SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

A Nearly Optimal Distributed Algorithm for Computing the Weighted Girth

Qiang-Sheng HUA[1*], Lixiang QIAN[1], Dongxiao YU[2], Xuanhua SHI[1] & Hai JIN[1]

[1]*National Engineering Research Center for Big Data Technology and System/Services Computing Technology and System Lab/ Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan* 430074*, P. R. China;*
[2]*School of Computer Science and Technology, Shandong University, Qingdao* 266237*, P. R. China*

Abstract Computing the weighted girth, which is the sum of weights of edges in the minimum weight cycle, is an important problem in network analysis. The problem for distributively computing girth in unweighted graphs has garnered lots of attention, but there are few studies in weighted graphs. In this paper, we propose a distributed randomized algorithm for computing the weighted girth in weighted graphs with integral edge weights in range $[1, n^c]$, where n is the number of vertices and c is a constant. The algorithm is devised under the standard synchronous $\mathcal{CONGEST}$ model, which limits each vertex can only transfer $O(\log n)$ bits information along each incident edge in a round. The upper bound of the algorithm is $O(n \log^2 n)$ rounds. We also prove the lower bound for computing the weighted girth is $\Omega(D + n/\log n)$ where D is the hop diameter of the weighted graph. This means our distributed algorithm is optimal within a factor of $O(\log^3 n)$.

Keywords distributed algorithms, weighted girth, CONGEST model, communication complexity, round complexity

Citation Hua Q-S, Qian L, Yu D, et al. A Nearly Optimal Distributed Algorithm for Computing the Weighted Girth. Sci China Inf Sci, for review

1 Introduction

Computing the weighted girth, which is the sum of weights of edges in the minimum weight cycle, is an important graph analysis problem in weighted graphs [1]. It has been shown that the weighted girth has a pivotal role in understanding the complexity of some other graph problems such as APSP (All-Pairs-Shortest-Paths), replacement paths [2] and matrix problems [3]. The girth problem was first presented by Itai and Rodeh [4] on unweighted graphs. They showed that the shortest cycle can be found by using a sequential Breadth-First-Search (BFS) for each vertex in $O(mn)$ time, or in $\tilde{O}(n^\omega)$ time 1) by using fast matrix multiplication, where m and n are the number of edges and vertices in the graph, respectively. For weighted graphs, it is much more difficult to get a similar result. A recent sequential algorithm presented in [6] computes the weighted girth in $\tilde{O}(Wn^\omega)$ time, where W is the maximum edge weight in the graph and $\tilde{O}$ suppresses a polylog$n$ factor. One of the authors in this paper, Virginia Vassilevska Williams, explained the lack of improvements of the weighted girth problem in [3]. The authors of [3] showed that the weighted girth problem has a close relationship with the APSP problem. If there is a truly subcubic sequential algorithm for the weighted girth, the APSP problem also has truly subcubic algorithms but many people conjectured that there is no truly subcubic algorithm for APSP [7]. To get a subcubic sequential algorithm for computing the weighted girth, many literatures aim to find efficient

* Corresponding author (email: qshua@hust.edu.cn)
1) ω is the exponent of square matrix multiplication over a ring and $\omega < 2.376$ [5].

approximation algorithms. Lingas and Lundell [8] presented a 2-approximation algorithm for computing weighted girth in undirected weighted graphs in $O(n^2 \log n(\log n + \log W))$ time. Furthermore, Roditty and Tov [9] reduced the approximation factor to 4/3 with the same running time as in [8].

In terms of the distributed scenario, we study the standard model of the distributed computation, called $\mathcal{CONGEST}$ model, which is a message passing model with limited bandwidth. More precisely, each vertex of the network can only send $O(\log n)$ bits of information along each incident edge in each round. In this model, some problems can be solved by local communication, but many problems need global communication such as computing diameter, shortest paths and girth problems. The global communication is very troublesome if there are many messages that need to be passed in the network. In this case, we have to take into account the congestion issue to get an efficient algorithm. Fortunately, many previous works have concerned $\mathcal{CONGEST}$ model and they presented many useful techniques for handling congestion in this model [10–13]. A fundamental technique is to distributively run n BFS processes in parallel to get distant information of vertices in the network [10] [13]. This technique is used by Holzer and Wattenhofer [10] to show that the girth can be computed in $O(n)$ rounds in unweighted graphs. A more efficient algorithm with running time of $O(n/\log n + D)$ rounds was proposed by Hua et al. in [14]. These two algorithms are based on computing distributed APSP in unweighted graphs. [10] and [15] showed that the lower bound of computing APSP in unweighted graphs is $\Omega(n/\log n + D)$. The close relationship of girth and APSP means we may not be able to get a better result by computing APSP. But, to our knowledge, there is no known lower bound for exactly computing girth under the $\mathcal{CONGEST}$ model. A recent lower bound for computing girth is presented by Frischknecht, Holzer and Wattenhofer [16], which proved that any $(2 - \epsilon)$ multiplicative approximation algorithm for computing girth needs $\tilde{\Omega}(\sqrt{n})$ rounds.

For the weighted graphs, a natural question is: can we design a distributed algorithm as fast as that on the unweighted graphs? As far as we know, there are no published distributed weighted girth algorithms. However, as mentioned above, we can distributively compute the weighted girth based on distributed APSP algorithms on weighted graphs. Although distributed APSP problem on unweighted graphs can be solved in $O(n/\log n + D)$ rounds, the state-of-the-art distributed randomized algorithm for computing weighted APSP under the $\mathcal{CONGEST}$ model needs $\tilde{O}(n^{5/4})$ rounds [17]. This means distributively computing the weighted girth via computing APSP also needs $\tilde{O}(n^{5/4})$ rounds. In this paper, we will propose a different method to distributively compute the weighted girth by circumventing the computation of APSP. This method provides a special perspective for handling congestion. Instead of computing all pairs shortest paths, our algorithm can always find a range to bound the weighted girth and narrow the range by the modified distributed BFS with a distance constraint. After a careful control of the range, the girth will be computed at the end of the algorithm with a detailed correctness proof. Our contributions are briefly summarized as follows:

(1) We propose a distributed randomized algorithm of $O(n \log^2 n)$ rounds for exactly computing the weighted girth in undirected weighted graphs under the $\mathcal{CONGEST}$ model.

(2) We prove an $\Omega(n/\log n + D)$ lower bound for exactly computing the weighted girth under the $\mathcal{CONGEST}$ model. The lower bound is applied for both deterministic and randomized algorithms, which means our upper bound for computing the weighted girth is optimal within a factor of $O(\log^3 n)$.

The remainder of the paper is organized as follows: In Section 2, we introduce the system model and the problem definition. In Section 3, we give two straightforward methods together with two insights we gained for computing the weighted girth. Section 4 provides an overview of techniques used in our algorithm. The algorithm details are presented in Section 5. The correctness and round complexity analyses are given in Section 6. In Section 7, we prove a lower bound for distributively computing the weighted girth under the $\mathcal{CONGEST}$ model. We conclude the paper in Section 8.

2 Preliminary

2.1 The System Model

We are given a network of processors modeled by an undirected weighted graph $G(V, E, w)$, where $|V| = n$, $|E| = m$ and $w(u, v)$ is the integral weight of edge (u, v). The maximum edge weight of the graph is $W \in [1, n^c]$, where c is a constant. Each vertex can be represented by an $O(\log n)$-bit size unique identifier (ID) and it only knows very little topological knowledge, including its neighbors' IDs and the weights of its incident edges. Denote $N(v)$ as a set of neighbors of vertex v.

We consider a synchronous communication model where each vertex can only send $O(\log n)$ bits of information through its incident edges in one synchronous round. In addition, each vertex can send different messages of size $O(\log n)$ to its neighbors and each vertex's local computation time is neglected. This model is called $\mathcal{CONGEST}$ model, which is widely employed in the distributed setting [18]. To measure the performance of a distributed algorithm, we introduce *round complexity*, which is defined as the number of rounds until the algorithm terminates. Note that, comparing with the $\mathcal{LOCAL}$ model where each edge can transmit unbounded size of messages, designing a low-round-complexity distributed algorithm under the $\mathcal{CONGEST}$ model faces greater challenges.

2.2 Problem Definition

Let $G = (V, E, w)$ be a weighted undirected graph, a simple path is defined as a sequence of vertices $\{v_1, v_2, \cdots, v_l\}$, $(v_i, v_{i+1}) \in E$ for every $i < l$ $(2 \leqslant l \leqslant n)$. We define paths from u to v as set $Paths(u, v)$. All the shortest paths from u to v constitute set $SPaths(u, v)$. We define $P(u, v)$ as an arbitrary path in $Paths(u, v)$ and define $SP(u, v)$ as an arbitrary shortest path in $SPaths(u, v)$. Denote $w(P(u, v))$ and $w(SP(u, v))$ as the sum of weights of edges in $P(u, v)$ and in $SP(u, v)$, respectively.

When we talk about Breadth-First-Search (BFS) in weighted graphs, we will use $hop(P(u, v))$ to measure the number of hops on path $P(u, v)$. Define $hop(u, v)$ as the minimum number of hops between u and v, i.e. $hop(u, v) = \min\{hop(P(u, v)) : P(u, v) \in Paths(u, v)\}$. The hop diameter D of the weighted graph is the maximum value of $hop(u, v)$ for all pairs u, v in G. Denote $Pre_s(v)$ as the predecessor of v in a BFS tree rooted at s. In addition, we denote a BFS started from vertex s as BFS_s. There is only one path $P(s, v)$ from vertex s to some vertex v in BFS_s. We define this path as $P_{BFS}(s, v)$. If $P(s, v)$ is a shortest path from s to v, we define it as $SP_{BFS}(s, v)$.

Define the cycle as a set of vertices appearing in ordering: $C = \{v_1, v_2, \cdots, v_l\}$, $(v_i, v_{i+1}) \in E$ for every $i < l$ $(3 \leqslant l \leqslant n)$ and $(v_l, v_1) \in E$. Define $w(C)$ as the sum of weights of edges in C. The weight of the minimum weight cycle, i.e., the weighted girth, is the minimum value of all $w(C)$ in G. We denote this value as g and $g \in [3, n^{c+1}]$. In addition, a path from u to v on cycle C in clockwise is defined as $P_C(u, v)$. If the path is the shortest path, define it as $SP_C(u, v)$. This path is unique corresponding to cycle C.

In this paper, we aim to devise a low round complexity distributed algorithm for exactly computing g in undirected weighted graphs under the $\mathcal{CONGEST}$ model. Define $d(u, v)$ as an estimation of the weighted shortest distance from u to v during the execution of the algorithm. $d(u, v)$ is initialized to infinity, i.e. $d(u, v) = \infty$ for each pair of u and v at the beginning of the algorithm. The minimum value of $d(u, v)$ is $w(SP(u, v))$. The notations and their definitions are listed in Table 1.

3 Bottlenecks of Previous methods for Computing the Weighted Girth

In this section, we give two straightforward distributed methods together with their analyses for computing the weighted girth. Based on the analyses of the bottlenecks, we present the two insights for devising our distributed algorithm.

Claim 1 (Lemma 3.1 of [9])**.** For a minimum weight cycle C and its weight $w(C) = g$, there exist three vertices $s, u, v \in C$ and edge (u, v) such that C consists of shortest paths $SP_C(s, u)$, $SP_C(s, v)$ and an

Table 1 Notations and Their Definitions

Notation	Definition
n	the number of vertices in a graph
m	the number of edges in a graph
W	the maximum edge weight of a graph
D	the hop diameter of a weighted graph
BFS_s	a BFS tree originating from vertex s
$Paths(s,v)$	the set of paths from vertex s to vertex v
$SPaths(s,v)$	the set of shortest paths from vertex s to vertex v
$P(s,v)$	a path from vertex s to vertex v
$SP(s,v)$	a shortest path from vertex s to vertex v
$P_{BFS}(s,v)$	the path from vertex s to vertex v in BFS_s
$P_C(s,v)$	the path from s to v on cycle C in clockwise
$SP_C(s,v)$	the shortest path from s to v on cycle C in clockwise
$w(u,v)$	the weight of edge (u,v)
$w(P(s,v))$	the sum of edge weights in path $P(s,v)$
$w(C)$	the sum of weights of edges in C
$N(v)$	the set of neighbors of vertex v
$hop(P(u,v))$	the number of hops on path $P(u,v)$
$hop(u,v)$	the minimum number of hops between u and v
$Pre_s(v)$	the predecessor of vertex v in the BFS_s tree
g	the value of the weighted girth
$d(u,v)$	the estimation of the shortest distance from u to v

edge (u,v). Furthermore, $w(SP_C(s,u)) \leqslant g/2$ and $w(SP_C(v,s)) \leqslant g/2$.

Claim 1 shows an important property of the minimum weight cycle. From Claim 1, we need to compute the shortest paths values $d(s,v) = w(SP(s,v))$ and $d(s,u) = w(SP(s,u))$ for each vertex pair s,v and s,u for computing the weighted girth. As a result, the total time (number of rounds) for computing the weighted girth is the time for computing the shortest paths plus the time for computing $w(C) = d(s,u) + d(s,v) + w(u,v)$ for each pair s,u and s,v. The state-of-the-art distributed shortest paths algorithm [17] on weighted graphs takes $\tilde{O}(n^{3/4}\lambda^{1/2} + n)$ rounds for computing λ-sources shortest paths[2]. Thus, we can take $\tilde{O}(n^{5/4})$ rounds to compute shortest paths. $w(C)$ can be computed at vertex u if vertex v sends value $d(s,v)$ to u. For each vertex, it sends at most n shortest paths values to its neighbors, which takes $O(n)$ rounds. The total rounds are still $\tilde{O}(n^{5/4})$. From the above analysis, we know that the number of shortest paths computations dominate the round complexity for computing the weighted girth. So the first insight is that if we can reduce the number of shortest paths computations for computing the weighted girth, the round complexity will be reduced to a great extent.

Second, we can look back to the distributed girth algorithms on unweighted graphs [10], [14], which mainly utilize the distributed BFS algorithm. These algorithms are based on the fact that if a vertex v is visited by a BFS_s twice, a cycle C will be computed at v. For BFS_s in unweighted graph, it must visit v along a shortest path from s to v for the first time. Denote this shortest path as $SP_{BFS}(s,v)$ and v can get $d(s,v) = w(SP_{BFS}(s,v))$ when it was visited. BFS_s will visit v from v's neighbor u for the second time. The path from s to u is also the shortest path $SP_{BFS}(s,u)$ and u can get value $d(s,u) = w(SP_{BFS}(s,u))$ when it was visited by BFS_s. Thus, $w(C)$ can be computed by $d(s,u) + d(s,v) + w(u,v)$ at v. However, efficiently applying the distributed BFS method to weighted graphs is a non-trivial task. There are two kinds of paths in weighted graphs that make the algorithm for unweighted graph hard to apply. One is the path with heavier weights but less hops and the other is with lighter weights but more hops. Taking Figure 1 as an example, there are two paths $SP_C(v,s)$ and $P(s,v)$ where $w(SP_C(v,s)) < w(P(s,v))$ and

2) λ-sources shortest paths problem is defined as follows. For vertices sets S and V where $|S| = \lambda$ and $|V| = n$, we need to compute shortest paths of all pairs s,v where $s \in S$ and $v \in V$. Note that when $\lambda = \tilde{O}(n^{1/4})$, better results are given by Elkin in [19].

$hop(SP_C(v,s)) > hop(P(s,v))$. A BFS message sent from s will arrive at v through $P(s,v)$ first. This makes $d(s,v)$ equal to $w(P(s,v))$, which is not the shortest path value. Accordingly, it will also lead to a wrong computation of the weighted girth where the minimum weight cycle is formed by $SP_C(v,s)$, $SP_C(s,u)$ and edge (u,v). A trivial approach to handle this problem is that a message needs $w(u,v)$ rounds to pass through an edge (u,v) instead of one round. By adopting this trivial method, the shortest path value can be computed correctly but it takes $O(nW)$ rounds to compute the weighted girth.

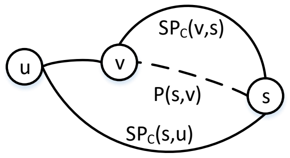

Figure 1 There is a minimum weight cycle formed by two solid paths $SP_C(s,u)$, $SP_C(v,s)$ and a solid edge (u,v). The solid path $P(s,v)$ has $w(P(s,v)) > w(SP_C(v,s))$ but $hop(P(s,v)) < hop(SP_C(v,s))$. So the BFS message from vertex s first arrives at v through $P(s,v)$. Then v sends messages to vertices in $SP(v,s)$ and vertex u, which let u receive wrong values of shortest paths.

From the above analysis, we know the distributed BFS method that sends a message along an edge with the number of rounds equivalent to its weight value takes too long time, we prefer to send a message along an edge (one hop) with just one round. This gives us the second insight: if we can find a method that correctly computes the weighted shortest paths values by sending messages hop by hop, we can greatly reduce the round complexity.

4 Technique Overview

In this paper, based on the insights in Section 3, we propose an approach that can compute the weighted girth in $O(n\log^2 n)$ rounds. The approach is based on two observations: (1) For the minimum weight cycle C and $w(C) = g$, C only consists of the shortest paths whose weights are in range $[1, g/2]$ [3]. (2) If a vertex is visited by a BFS twice, then we can compute an upper bound of the weighted girth. These two observations mean that we need not compute all pairs shortest paths. If we can find a range to bound the weighted girth and narrow this range by a proper way, the computation of shortest path can be reduced to a great extent and the weighted girth will be computed eventually.

To find a range for bounding the weighted girth and reducing the computation of shortest paths, we set an upper bound β for the weighted girth. If an algorithm has searched a part of a graph and found a cycle whose weight is an upper bound of the weighted girth, the algorithm does not need to search for the remaining part of the graph. This upper bound can be refined if we found a lesser weighted cycle. Correspondingly, we also set a lower bound α for the weighted girth, which is updated when the algorithm cannot find a cycle. During the process of narrowing the gap of α and β, our algorithm can guarantee that some vertices on the minimum weight cycle can get their weighted shortest paths correctly and some cycles' weights are computed by these vertices. The cycles' weights in turn affect values of α and β so that we can get the weight of the minimum weight cycle in the end. The details of controlling α and β are given in Section 5.2.

The main technique for searching the graph and finding cycles is based on the distributed BFS method. As described in Section 3, distributed BFS cannot compute the correct weighted shortest paths between vertices. But in our method, we do not aim to compute all pairs shortest paths by the BFS process. Instead, we modify the BFS method by adding a distance constraint and call it bounded BFS. The bounded BFS method sets a distance constraint t for each distributed BFS process where t is computed

3) This observation can be proved by a contradiction. Assuming that the minimum weight cycle C contains a shortest path $SP_C(s,v)$ where $w(SP_C(s,v)) > g/2$. $SP_C(s,v)$ is a clockwise path from s to v in the cycle. Thus, the path from v to s, denoted as $P_C(v,s)$, has $w(P_C(v,s)) < g/2$. We have $w(SP_C(s,v)) > w(P_C(v,s)$ such that $SP_C(s,v)$ is not the shortest path. This contradiction means $w(SP_C(s,v))$ must be less or equal than $g/2$.

by using α and β. For vertex v, it can be visited by bounded BFS_s only if there is a path $P(s,v)$ such that $w(P(s,v)) \leqslant t$. The bounded BFS process has two purposes, (1) find some conditions that can update lower bound α and upper bound β, (2) compute the shortest paths between some vertices in the minimum weight cycle.

Thus, our algorithm for computing the weighted girth can be sketched as follows: The execution of the algorithm is divided into phases. In each phase, each vertex v selects a random time r_v in range $[0,n]$, then it performs distributed Bounded BFS_v with distance constraint t at time r_v. After finishing the distributed Bounded BFS processes for each vertex v, the lower bound α and the upper bound β are updated. Next, the algorithm sets a new distance constraint t by these two bounds for the next phase. The algorithm terminates when $\beta - \alpha \leqslant 2$.

5 Details for Computing the Weighted Girth

In this section, we give details of our distributed algorithm for computing the weighted girth on weighted graphs where each edge weight is in $[1, n^c]$. We first give the distributed Bounded BFS algorithm, followed by the distributed weighted girth algorithm.

5.1 Distributed Bounded BFS

The distributed Bounded BFS means performing the distributed BFS process by a distance constraint t. Its purpose is to output conditions used for determining lower bound α and upper bound β in the next subsection. For a bounded BFS_s started from s with distance constraint t, vertex v will be visited by BFS_s if there is a path $P(s,v)$ where $w(P(s,v)) \leqslant t$. If there are more than one path $P(s,v)$ that $w(P(s,v)) \leqslant t$, the path $P(s,v)$ with the minimum $hop(P(s,v))$ will be visited first. We outline the Bounded BFS from source s with distance constraint t as follows and the detailed algorithm is given in Algorithm 1. In the following description, the algorithm runs $O(n)$ rounds, which is proved in Lemma 5 of Section 6.

- **Distributed bounded BFS from source s with distance constraint t:**

Vertex s sends message $\langle (s,s), d(s,s)\rangle$ to its neighbor $u \in N(s)$ that satisfies $d(s,s)+w(s,u) \leqslant t$. A message takes one round to pass an edge. Then u sets $d(s,u)$ as $d(s,s)+w(s,u)$ and sends message $\langle (s,u), d(s,u)\rangle$ to its neighbor $v \in N(u)\backslash\{s\}$ satisfying $d(s,u)+w(u,v) \leqslant t$. Each vertex v which received messages from its predecessor u will send messages to $\{N(v)\backslash u\}$ until v knows that Condition 1 occurs or $O(n)$ rounds passed. If Condition 1 occurs, the vertex which is visited by a bounded BFS twice broadcasts terminal message $\langle -1\rangle$ to the graph. Then all the vertices will stop running the algorithm after they received the terminal message or n rounds passed.

- **Condition 1:** there exists a vertex visited by a bounded BFS twice, which means a cycle is found.
- **Condition 2:** there are no vertices visited by a bounded BFS twice, which means no cycles are found.

5.2 Distributively Computing the Weighted Girth

As mentioned before, our distributed algorithm for computing the weighted girth is divided into phases. In each phase, the algorithm updates the lower bound α and the upper bound β of the weighted girth. When $\beta - \alpha \leqslant 2$, the algorithm terminates. The detailed algorithm is given in Algorithm 2, which is a distributed algorithm running at each vertex. The correctness and the round complexity analyses of this algorithm are given in Sections 6.1 and 6.2, respectively.

Before running the algorithm, we select an arbitrary vertex v_0 as the leader, which is used for computing α, β, t and then broadcasts these values. We set $\alpha = 1$ and $\beta = nW$ at the beginning of the algorithm, where W is the maximum weight of edges in the graph and it can be computed by Algorithm 3. To update α and β, the distributed Bounded BFS processes for each vertex with a given distance constraint t are performed in each phase (lines 4-27 of Algorithm 2). In phase i, t is set to be $\lfloor(\alpha+\beta)/4\rfloor$ if $\beta < nW$,

Algorithm 1 Distributed Bounded BFS running at vertex v

```
Input: Graph $G = (V, E, w)$, source $s$ and distance constraint $t$
Output: Condition 1 or Condition 2
1: Initialize $d(s, s) = 0$ and initialize $d(s, v) = \infty$ for $v \in \{V \setminus s\}$
2: Initialize $round = 0$
3: while $round < n$ do
4:   for each vertex $v$ receives message $\langle (s, u), d(s, u) \rangle$ do
5:     if $d(s, v) == \infty$ then
6:       $d(s, v) = d(s, u) + w(u, v)$
7:       $Pre_s(v) = u$
8:       for each vertex $z \in \{N(v) \backslash u\}$ do
9:         if $d(s, v) + w(v, z) \leqslant t$ then
10:          Send message $\langle (s, v), d(s, v) \rangle$ to $z$
11:        end if
12:      end for
13:    else
14:      if $d(s, u) + w(u, v) < d(s, v)$ then
15:        $Pre_s(v) = u$
16:        $d(s, v) = d(s, u) + w(u, v)$
17:        Condition 1 occurs and $v$ broadcasts message $\langle -1 \rangle$ to all vertices in $V$
18:      end if
19:    end if
20:  end for
21:  $round = round + 1$
22: end while
23: if Condition 1 does not occurs then
24:   Condition 2 occurs
25: end if
```

and otherwise $t = 2^i$. Noting that in lines 7-8 of Algorithm 2, we increase t by 1, it is used for the case that $\beta - \alpha < 5$. In this case, t is increased in order for continuing the algorithm.

After finishing the distributed Bounded BFS processes, each vertex v computes g_v by the following Equation (1) (line 18 of Algorithm 2). Here g_v is an auxiliary variable for computing α and β. It indicates that there is a cycle C formed by $P_C(s, u)$, $P_C(s, v)$ and an edge (u, v) so that g_v is the minimum weight cycle we can find in vertex v at the current phase. α and β are set according to Condition 1 and g_v at the end of each phase. Then t is updated and the algorithm starts the next phase.

$$g_v = \min_{s \in V} \{d(s, u) + d(s, v) + w(u, v) : u \in N(v), u \neq Pre_s(v), v \neq Pre_s(u)\} \tag{1}$$

The lower bound α and the upper bound β are set by the following Equations (2) and (3). When Condition 1 occurs in at least one bounded BFS process, it means a cycle has been found so that the upper bound can be reduced to $\min\{2t, \min_{v \in V} g_v\}$. When Condition 2 occurs in every bounded BFS process of vertices, there are no cycles detected by the bounded BFS processes. In this case, we should be careful to update α. As shown in Figure 2, there may be a cycle with weight less than $2t$ such that we cannot update $\alpha = 2t$ directly. So here we only update α if there is no $g_v \leqslant 2t$ or if there is no cycle found ($\min_{v \in V} g_v = \infty$). By setting α and β in this way, it can be guaranteed that $\alpha \leqslant g - 1$ and $\beta \geqslant g$ in each phase. We postpone the proof to Section 6.

- **if Condition 1 occurs in at least one bounded BFS process**

$$\alpha = \alpha, \quad \beta = \min\{2t, \min_{v \in V} g_v\} \tag{2}$$

- **if Condition 2 occurs in all the bounded BFS processes**

$$\begin{cases}\alpha = 2t, \beta = \beta, \text{if } \min_{v\in V} g_v = \infty \text{ or } \min_{v\in V} g_v > 2t \\ \alpha = \alpha, \beta = \min\{\beta, \min_{v\in V} g_v\}, \text{otherwise}\end{cases} \tag{3}$$

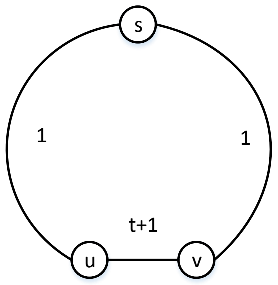

Figure 2 In this example, the weights of edges (s, u), (s, v) and (u, v) are 1, 1 and $t+1$, respectively, where $t > 3$. Any bounded BFS with distance constraint t cannot detect a cycle directly, but we can get a value by g_v according to Equation 1.

In Algorithm 2, we employ the random delay technique [20] to set the starting time of each distributed Bounded BFS process. In each phase, each vertex v selects a random time r_v in range $[0, n]$, then the distributed Bounded BFS_v starts at time r_v. Algorithm 3 and Algorithm 4 are subroutines of computing the weighted girth. They are both convergecast procedures based on a BFS tree T rooted at an arbitrary leader v_0: The messages are delivered from leaves to the root of the tree. During the delivering process, each internal vertex of the tree collects its receiving messages and does a local computation, then packets the result in a message and sends the message to its parent in the tree. The correctness and the round complexity analyses of these algorithms are given in Sections 6.1 and 6.2, respectively.

6 Algorithms Analyses

6.1 Correctness Analysis

Theorem 1. Algorithm 2 will terminate when $\beta - \alpha \leqslant 2$, and the weighted girth will be computed correctly.

Before proving Theorem 1, we introduce several properties involved in the proof of Theorem 1 and we will prove them later.

Proposition 1. In all the phases during the execution of Algorithm 2, $\min_{v\in V} g_v \geqslant g$ and $\min_{v\in V} g_v = g$ if $2t = g - 1$.

Proposition 2. In all the phases during the execution of Algorithm 2, $\beta \geqslant g$.

Proposition 3. During the execution of Algorithm 2: if g is even, $\alpha \leqslant g-2$ in all the phases; if g is odd, $\alpha \leqslant g - 1$ in all the phases.

Proof. [**Proof of Theorem 1**] From Proposition 2 and Proposition 3, we have $\alpha \leqslant g - 1 < g \leqslant \beta$. We are going to prove that $\beta - \alpha$ decreases in each phase.

For the phase i with distance constraint t, denote α_i and β_i as the values of α and β in the phase i, respectively. According to Equation (2) and Equation (3), β_i is at most $2t$ if β_{i-1} is updated and $\alpha_i \geqslant \alpha_{i-1}$. So we have $\beta_i - \alpha_i \leqslant 2\lfloor(\alpha_{i-1}+\beta_{i-1})/4\rfloor - \alpha_{i-1} \leqslant (\beta_{i-1}-\alpha_{i-1})/2$. If α_{i-1} is updated to $\alpha_i = 2t$, we have $\beta_i - \alpha_i = \beta_{i-1} - 2\lfloor(\alpha_{i-1}+\beta_{i-1})/4\rfloor \leqslant (\beta_{i-1}-\alpha_{i-1}+4)/2$ so that $\beta_i - \alpha_i - 4 \leqslant (\beta_{i-1}-\alpha_{i-1}-4)/2$, which means $\beta - \alpha$ decreases when $\beta - \alpha > 5$. If $\beta - \alpha \leqslant 5$, we increase t by 1 so that $\beta - \alpha$ will decrease till $\beta - \alpha \leqslant 2$ (line 8 in Algorithm 2).

According to Proposition 2 and Proposition 3, if g is even, the algorithm only terminates when $\beta = g$ and $\alpha = g - 2$. So we obtain the weighted girth $g = \beta$ when the algorithm terminates. If g is odd, the

Algorithm 2 Distributively Computing the Weighted Girth on vertex v

```
Input: Graph $G = (V, E, w)$, $W$, $v_0$
Output: The weighted girth $g$
1: Initialize $i = 0$, $\alpha = 1$, $\beta = nW$, $t = 1$
2: Initialize $d(v, v) = 0$, $d(u, v) = \infty$ for each $u \in \{V \backslash v\}$
3: while $\beta - \alpha > 2$ do
4:   for each vertex $v$ at Phase $i$ do
5:     if $v == v_0$ then
6:       if $\beta < nW$ then
7:         if $t == \lfloor(\alpha + \beta)/4\rfloor$ then
8:           Set $t = t + 1$
9:         else
10:          Set $t = \lfloor(\alpha + \beta)/4\rfloor$
11:        end if
12:      else
13:        Set $t = 2^i$
14:      end if
15:      $v_0$ broadcasts $t$, $\alpha$ and $\beta$ to the graph
16:    end if
17:    $v$ randomly selects a time $r_v$ from $[0, n]$ and starts *Distributed Bounded BFS* at time $r_v$
18:    Compute $g_v$ by Equation (1) and inform $v_0$ whether Condition 1 occurs
19:    if $v == v_0$ then
20:      Compute $\min_{v \in V} g_v$ by Algorithm 4
21:      if Condition 1 occurs then
22:        Update $\alpha$ and $\beta$ by Equation (2)
23:      else
24:        Update $\alpha$ and $\beta$ by Equation (3)
25:      end if
26:    end if
27:    End Phase $i$
28:    $i = i + 1$
29:  end for
30: end while
31: if $v == v_0$ then
32:   if Condition 1 did not occur in all the phases then
33:     Report there is no cycle
34:   else
35:     Compute the weighted girth $g = \beta$
36:   end if
37: end if
```

maximum value of α is $g - 1$. In the phase that α is updated to the maximum value, $\alpha = 2t = g - 1$. Then according to Proposition 1, we have $\beta = \min_{v \in V} g_v = g$. So the algorithm terminates in this phase and the weighted girth is computed.

Proof. [**Proof of Proposition 1**] We first take a deep look at Equation (1). If $d(s, u)+d(s, v)+w(u, v) \neq \infty$, it computes the weight of a cycle. $\min_{v \in V} g_v$ is the minimum weight of cycles computed by all vertices v, so $\min_{v \in V} g_v \geqslant g$.

Then we prove that in the phase with $2t = g - 1$, $\min_{v \in V} g_v = g$. When $2t = g - 1$, from Claim 1, there exist three vertices s, u and v in the minimum weight cycle such that $w(SP_C(s, u)) \leqslant t$ and $w(SP_C(v, s)) \leqslant t$, where (u, v) is an edge and $u \notin SP_C(v, s)$ and $v \notin SP_C(s, u)$. We prove $\min_{v \in V} g_v = g$ when $2t = g-1$ by contradiction. Assuming $\min_{v \in V} g_v > g$, it means u or v is not visited by BFS_s through

Algorithm 3 Computing W

```
Input: Graph $G = (V, E, w)$, $v_0$, $T$
Output: $W$
1: initialize $round = 0$
2: while $round < D$ do
3:   if $v$ is leaf in $T$ then
4:     $W_v = max\{w(u, v) : u \in N(v)\}$
5:     send $W_v$ to $Pre_{v_0}(v)$
6:   else if $v$ receives messages from $u$ then
7:     $W_v = \max\{W_u, max\{w(u, v) : u \in N(v)\}\}$
8:     send $W_v$ to $Pre_{v_0}(v)$
9:   end if
10:  $round = round + 1$
11: end while
12: if $v == v_0$ then
13:   output $W = W_{v_0}$
14: end if
```

Algorithm 4 Convergecasting g_v

```
Input: Graph $G = (V, E, w)$, leader $v_0$, $T$
Output: $\min_{v \in V} g_v$
1: initialize $round = 0$
2: while $round < D$ do
3:   if $v$ is a leaf in $T$ then
4:     send $g_v$ to its $Pre_{v_0}(v)$
5:   else if $v$ receives messages from $u$ then
6:     send $g_v = \min g_u$ to $Pre_{v_0}(v)$
7:   end if
8: end while
9: if $v == v_0$ then
10:  output $\min_{v \in V} g_v$
11: end if
```

their shortest paths. Otherwise $g_v = d(s, u) + d(s, v) + w(u, v) = g$ such that $\min_{v \in V} g_v = g$. W.l.o.g., suppose u is visited by BFS_s through path $P(s, u)$ instead of $SP_C(s, u)$ where $P(s, u) \notin SPaths(s, u)$ (this occurs when $hop(P(s, u)) < hop(SP_C(s, u))$). So there is a cycle formed by $SP_C(s, u)$ and $P(s, u)$. Its weight is $w(SP_C(s, u)) + w(P(s, u)) \leqslant 2t$, which contradicts $g = 2t + 1$ (See the illustrating example in Figure 3). So $\min_{v \in V} g_v = g$ is proven when $2t = g - 1$.

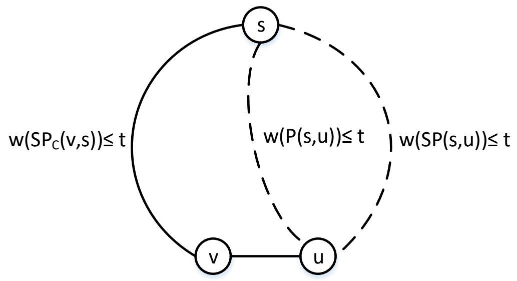

Figure 3 A cycle consists of dashed paths $SP_C(s, u)$ and $P(s, u)$, which has weight of $w(SP_C(s, u)) + w(P(s, u)) \leqslant 2t$.

Next, we give Lemmas 1 – 4 before proving Properties 2 and 3. Lemma 1 shows which vertices will be visited during the procedure of the distributed Bounded BFS process. Lemmas 3 and 4 show which phases that α or β will be updated.

Lemma 1. Given a source s and a distance constraint t, for any vertex v, if there exists a path $P(s,v)$ with $w(P(s,v)) \leqslant t$, then only two cases will occur: 1) BFS_s finds a cycle before visiting v. 2) v is visited by BFS_s.

Proof. If there is a vertex visited by BFS_s twice before v is visited, according to Condition 1, the phase is terminated. In this case, vertex v will not be visited. If no cycles have been found, Condition 2 occurs and v will be visited since there is a path $P(s,v)$ with $w(P(s,v)) \leqslant t$.

Lemma 2. In a phase with distance constraint t in which Condition 1 occurs, $g \leqslant 2t$.

Proof. Consider vertices s and v, where s is the source of the Bounded BFS_s and v is the first vertex visited by BFS_s twice, there are two paths $P(s,v) \leqslant t$ and $\hat{P}(s,v) \leqslant t$ such that $g \leqslant w(P(s,v)) + w(\hat{P}(s,v)) \leqslant 2t$.

Lemma 3. In a phase with distance constraint t such that $2t < g$, α is updated and vice versa.

Proof. We first prove that when $2t < g$, α is updated. In the phase with $2t < g$, we prove that Condition 1 will not occur by contradiction. If Condition 1 occurs, it means a cycle with weights at most $2t$ is found, which contradicts $2t < g$. So in this phase, Condition 2 occurs in all the bounded BFSs. From Proposition 1, $\min_{v\in V} g_v \geqslant g > 2t$. From Equation (3), α is updated when $\min_{v\in V} g_v > 2t$. So in the phase with $2t < g$, α is updated.

We next prove that when α is updated, $g > 2t$. From Equation (3), when α is updated, $\min_{v\in V} g_v > 2t$. From Proposition 1, $\min_{v\in V} g_v \geqslant g$. So if $\min_{v\in V} g_v = g$, $g > 2t$ is proven. Then we need to prove that $g > 2t$ when $\min_{v\in V} g_v > g$.

We are going to prove $g > 2t$ when $\min_{v\in V} g_v > g$ by contradiction. Assuming $g \leqslant 2t$, from Claim 1, there exist three vertices s, u and v in the minimum weight cycle, where (u,v) is an edge and $w(SP_C(s,u)) \leqslant g/2 \leqslant t$ and $w(SP_C(v,s)) \leqslant g/2 \leqslant t$. So according to Lemma 1 and the fact that α is only updated when Condition 2 occurs in all the bounded BFSs, u and v must be visited by BFS_s through shortest paths $SP_C(s,u)$ and $SP_C(v,s)$. By Equation (1), we have $g_v = d(s,u) + d(s,v) + w(u,v) = g$ and $\min_{v\in V} g_v = g$, which contradicts the inequality $\min_{v\in V} g_v > g$. So when $\min_{v\in V} g_v > g$, $g > 2t$ is proven.

Lemma 4. In a phase with distance constraint t such that $2t \geqslant g$, β is updated and vice versa.

Proof. We first prove that if β is updated, $2t \geqslant g$. From Equations (2) and (3), β will be updated in Condition 1 or in Condition 2. If Condition 1 occurs, according to Lemma 2, $g \leqslant 2t$. If Condition 2 occurs in all the bounded BFSs, β is updated only when $\min_{v\in V} g_v \leqslant 2t$. From Proposition 1, $g \leqslant 2t$ is proven.

We then prove that if $2t \geqslant g$, β will be updated. In the phase with $2t \geqslant g$, either Condition 1 or Condition 2 will occur. If Condition 1 occurs, β is updated by Equation (2). If Condition 2 occurs in all the bounded BFSs, we prove that inequality $g \leqslant \min_{v\in V} g_v \leqslant 2t$ must hold by contradiction. Assuming $g \leqslant 2t < \min_{v\in V} g_v$, from Claim 1, there exist three vertices s, u and v in the minimum weight cycle, where (u,v) is an edge and $w(SP_C(s,u)) \leqslant g/2$ and $w(SP_C(v,s)) \leqslant g/2$. Since $g/2 \leqslant t$, according to Lemma 1 and Condition 2, u and v will be visited by BFS_s through shortest paths $SP_C(s,u)$ and $SP_C(v,s)$. We have $g_v = d(s,u) + d(s,v) + w(u,v) = g$ by Equation (1) and $\min_{v\in V} g_v = g$, which contradicts the inequality $g \leqslant 2t < \min_{v\in V} g_v$. So $g \leqslant \min_{v\in V} g_v \leqslant 2t$ is proven when Condition 2 occurs. From Equation 3, β is updated.

Proof. [**Proof of Proposition 2**] From Equation (2) and Equation (3), we know that β is updated to $\min\{2t, \min_{v\in V} g_v\}$ when Condition 1 occurs and is updated to $\min_{v\in V} g_v$ when Condition 2 occurs in all the bounded BFSs. From Proposition 1 and Lemma 4, $\min_{v\in V} g_v \geqslant g$ and $2t \geqslant g$ in all the phases when β is updated. We conclude that $\beta \geqslant g$ during the procedure of Algorithm 2.

Proof. [**Proof of Proposition 3**] According to Equation (3), α can only be updated to $2t$. From Lemma 3, we know if α is updated, $g > 2t$. So in all the phases, $\alpha < g$. We next prove that in all these phases, $\alpha \leqslant g - 2$ if g is even and $\alpha \leqslant g - 1$ if g is odd.

If g is even, according to Lemma 3, the maximum value for t when α is updated is $t = g/2 - 1$. So we have $\alpha = 2t = g - 2$, which is the maximum value of α in this case.

If g is odd, according to Lemma 3, the maximum value for t when α is updated is $t = \lfloor g/2 \rfloor$. So $\alpha = 2t = g - 1$ is the maximum value for α in this case.

6.2 Round Complexity

We first prove that Algorithm 1 can be finished in $O(n)$ rounds. Then we prove that in each phase of Algorithm 2, performing n parallel distributed Bounded BFS algorithms can be finished in $O(n \log n)$ rounds with probability at least $1 - 1/n^2$ by Lemma 6. The round complexity of Algorithm 2 will be analyzed in Theorem 2.

Lemma 5. The Distributed Bounded BFS algorithm (Algorithm 1) terminates in $O(n)$ rounds.

Proof. For any two vertices u and v, any path $P(u,v)$ has $hop(P(u,v)) < n$. So no matter what conditions of Algorithm 1 occur, the algorithm will terminate in $O(n)$ rounds.

Lemma 5 shows that the bounded BFS is different with normal BFS which always takes $O(D)$ rounds. Algorithm 1 may not terminate in $O(D)$ rounds since there exists cases that only the paths with large hops have weights less than t. For example, if there only exist one path $P(s,u)$ that satisfies $w(P(s,u)) \leqslant t$ and $hop(P(s,u)) = n - 1$. Algorithm 1 will terminate at round n.

Lemma 6. In each phase of Algorithm 2, each vertex performs the distributed Bounded BFS algorithm at a random time in range $[0,n]$. This procedure can be finished in $O(n \log n)$ rounds without violating the $\mathcal{CONGEST}$ model with probability at least $1 - 1/n^2$.

Proof. We first prove that in each phase, the probability that any vertex sends more than $\log n$ messages in each round is at most $1/n^2$.

For each phase, a given vertex v can send at most one message in BFS_s. Denote $Event_{s,v,q}$ as an event that a message originated from s is sent by vertex v at time q. If $w(SP(s,v)) > t$, $Event_{s,v,q}$ will not happen. If $w(SP(s,v)) \leqslant t$ and the number of hops from s to v in BFS_s is h, we know that s must start its Bounded BFS at time $q - h$ if $Event_{s,v,q}$ happens. While the start time of BFS_s is selected from $[0,n]$ randomly and uniformly, so we get the probability $\Pr(Event_{s,v,q}) \leqslant \frac{1}{n}$.

Suppose M is the set of messages originated from different BFSs that vertex v will send in a certain round, then v has $\binom{n}{|M|}$ different kinds of set M. Let ξ denote an event that more than $\log n$(i.e.$|M| \geqslant \log n$) messages are sent by vertex v in one round. Then the probability of ξ is

$$\begin{aligned}
\Pr(\xi) &\leqslant \sum_{|M|=\log n}^{n} \binom{n}{|M|} (\frac{1}{n})^{|M|} (1 - \frac{1}{n})^{n-|M|} \\
&\leqslant \sum_{|M|=\log n}^{n} (\frac{ne}{|M|})^{|M|} (\frac{1}{n})^{|M|} \\
&= \sum_{|M|=\log n}^{n} (\frac{e}{|M|})^{|M|} \\
&\leqslant \sum_{|M|=\log n}^{n} (\frac{e}{\log n})^{\log n} \leqslant \frac{1}{n^4}
\end{aligned}$$

By using the union bound we conclude that in each phase, the probability of any vertex sending more than $\log n$ messages in one round is at most $1/n^2$. In the $\mathcal{CONGEST}$ model, sending at most $\log n$ messages takes $O(\log n)$ rounds. To get the round complexity of performing n bounded BFS, we break the rounds into epochs where each epoch consists of $O(\log n)$ rounds. Each vertex starts the BFS at epochs of range $[0,n]$ according to random delay time, and the latest bounded BFS finishes at epoch $n + n$ according to Lemma 5. The total rounds we get in the $\mathcal{CONGEST}$ model are $O(n \log n)$.

Lemma 7. Setting α and β takes $O(n)$ rounds in each phase of Algorithm 2.

Proof. From Equation (1), g_v is the minimum value of $d(s,u)+d(s,v)+w(u,v)$ and there are at most n different s. It takes $O(n)$ rounds to compute g_v. Construct a BFS tree or perform a broadcast process takes $O(D)$ rounds in the $\mathcal{CONGEST}$ model [18]. Algorithm 4 takes $O(D)$ rounds to compute $\min_{v\in V} g_v$. So there are $O(n)$ rounds in total to set α and β.

Theorem 2. Algorithm 2 computes the correct weighted girth value in $O(n\log^2 n)$ rounds under the $\mathcal{CONGEST}$ model with probability at least $1-c\log n/n^2$, where c is a positive constant.

Proof. If Algorithm 2 successfully computes the weighted girth without violating the $\mathcal{CONGEST}$ model, the algorithm has $O(\log g)$ phases. At the beginning of the algorithm, t is updated by $t=2^i$ in phase i. From Lemma 4, β is updated for the first time no later than phase $\log(g/2)$. At that time, $\beta=O(g)$. So in phase $\log(g/2)$, $\beta-\alpha=O(g)$. For the following phases, we have $t=\lfloor(\alpha+\beta)/4\rfloor$. In these phases, α_i and β_i are denoted as α and β in phase i. If β_{i-1} is updated to β_i, according to Equation (2) and Equation (3), β_i is at most $2t$. So we have $\beta_i-\alpha_i\leqslant 2\lfloor(\alpha_{i-1}+\beta_{i-1})/4\rfloor-\alpha_{i-1}\leqslant(\beta_{i-1}-\alpha_{i-1})/2$. If α_{i-1} is updated to $\alpha_i=2t$, we have $\beta_i-\alpha_i=\beta_{i-1}-2\lfloor(\alpha_{i-1}+\beta_{i-1})/4\rfloor\leqslant(\beta_{i-1}-\alpha_{i-1}+4)/2$ so that $\beta_i-\alpha_i-4\leqslant(\beta_{i-1}-\alpha_{i-1}-4)/2$. According to the above analysis, $\beta-\alpha$ will decrease to 5, then it takes $O(1)$ rounds to decrease to 2 or 1 (line 8 in Algorithm 2). Thus, we conclude that there are $O(\log g)$ phases until $\beta-\alpha\leqslant 2$.

According to Lemma 6, we know that in each phase, the probability that any vertex sends more than $\log n$ messages in one round is at most $1/n^2$. The algorithm has $O(\log n)(g=n^{c+1})$ phases, so by using the union bound we know that in all phases, the probability that any vertex sends more than $\log n$ messages in one round is at most $c\log n/n^2$, which means Algorithm 2 can be implemented distributively without violating the $\mathcal{CONGEST}$ model with probability at least $1-c\log n/n^2$. Algorithm 3 is a convergecast process, which takes $O(D)$ rounds [18]. From Lemma 7, setting α and β takes $O(n)$ rounds. So there are $O(n\log^2 n)$ rounds to finish all the phases.

7 Lower bound

In this section, we give an $\Omega(D+n/\log n)$ rounds lower bound for distributively computing the weighted girth under the $\mathcal{CONGEST}$ model, based on the reduction technique from [16]. The lower bound result of distributively computing the weighted girth is concluded in Theorem 3.

Theorem 3. For some weighted graph with n vertices and hop diameter D, any distributed algorithm for computing the exact weighted girth needs at least $\Omega(n/\log n+D)$ rounds under the $\mathcal{CONGEST}$ model. This bound holds for both randomized and deterministic algorithms.

First we give the definition of two-party communication complexity, which was introduced by Yao in [21]. Then we will construct a graph and reduce a basic two-party communication complexity problem to the weighted girth problem.

Definition 1 (Two-Party Communication Complexity [22])**.** We are given two players Alice and Bob with arbitrary input strings a and b, respectively. They want to compute a function h of a and b with error probability at most ϵ. Denote the set of two party algorithms for computing the function as A_ϵ. Let $A\in A_\epsilon$ be an algorithm to compute h. Denote $R_\epsilon^{cc}(A(a,b))$ as the number of bits that Alice and Bob need to exchange on inputs a and b using A. We define

$$R_\epsilon^{cc}(h)=\min_{A\in A_\epsilon}\max_{a,b} R_\epsilon^{cc}(A_\epsilon(a,b))$$

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

We will construct a graph given input strings a and b in Definition 1, then reduce a well studied problem called set disjointness to our problem.

Definition 2 (Set Disjointness)**.** The set disjointness function $DISJ_k:\{0,1\}^k\times\{0,1\}^k\to\{0,1\}$ is defined as

$$DISJ_k(a,b)=\begin{cases}0, & a[i]=b[i]=1 \text{ for some } i \text{ in } [0,k-1]\\ 1, & \textit{otherwise}\end{cases}$$

Lemma 8 (Example 3.22 of [22])**.** For any sets $a \in \{0,1\}^k$ and $b \in \{0,1\}^k$, the communication complexity for computing $DISJ_k$ is $\Omega(k)$, i.e. $R^{cc}_{\epsilon}(DISJ_k) = \Omega(k)$.

The connection between the two-party communication complexity and the round complexity of distributed graph problems is given in Lemma 10. We first define the "cut" of the graph used in the lemma.

Definition 3 (Cut [16])**.** Given a graph $G = (V, E)$, a cut (G_l, G_r, C_k) is a partition of G into two disjoint subgraphs G_l and G_r and a cut-set $C_k \subseteq E$. Denote c_k as the size of the cut-set. C_k consists of $c_k = |C_k|$ edges whose endpoints are in different subsets of the node-partition.

Now for any graph G and cut (G_l, G_r, C_k), we define a two-party communication problem $f'((G_l, C_k), (G_r, C_k)) := f(G)$, where $f(G)$ is a graph problem. When computing $f'((G_l, C_k), (G_r, C_k))$, Alice will get input (G_l, C_k) and Bob will get input (G_r, C_k). Then we can get the following Lemmas 9 and 10.

Lemma 9 (Lemma 4.1 of [16])**.** The function f' can be reduced to f.

Lemma 10 (Theorem 4.1 of [16])**.** Let $B \geqslant 1$ be the number of bits each edge can transfer in each round, $f(G)$ be a function on graph G and $f'((G_l, C_k), (G_r, C_k))$ be a function derived from $f(G)$ as described above, then we have

$$\frac{R^{cc}_{\epsilon}(f')}{2c_k \cdot B} \leqslant R^{dc}_{\epsilon}(f),$$

where $R^{cc}_{\epsilon}(f')$ is the communication complexity for computing f', $R^{dc}_{\epsilon}(f)$ is the distributed round complexity for computing f.

Now we need to construct the graph for proving the lower bound of distributively computing weighted girth. Construct a graph G with two parts G_l and G_r, where each part consists of $2k(n)$ vertices, where $k(n) = \lfloor \frac{n}{c} \rfloor$ (c is a constant). We denote the vertices in G_l as L_0, L_1, $\cdots$, $L_{2k(n)-1}$ and the vertices in G_r as R_0, R_1, $\cdots$, $R_{2k(n)-1}$. For each pair of L_i and R_i, we add an edge with weight equal to 1. In addition, we add sets A and B of vertices connected to L_0 and R_0, each of them is a path that consists of $\frac{n-2k(n)}{2}$ vertices and the weights of edges between these vertices are 1. We can also connect these vertices to each of L_i and R_i to reduce the diameter if we need. But here we only connect these vertices into two paths for the sake of simplicity.

Then we are going to reduce the communication complexity problem to the weighted girth problem on this graph. For given sets $a \in \{0,1\}^{k(n)^2}$ and $b \in \{0,1\}^{k(n)^2}$, Alice uses a to construct G_l and Bob uses b to construct G_r. If $a[i] = 1$, we add an edge (L_u, L_v) with $w(L_u, L_v) = W > 2$, where $u = (i \mod k(n))$ and $v = (k(n) + \lfloor i/k(n) \rfloor)$. If $a[i] = 0$, we add an edge (L_u, L_v) with $w(L_u, L_v) = W'$, where $W' = 2W$. The same construction is applied to G_r by Bob with input b. An example of this graph is illustrated in Figure 4.

In order to prove the lower bound of computing the weighted girth, as mentioned above, we need to reduce the set disjointness problem to the weighted girth problem. Lemma 11 shows that whether the inputs a and b are disjoint or not corresponds to the different weights of cycles.

Lemma 11. For a graph G constructed by Alice and Bob with input sets $a, b \in \{0,1\}^{k(n)^2}$, we have a function $g(G, a, b)$ that computes the weighted girth.

$$g(G,a,b) = \begin{cases} 2W+2, & DISJ_{k(n)^2}(a,b) = 0 \\ > 2W+2, & DISJ_{k(n)^2}(a,b) = 1 \end{cases}$$

Proof. From the graph construction, we know that there are several ways to form a cycle. We discuss these cases as follows.

Case 1: The cycle is formed by edges in G_l or in G_r. In this case, the cycle only consists of edges (L_i, L_j) (or (R_i, R_j)) where $i < k(n)$ and $j \geqslant k(n)$. From the graph construction, if $a[(j-k(n))k(n)+i] = 0$, it means there is an edge between L_i and L_j with $w(L_i, L_j) = W'$, and otherwise $w(L_i, L_j) = W$. In this case, we need at least 4 edges (L_i, L_j) to form a cycle. Since $W' = 2W$, the weight of this cycle is at least $4W$. An example is illustrated in Figure 5.

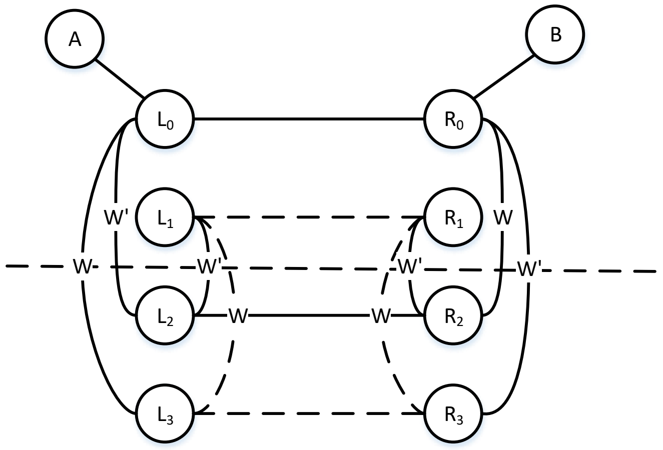

Figure 4 In this example, $k(n) = 2$, $a = [0, 0, 1, 1]$, $b = [1, 0, 0, 1]$. So we connect L_0 and L_2 with weight W', connect L_1 and L_2 with weight W', connect L_0 and L_3 with weight W, connect L_1 and L_3 weight W. G_r is constructed via b in the same way. Since $a[3] = b[3] = 1$, it means a and b are not disjoint and there is a cycle (L_1, L_3, R_3, R_1) with the minimum weight $2W + 2$ formed by dotted lines.

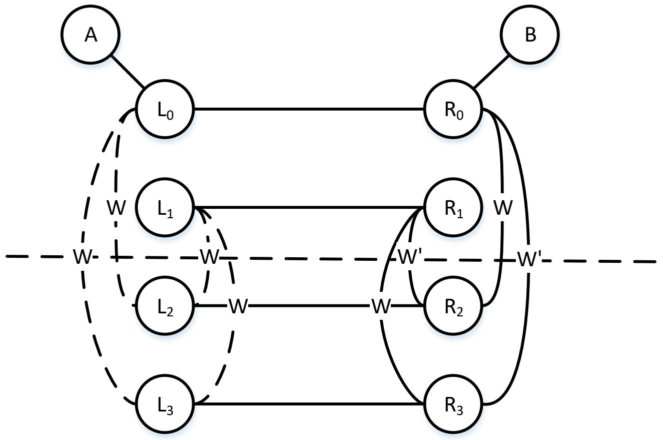

Figure 5 In this example, a cycle of weight $4W$ is formed by four dotted edges in G_l. It is the minimum weight cycle that can be formed in G_l.

Case 2: The cycle is formed by edges in G_l, G_r and C_k. In this case, the cycle consists of vertices L_i, L_j in G_l and R_i, R_j in G_r. According to the construction of our graph, if $a[(j - k(n))k(n) + i] = 0$ and $b[(j - k(n))k(n) + i] = 0$, we have a cycle with weight $2W' + 2$. If $a[(j - k(n))k(n) + i] = 0$ and $b[(j - k(n))k(n) + i] = 1$, or $a[(j - k(n))k(n) + i] = 1$ and $b[(j - k(n))k(n) + i] = 0$, we have a cycle with weight $W' + W + 2$. If $a[(j - k(n))k(n) + i] = 1$ and $b[(j - k(n))k(n) + i] = 1$, we get the minimum weight cycle with weight $2W + 2$.

From the above, if there exists $a[i] = b[i] = 1$, we get a cycle with weight $2W + 2$. Since $W' = 2W > 4$, the weights of all the other cycles in the graph are greater than $2W + 2$. From Definition 2, we conclude that $g(G, a, b) = 2W + 2$ if and only if $DISJ_{k(n)^2}(a, b) = 0$ and $g(G, a, b) > 2W + 2$ if and only if $DISJ_{k(n)^2}(a, b) = 1$.

Proof. [Proof of Theorem 3] Given the graph constructed on inputs a and b, from Lemma 11 we know that we can compute $DISJ_{k(n)^2}$ by distinguishing whether there is a cycle of weight $2W + 2$ or not, which means that the problem of set disjointness is reduced to our problem successfully. From Lemma 10, we have $R^{cc}_{\epsilon}(DISJ_{k(n)^2}(a, b))/(2c_k \cdot B) \leqslant R^{dc}_{\epsilon}(g(G, a, b))$ where $c_k = 2k(n) = O(n)$ and $B = O(\log n)$. Then we have $R^{dc}_{\epsilon}(g(G, a, b)) \geqslant \Omega(n/\log n)$. Therefore we can get that the lower bound for distributively computing the exact weighted girth is $\Omega(n/\log n + D)$ rounds.

8 Conclusion

In this paper, we propose an $O(n \log^2 n)$ rounds distributed randomized algorithm for computing weighted girth in the $\mathcal{CONGEST}$ model. The lower bound we constructed means the upper bound is optimal within a factor of $O(\log^3 n)$. The algorithm is a randomized algorithm and it only focuses on weighted undirected graphs. For directed graphs, our algorithm cannot guarantee the round complexity. The reason is that Condition 1 of the bounded BFS algorithm is changed and the algorithm does not terminate when the condition occurs. In directed graphs, a vertex v visited by a bounded BFS twice cannot determine whether a cycle has been found. In this case, v needs to delivery messages to its neighbors so that a tricky problem comes back to us. How to handle the congestion if we parallel n bounded BFSs? Although we have some methods to let messages be delivered in the graph without congestion, the round complexity will be too high for us.

For deterministic algorithm, we have not found a deterministic algorithm that can schedule the bounded BFS in an acceptable round complexity. How to design such an algorithm is the main future work for us. In addition, it is very interesting to apply our algorithm in planar graphs, which have many properties that might be helpful for us to speedup the algorithm even bypass the lower bound of the general graph. Finally, implementing the distributed algorithm in the dataflow model might be also an interesting future work [23].

Acknowledgements We thank the anonymous reviewers for the helpful comments to improve the presentation of this paper. This work is supported in part by National Key Research and Development Program of China under Grant No. 2018YFB1003203, the National Natural Science Foundation of China Grants No. 61972447, and the Fundamental Research Funds for the Central Universities No. 2019kfyXKJC021.

References

1 Diestel R. Graph Theory, 4th Edition. Springer, 2012
2 Bernstein A. A nearly optimal algorithm for approximating replacement paths and k shortest simple paths in general graphs. In: Proceedings of ACM-SIAM symposium on Discrete algorithms (SODA), 2010. 742-755
3 Williams V V, Williams R. Subcubic equivalences between path, matrix and triangle problems. In: Proceedings of Symposium on Foundations of Computer Science (FOCS), 2010. 645-654
4 Itai A, Rodeh M. Finding a minimum circuit in a graph. SIAM J. Comput., 1978, 7(4): 413-423
5 Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. J. Symb. Comput, 1990. 9(3): 251-280
6 Roditty L, Williams V V. Minimum weight cycles and triangles: Equivalences and algorithms. In: Proceedings of Symposium on Foundations of Computer Science (FOCS), 2011. 180-189
7 Williams V V. Hardness of Easy Problems: Basing Hardness on Popular Conjectures such as the Strong Exponential Time Hypothesis. In: Proceedings International Programme on the Elimination of Child Labour (IPEC), 2015. 17-29
8 Lingas A, Lundell E. Efficient approximation algorithms for shortest cycles in undirected graphs. Inf. Process. Lett, 2009. 109(10): 493-498
9 Roditty L, Tov R. Approximating the girth. ACM Trans. Algorithms, 2013. 9(2): 15:1-15:13
10 Holzer S, Wattenhofer R. Optimal distributed all pairs shortest paths and applications. In: Proceedings of ACM symposium on Principles of distributed computing (PODC), 2012. 355-364
11 Lenzen C, Peleg D. Efficient distributed source detection with limited bandwidth. In: Proceedings of ACM symposium on Principles of distributed computing (PODC), 2013. 375-382
12 Nanongkai D. Distributed approximation algorithms for weighted shortest paths. In: Proceedings of Symposium on the Theory of Computing (STOC), 2014. 565-573
13 Peleg D, Roditty L, Tal E. Distributed algorithms for network diameter and girth. In: Proceedings of International Colloquium on Automata, Languages and Programming (ICALP), 2012. 660-672
14 Hua Q, Fan H, Qian L, Ai M, Li Y, Shi X, Jin H. Brief announcement: A tight distributed algorithm for all pairs shortest paths and applications. In: Proceedings of Symposium on Parallelism in Algorithms and Architectures (SPAA), 2016. 439-441
15 Hua Q, Fan H, Ai M, Qian L, Li Y, Shi X, Jin H. Nearly optimal distributed algorithm for computing betweenness centrality. In: Proceedings of International Conference on Distributed Computing Systems (ICDCS), 2016. 271-280
16 Frischknecht S, Holzer S, Wattenhofer R. Networks cannot compute their diameter in sublinear time. In: Proceedings of ACM-SIAM symposium on Discrete algorithms (SODA), 2012. 1150-1162
17 Huang C C, Nanongkai D, Saranurak T. Distributed exact weighted all-pairs shortest paths in o(n5=4) rounds. In: Proceedings of Symposium on Foundations of Computer Science (FOCS), 2017. 168-179
18 Peleg D. Distributed computing a locality sensitive approach. SIAM Monographs on discrete mathematics and applications, 5, 2000

19 Elkin M. Distributed exact shortest paths in sublinear time. In: Proceedings of Symposium on the Theory of Computing (STOC), 2017. 757-770

20 Leighton F T, Maggs B M, Rao S. Packet routing and job-shop scheduling in O(congestion + dilation) steps. Combinatorica, 1994. 14(2): 167-186

21 Yao A C. Some complexity questions related to distributive computing (preliminary report). In: Proceedings of Symposium on the Theory of Computing (STOC), 1979. 209-213

22 Kushilevitz E, Nisan N. Communication complexity. Cambridge University Press, 1997

23 Jin H, Yao P, Liao X. Towards dataflow based graph processing. SCIENCE CHINA Information Sciences, 2017. 60(12): 126102:1-126102:3