*, 2013.

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