Distributively Computing Girth on Planar Graphs

Chun-Xuan Zhou, Qiang-Sheng Hua
School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China.

***Abstract*—In this paper, we consider both the girth problem (the shortest cycle) and the weighted girth problem (the minimum weighted cycle), on planar graphs under the $\mathcal{CONGEST}$ model. For unweighted planar graphs, we give a distributed algorithm of $O(D\log^2 n)$ rounds for computing the girth with hop diameter D and n nodes. For weighted planar graphs, we prove that distributively computing the weighted girth problem needs $\Omega(\sqrt{n}/\log n + D)$ rounds which means there is only an $O(\sqrt{n})$ factor gap from the $O(n)$ trivial upper bound, as planar graphs only have $O(n)$ edges.**

I. Introduction

Girth is the minimum length of graph G's all cycles, and it is a combinatorial characteristic graph problem, see Diestel [1]. The girth has a close relation to many graph properties, for example, connectivity, treewidth and diameter.

In the centralized setting, the girth of a graph was first presented by Itai and Rodeh [2] with an $O(mn)$ time algorithm on an m-edge and n-node general graph. For planar graphs, Djidjev [3] gave the first $O(n^{5/4}\log n)$ time algorithm. An $O(n\log^2 n)$ time min-cut algorithm was presented by Chalermsook et al.[4]. This min-cut algorithm can settle the problem of girth in the dual planar graph through planar embedding. Weimann and Yuster [5] presented an $O(n\log n)$ time algorithm by the technical method of planar separator theorem. By means of planar separator theorem, we can utilize *divide-and-conquer* approach for designing efficient algorithms. Furthermore, Chang and Lu [6] gave a linear time $O(n)$ algorithm.

The planar separator theorem is especially efficient for many problems in planar graphs. For example, maximum matching, maximum independent set [7], APSP(All-Pairs Shortest Paths) and SSSP(Single-Source Shortest Paths) [8]. Lipton and Tarjan gave the original proof of the planar separator theorem [9]. Alon, Seymour and Thomas [10] presented a much shorter proof.

In the distributed setting, there is an $\tilde{\Omega}(\sqrt{n} + D)$ lower bound for many graph problems on general graphs [11], e.g., min-cost connected dominated set, shortest paths, min-cut [11]. For planar graphs, can we bypass this lower bound? To the best of my knowledge, this paper presents the first sublinear-time distributed exact algorithm for computing girth of planar graphs, running in $O(D\log^2 n)$ rounds. This algorithm utilizes *divide-and-conquer* strategy to separate the graph into components by a distributed algorithm for separator theorem which is based on [9]. In each component, we compute girth recursively based on [5]. Our contributions are summarized as follows:

1) We present the first distributed deterministic exact algorithm of $O(D\cdot min\{D,\log n\})$ rounds for separator theorem in planar graphs.
2) We present the first distributed exact algorithm of $O(D\log^2 n)$ rounds for computing the girth of an undirected unweighted planar graph. And this upper bound is nearly optimal to the trivial lower bound $O(D)$.
3) We give a proof of $\Omega(\sqrt{n}/\log n + D)$ lower bound for computing the weighted girth exactly of a planar graph under the $\mathcal{CONGEST}$ model. This lower bound has an $O(\sqrt{n})$ gap to the trivial upper bound $O(n)$ in planar graphs.

II. Related Work

A. Distributed Graph Algorithms in Planar Graphs

Because an $\tilde{\Omega}(D + \sqrt{n})$ lower bound exists on general graphs, some researchers have studied the efficient distributed algorithm to bypass this lower bound on planar graphs. For example, Ghaffari and Haeupler give an $\tilde{O}(D)$ rounds distributed algorithm for MST in planar graphs by the distributed planar embedding and low-congestion shortcuts [12], [13]. This means the lower bound in general graphs can be bypassed in planar graphs by a proper way.

B. Separator Theorem

The separator's existence was proven by Ungar [14] with the size of $O(\sqrt{n\log n})$. Lipton and Tarjan [9] introduced the planar separator theorem, and gave a linear time centralized algorithm for finding a separator of size $2\sqrt{2}\sqrt{n}$. And they proved that the planar separator theorem is efficient to solve some planar graph problems, as mentioned in the previous section. Djidjev [15] reduced the bound of the separator size to $\sqrt{6n}$, and gave a lower bound $(\sqrt{4\pi\sqrt{3}}/3)\sqrt{n}$.

The separator theorem has been developed into various kinds of directions for different graphs: bounded genus graphs [16], string graphs [17], planar intersection graphs [18], and graphs which has a forbidden minor [19].

III. Preliminaries

A. System Model

Our distributed algorithms work under the classical $\mathcal{CONGEST}$ model [20]. In this model, the communication is synchronous. In the n-node undirected graph G, every node does not know the graph in the beginning, and they can only communicate directly to its neighbors with $O(\log n)$ bits every round. This means if two nodes are not adjacent, their communication needs to deliver the message through a path between them.

B. Definition

Planar Embedding: A planar embedding of a planar graph means that the graph can be drawn on the plane such that no edges cross with each other unless at their endpoints. Given a planar embedding, we know that the clockwise cyclic orders of connected edges on v for every node v in the planar graph.

k-Outerplanar Graph: An outerplanar graph is a planar graph G that all its nodes belong to a single face, and this outerplanar graph is called the 1-outerplanar graph. Graph G is k-outerplanar if removing the nodes on the outer face results in a $(k-1)$-outerplanar graph.

IV. Algorithm Overview

In this section, we give an overview of techniques used in our distributed algorithm for computing girth of planar graphs.

Algorithm 1 Distributed Algorithm for Computing the Girth of a Planar Graph

```
Input: Planar Graph $G = (V, E)$
Output: the girth $g$ of the planar graph $G$
1: cover the planar graph $G$ by $k$-outerplanar graphs
2: for every $k$-outerplanar graph do
3:     find a separator of $G$ by Algorithm 2
4:     generate several components by removing the separators
5:     recursively compute the girth on each component
6: end for
```

The main idea of our distributed algorithm is a *divide-and-conquer* strategy as shown in **Algorithm** 1 (an outline of our distributed algorithm **Algorithm** 7). At the beginning, we cover the planar graph G by k-outerplanar graphs where the k-outerplanar graphs are subsets of the planar graph [5] (line 1). For the purpose of implementing the *divide-and-conquer* strategy, we utilize the **Algorithm** 2 to find a separator of the planar graph G (line 2). Then we divide the graph into components by removing the separators (line 4). Finally, we compute the girth in every k-outerplanar graph recursively and get the girth of G (line 5).

To be specific, we divide the planar graph G into overlapping layers by a distributed BFS process. Each layer is a k-ouerplanar graph. And the girth is entirely contained in one of these layers [5]. Then we design a distributed algorithm to compute the girth of k-outerplanar graphs in parallel. The girth computation is based on a lemma of [5]: If girth goes through v then it has just one edge $(u, w) \notin T$ where T is the BFS tree of G on an unweighted planar graph. Next we construct the BFS tree from every node of separator and check $d(u) + d(w) + 1$ for every edge. Finally, we compute the girth of G in $O(D \log^2 n)$ rounds.

V. Distributed Algorithm for Planar Separator Theorem

In this section, we present a distributed algorithm which finds the separator of planar graphs. **Algorithm** 2 is a distributed counterpart of the classical method of Lipton and Tarjan[9] for finding the separator of planar graphs. The algorithm partitions the graph's nodes into three different sets. Theorem 1 concludes this result and Figure 1 is a simple example of this theorem. In the algorithm, we define **cost** as the number of nodes.

Theorem 1 ([9]). *The nodes of the planar graph G can be partitioned into sets A, B and C. There is no edge joins a node in A with a node in B. The total cost of A or B is less than $2n/3$, and the cost of C is bounded by $2\sqrt{2}\sqrt{n}$.*

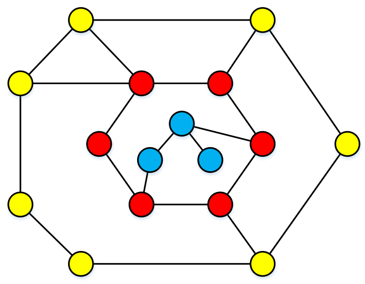

Fig. 1. The node sets A, B, C. A is the set of blue nodes, B is the set of yellow nodes, and C is the set of red nodes.

Theorem 2. *A distributed deterministic exact algorithm **Algorithm** 2 finds the separator of a planar graph in $O(D \cdot min\{D, \log n\})$ rounds with diameter D and n nodes.*

Algorithm 2 Distributed Algorithm for Planar Separator Theorem

```
Input: Planar Graph $G = (V, E)$
Output: a separator of $G$
1: find a planar embedding of $G$ by Lemma 1
2: find $G$'s connected components, and compute every component's cost by Algorithm 3
3: for each component do
4:     if the cost of component $\leq 2n/3$ then
5:         get the separator directly [9]
6:     else
7:         find two levels $l_0$ and $l_2$ by Algorithm 5
8:         construct a new graph $G'$ by $l_0$ and $l_2$
9:         find a cycle to separate $G'$ by Algorithm 6
10:        get the separator by Lemma 3 and Lemma 4
11:    end if
12: end for
```

Lemma 1 ([12]). *A distributed deterministic algorithm of [12] can find a planar embedding for the planar graph G in $O(D \cdot min\{D, \log n\})$ rounds, with diameter D and n nodes.*

The following three paragraphs (include this paragraph) are the presentation of **Algorithm** 2. First, we find a planar embedding of G by **Lemma** 1 (line 1). Then we construct a list structure for indicating the clockwise cyclic orders of connected edges around every node [9].

Next we find G's connected components, and compute every component's cost by **Algorithm** 3 (line 2). If none has the cost exceeding $2n/3$, we get the separator directly as described in Theorem 4 of [9] (lines 4-6). If a component has the cost exceeding $2n/3$, we need a further processing (lines 7-12).

At the beginning, we will find two levels l_0 and l_2 by **Algorithm 5** (line 7). Then we generate a new graph G' by these levels (line 8). After generating new graph G', we utilize **Algorithm** 6 to find the cycle to separate the new graph (line 9), and the cost on the cycle is bounded by $2\sqrt{2}\sqrt{n}$.

Lemma 2. *A distributed deterministic exact algorithm* **Algorithm 3** *can find the connected components of the n-node planar graph and get the cost of each component.*

Lemma 3. *A distributed deterministic exact algorithm* **Algorithm 5** *can find the the low level l_0 and high level l_2.*

Lemma 4. *A distributed deterministic exact algorithm* **Algorithm 6** *can find the the cycle nodes in the new graph G'.*

The results of Algorithm 3 to Algorithm 6 are concluded in Lemma 2 to Lemma 4. We will show the details of each algorithm we used in the following subsections. Based on Lemma 3 and Lemma 4, we can find that the separator of a planar graph are the combinations of the nodes of l_0 and l_2 found in Lemma 3 and the cycle found in Lemma 4.

A. Find the Connected Components

We will show how to find the connected components by a distributed algorithm **Algorithm** 3 in this subsection. The algorithm is based on distributed BFS algorithm. At the beginning, we give each node a number c_i (line 1). Then for each node, we send messages for computing the minimum number c_i in a BFS tree (lines 2-8). In the end, we count the cost of each BFS tree (lines 9-13).

B. Construct the New Graph G'

In this subsection, we construct the new graph G' by two levels l_0 and l_2. In the BFS tree of G, we define the number of level l for every node u according to the distance from node u to the root node, and $L(l)$ denote the cost in level l. In Algorithm 4 and Algorithm 5, we denote H, sum_i and k as the height of the distributed BFS tree, the cost from level 0 to level l_i and the cost from level 0 to l_1.

To get these two levels, we first find a distributed BFS tree from an source node r_0 for the component which has the maximum cost. Take Figure 2 as an example of two levels l_0 and l_2. Note that level 0 means the source node(it is different from l_0). In **Algorithm** 4, we compute every node's level l_i and the height H of distributed BFS tree (lines 1-14). In lines 15-26, we make the source node r_0 get every level's cost.

Algorithm 3 Find the Connected Components of Planar Graph

```
Input: Graph $G = (V, E)$, every node has an unique id $id_i$
Output: the cost of every component
 1: Run a distributed $BFS$ process, give every node a number $c_i$, and $c_i \leftarrow id_i$
 2: for each node $v_i$ in $V$ do
 3:     run a distributed BFS process at $v_i$
 4:     if $v_i$ receives the message $c_i$ from neighbors $u$ and $c_i \leq min\ c_u$ then
 5:         $c_u \leftarrow c_i$
 6:         send the message $c_u$ to its successors
 7:     end if
 8: end for
 9: for each node $v_i$ in $V$ do
10:     if $id_i == c_i$ then
11:         run a distributed BFS process at $v_i$, count the component's cost
12:     end if
13: end for
```

Algorithm 4 Get the Level Number l_i and Level Cost $L(l_i)$

```
Input: Graph $G = (V, E)$, every node has an unique id $id_i$
Output: every node get its level number $l_i$, the source node $r_0$ get every level's cost $L(l_i)$, the height $H$ of BFS tree
 1: $l_i$ (for all nodes $v_i$) $\leftarrow 0$
 2: for the source node $r_0$ do
 3:     $H \leftarrow 0$, $l_i \leftarrow 0$
 4:     Send the message $H, l_i$ to all successors
 5: end for
 6: for the source node $v_i \neq r_0$ do
 7:     if receive the message $H, l_i$ for the first time(over an edge $e$) then
 8:         $H \leftarrow H + 1$
 9:         $l_i \leftarrow l_i + 1$
10:         send out the message $H, l_i$ to all successors
11:     else if receive the message $H, l_i$ again (over other edges) then
12:         Discard the message $H, l_i$ and do nothing
13:     end if
14: end for
15: all $L(l_i) \leftarrow 0$, $i \leftarrow 0$
16: for every node $id_i$ in the BFS tree do
17:     $L(l_i) \leftarrow 1$, $i \leftarrow H$
18:     for $i \geq 0$ do
19:         send the message $id_i, L(l_i)$ to its parent node
20:         if receive the message $id_j, L(l_j)$ from different child node $id_j$ then
21:             $L(l_i) \leftarrow L(l_i) + L(l_j)$
22:             send the message $id_i, L(l_i)$ to its parent node
23:         end if
24:         $i \leftarrow i - 1$
25:     end for
26: end for
```

In **Algorithm** 5, we utilize the results of Algorithm 4 to find the level l_0 and l_2. We run the Algorithm 5 from source node r_0 to find the level l_1 that the cost does not exceed $n/2$ from level 0 to $l_1 - 1$, but the cost exceeds $n/2$ from level 0 to l_1 (lines 1-11). The highest level $l_0 \leq l_1$ is found in the case that $L(l_0) + 2(l_1 - l_0) \leq 2\sqrt{k}$ (lines 12-19). We find the lowest level $l_2 \geq l_1 + 1$ that $L(l_2) + 2(l_2 - l_1 - l_0) \leq 2\sqrt{n-k}$ (lines 20-27).

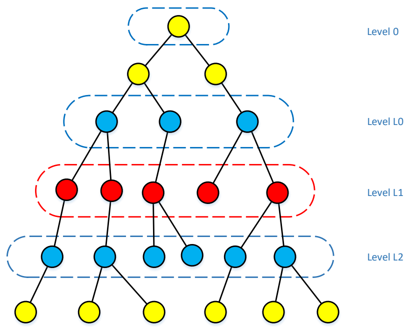

Fig. 2. The level 0, $L0$, $L1$ and $L2$.

Algorithm 5 Find Level l_0 and l_2

Input: Graph $G = (V, E)$, every node get its level number l_i, the source node r_0 get every level's cost $L(l_i)$, the height H of BFS tree

Output: level l_0 and l_2

```
1:  $sum_i \leftarrow 0$, $i \leftarrow 0$
2:  for source nodes $r_0$ in the BFS tree do
3:      for $i \leq H$ do
4:          $sum_i \leftarrow sum_i + L(l_i)$
5:          if $sum_i \geq n/2$ then
6:              $k \leftarrow sum_i$
7:              $l_1 \leftarrow i$
8:          end if
9:          $i \leftarrow i + 1$
10:     end for
11: end for
12: for source nodes $r_0$ in the BFS tree do
13:     $i \leftarrow l_1$
14:     for $i \geq 0$ do
15:         if $L(l_i) + 2(l_1 - l_i) \leq 2\sqrt{k}$ then
16:             $i \leftarrow i - 1$
17:         end if
18:     end for
19:     $l_0 \leftarrow i$
20:     $i \leftarrow l_1$
21:     for $i \leq H$ do
22:         if $L(l_i) + 2(l_i - l_1 - l_0) \leq 2\sqrt{n-k}$ then
23:             $i \leftarrow i + 1$
24:         end if
25:     end for
26:     $l_2 \leftarrow i$
27: end for
```

To get a modified graph G', we delete all nodes on level l_2 and above. We also delete the nodes from level 0 to level l_0. Then we get a new BFS tree T' by setting a new node r_0 to represent the nodes from levels 0 to l_0. This step is concluded in **Lemma** 5.

Lemma 5. *After deleting all nodes from level 0 to level l_0, level l_2 and above, the height h' of T' is $O(D)$.*

Proof. In the BFS tree T of graph G, the height h of T is bounded by $O(D)$. We get the new BFS tree T' from the BFS tree T by deleting some levels of nodes. So the height h' of T' is smaller than h, and $h' = O(D)$. □

C. Find a Cycle to Separate G'

To find a cycle to separate the new graph G', we first compute a distributed BFS tree T_x from a root node x in G'. The cost of every node v in the BFS tree is equal to the sum of all v's descendants' costs plus one (including v itself). For any nontree edge in T_x, the cycle is composed of the nontree edge and the BFS tree path between the notree edge's two nodes. Following this cycle-path, we compute the cost of associated nodes and we know the edges' side of the cycle by the planar embedding through Lemma 1. We define the "inside" means the more cost side, and "outside" means the less one [9]. Take Figure 3 as an example.

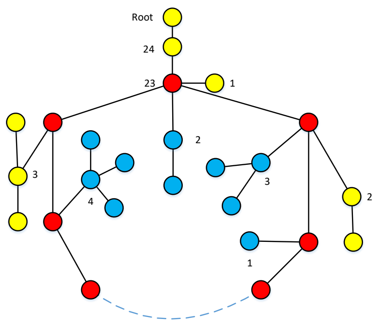

Fig. 3. The cost of nodes in G. The dashed line is the nontree edge. In this figure, the cycle cost is 7, the inside cost is 10, and the outside cost is 8.

Let (v_i, w_i) be an nontree edge. If the cost inside the cycle exceeds $2n/3$, we find a better cycle (cost does not exceed $2n/3$) by the **Algorithm 6**. During the process of finding the cycle, there are 3 cases. See Figure 4.

For case 1, lines 1-11 indicate that one of (v_i, y) and (y, w_i) is a tree edge. Let the node v_{i+1} is one node of the path from v_i to y, and w_{i+1} is one node of the path from w_i to y. Lines 3-11 are computing the cost inside the nontree edge along (v_i, y) and (y, w_i) cycle by aggregating costs of each node along the BFS tree path. Every node v_{i+1} or w_{i+1} computes the cost locally, and compares the cost $cost_v(i+1)$ or $cost_w(i+1)$ to $2n/3$. We take v_{i+1} as an example, w_{i+1} is the same as v_{i+1}. If v_{i+1}'s $cost_v(i+1)$ is bigger than $2n/3$, the node needs to send the message $cost_v(i+1)$ to the node along the BFS tree path. Otherwise, the node v_{i+1} is the node we want to find. Similarly, we can use the same way to find the node w_{i+1}.

Algorithm 6 Find the Cycle for the New Graph G'

```
Input: BFS tree of the graph $G'$, nontree edge $(v_i, w_i)$, node $y$ is in the tree path $(v_i, w_i)$, the cost $cost_i$ of every node $v_i$
Output: the cycle $c$ of the graph $G'$
1:  if $(v_i, y)$ is a tree edge then
2:      for node $v_{i+1}$ along the BFS tree's path $(v_i, y)$ do
3:          if node $v_{i+1}$ receive the message $cost_v(i+1)$ then
4:              compute the inside cost $cost_v(i+1)$
5:              if $cost_v(i+1) > 2n/3$ then
6:                  send the message $cost_v(i+1)$ to the node along the BFS tree path
7:              else the cycle $c \leftarrow$ the nontree edge $(v_{i+1}, w_i)$ and the path $(v_{i+1}, w_i)$ in the BFS tree
8:              end if
9:          end if
10:     end for
11: else
12:     for nodes $v_{i+1}, w_{i+1}$ along the BFS tree's path $(v_i, y)$, $(y, w_i)$ do
13:         if nodes $v_{i+1}, w_{i+1}$ receive the message $cost_v(i+1)$, $cost_w(i+1)$ then
14:             compute the inside cost $cost_v(i+1)$ of $v_{i+1}$ and inside cost $cost_w(i+1)$ of $w_{i+1}$
15:             if $cost_v(i+1) \leq 2n/3$ or $cost_w(i+1) \leq 2n/3$ then
16:                 the cycle $c \leftarrow$ the nontree edge $(v_{i+1}, w_{i+1})$ and the path $(v_{i+1}, w_{i+1})$ in the BFS tree
17:             else send the message to the node along the BFS tree path
18:             end if
19:         end if
20:     end for
21: end if
```

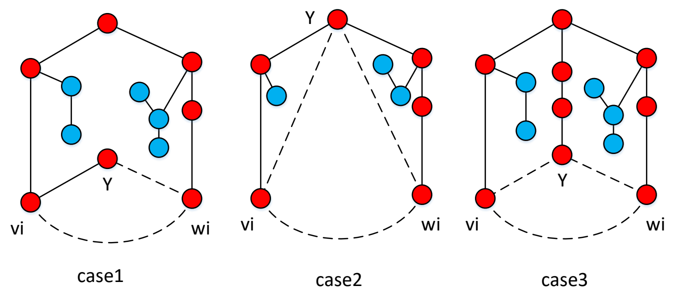

Fig. 4. Three cases 1, 2, 3 exist in Algorithm 6. The (v_i, w_i) is an additional edge. Case 1, (v_i, y) is a tree edge and (y, w_i) is not a tree edge. Case 2, (v_i, y) and (y, w_i) are not tree edges, node y is on the cycle path between v_i and w_i. Case 3, (v_i, y) and (y, w_i) are not tree edges, node y is not on the cycle path between v_i and w_i.

The cycle in case 1 is composed of the BFS tree path and the nontree edge between v_{i+1} and w_{i+1}.

For case 2 and case 3, lines 12-21 indicate that neither (v_i, y) nor (y, w_i) is a tree edge. The node v_{i+1} and w_{i+1} are the same definitions as in case 1. Different from case 1, we scan the cycle path on the BFS tree from node v_{i+1} and w_{i+1} in parallel, then we can aggregate the costs of all nodes on the cycle path. Finally, this process will finish until finding a cycle whose inside cost does not exceed $2n/3$.

VI. Distributed Algorithm for Computing Girth of Planar Graphs

In this section, we give details of **Algorithm** 7. The algorithm has two steps. The first step is to cover the planar graph G by k-outerplanar graphs (lines 1 and 15 of **Algorithm** 7), and the second step is to compute the girth in every k-outerplanar graph in parallel (lines 2-13 and 16-27 of **Algorithm** 7). This distributed algorithm is inspired by Weimann and Yuster's [5] centralized algorithm and the result is concluded in Theorem 3.

Theorem 3. *A distributed exact algorithm **Algorithm** 7 can compute the girth in $O(D \log^2 n)$ rounds for an unweighted planar graph G.*

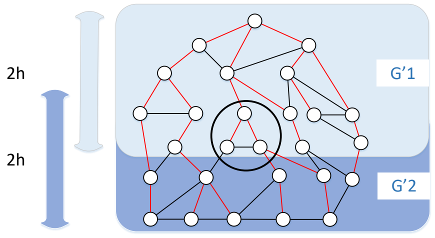

Fig. 5. Using the k-outerplanar graph to decompose the graph G. In this example, $k = 2h$. The girth (black cycle) is guaranteed to be contained in one of these k-outerplanar graphs.

For the first step, we run a distributed planar embedding algorithm of [12] and a distributed BFS of G rooted at a node r_0. We define h as the minimum face's size. And we define H as the BFS tree's height and G_i as the subgraphs induced by the nodes whose distance from x is between hi and $2h + hi$ for $k = 2h$ and $i = 0, 1, 2, 3, \ldots, 2(H - 2h)$. The distance is the same meaning with the level in **Lemma** 3. By this method, every G_i covers with at most two neighbor graphs. A simple example with $k = 2h$ is shown in Figure 5. For $k = 2h$, every graph G_i is a $2h$-outerplanar graph [5]. From **Lemma** 6, we know that the girth is bounded by h and we can chose $k = 2h$ so that the overlap between two neighbors G_i is h.

Lemma 6. *The minimum face's size h is bounded by the diameter D of the unweighted planar graph G which means $h = O(D)$.*

Proof. It is obvious that any face is a cycle in the planar embedding, so the length of minimum face is at most $O(2D) = O(D)$, which means the minimum face's size $h = O(D)$. □

For the second step, we compute the girth in every k-outerplanar graph. Our distributed algorithm is to compute the girth in every k-outerplanar graph in parallel. However, the diameter D'_i of every k-outerplanar graph G_i may be larger than the diameter D of G. And this problem will happen in the process of recursive computing the girth of the subgraphs. So we take advantage of the technical method "shortcuts" of Ghaffari and Haeupler [13] in each k-outerplanar graph (line 4). **Lemma** 7 shows that the shortcut method can reduce the diameter D'_i of each subgraph G_i to $O(D \log D)$ with a distributed algorithm in $O(D \log n)$ rounds. For each shortcut edge, there are $O(D \log D)$ congestion, which means the number of subgraph G_i containning the edges is at most $O(D \log D)$.

Lemma 7 ([13])**.** *For a planar graph G and any partition of its nodes into disjoint parts, there exists a diameter of $O(D \log D)$ shortcut with congestion $O(D \log D)$, where D denotes the diameter of G. And there exists an $O(D \log n)$ rounds distributed algorithm for computing the shortcut.*

To use "shortcuts", the collection of the subsets $G_1, G_2, \ldots, G_j$ must be *node-disjoint*. But in [5], the k-outerplanar graphs are overlapped with their neighbors. So in [5], these node sets are *not node-disjoint*. Hence, we need to do the modification for this problem. We set the k-outerplanar graphs between level hi and $2h + hi$ with $i = 0, 2, 4, 6, \ldots, 2(H - 2h) - 2$ to compute the shortcuts together, and the k-outerplanar graphs between level hi and $2h + hi$ with $i = 1, 3, 5, 7, \ldots, 2(H - 2h)$ to compute the shortcuts together. This step makes the k-outerplanar graphs satisfy the request of *node-disjoint*. Then we can use the shortcuts while executing the distributed algorithm in parallel.

After we construct the shortcut, the algorithm utilizes **Theorem 2** to find the separator (line 5). Then we remove the separator, and use the algorithm recursively to compute the girth on each connected component.

There are two cases when computing girth on each component. For the case that the shortest cycle passes through any node of the separator (lines 6-11), a BFS tree (because this algorithm works on the unweighted planar graphs) should be computed from every separator node. So we run a distributed BFS algorithm from every separator node (line 7). The distributed BFS takes $O(2h+D)$ rounds by constructing the BFS tree from every separator node [21]. The following **Lemma 8** indicates that we can compute the girth through the BFS tree T and the shortest cycle only has one edge not in T. Define $d(u)$ as the distance of T from root node to node u. Hence, $d(u) + d(v) + 1$ means the size of a cycle in the graph G, and the minimum of the sums over all edges $(x, y) \notin T$ is the girth of G.

Lemma 8 ([5])**.** *Let T be a BFS tree started from node v in an unweighted graph. If a node v is in a shortest cycle, then there is a shortest cycle that passes through v and has only one edge not in T.*

Algorithm 7 Compute the Girth for Every k-Outerplanar Graph in Parallel

```
Input: the connected planar graph $G = (V, E)$, the recursion level $RL$, the level number $l_i$ of every node $x$ in the distributed BFS from source node $r_0$
Output: the girth $g$ of the planar graph $G$
1: for every $k$-outerplanar graph $G_i$ whose nodes' level number $l_i$ is between $hi$ and $2h + hi$ with $i = 0, 2, 4, 6, \ldots, 2(H - 2h) - 2$ do the following work in parallel do
2:     $t \leftarrow 0$
3:     while $t \leq RL$ do
4:         compute the shortcuts of $G_i$ by Lemma 7
5:         compute the separator nodes set $X_i$ of $G_i$ by Theorem 2
6:         for every node $x_j$ in the set $X_i$ do
7:             do distributed BFS from $x_j$
8:             every node $u$ in the BFS tree gets the distance $d_u(x_j)$ to node $x_j$
9:             find the node $u$ and $v$ connected by an nontree edge, give the node $u$ and $v$ a label.
10:            get the $g_j = min(d_u(x_j) + d_v(x_j) + 1)$ by convergecast.
11:        end for
12:        $t \leftarrow t + 1$
13:    end while
14: end for
15: for every $k$-outerplanar graph $G'_i$ whose nodes' level number $l_i$ is between $hi$ and $2h + hi$ with $i = 1, 3, 5, 7, \ldots, 2(H-2h)$ do the following work in parallel do
16:     $t \leftarrow 0$
17:     while $t \leq RL$ do
18:         compute the shortcuts of $G_i$ by Lemma 7
19:         compute the separator nodes set $X_i$ of $G_i$ by Theorem 2
20:         for every node $x_j$ in the set $X_i$ do
21:             do distributed BFS from $x_j$
22:             every node $u$ in the BFS tree gets the distance $d_u(x_j)$ to node $x_j$
23:             find the node $u$ and $v$ connected by an nontree edge, give the node $u$ and $v$ a label.
24:             get the $g_j = min(d_u(x_j) + d_v(x_j) + 1)$ by convergecast.
25:         end for
26:         $t \leftarrow t + 1$
27:     end while
28: end for
29: run a distributed BFS from source node $r_0$, find the girth $g = min\ g_j$
```

If the shortest cycle does not pass through any separator node v, for all edges $e \notin T$, the the minimum of the sums is bigger than G's girth, so we do nothing and the algorithm goes to the next loop until the loop indicator $t > RL$ where RL is the recursion level and $RL = O(\log n)$ (the reason is shown in the later section). In the end, we run a parallel algorithm on every separated G_i to find the cycle. The minimum cycle among these g_i is the girth of G.

VII. Algorithm Analyses

A. Correctness Analysis

Lemma 9. *Algorithm 3 satisfies the $\mathcal{CONGEST}$ model.*

Proof. Line 1 just runs a distributed BFS process, and gives every node a number c_i, this message is $O(\log n)$ bits.

Lines 2-8 make every node run the distributed BFS process in parallel, and every node v_i except the root node sends the message only after it receives the message from neighbors u and $c_i < min\ c_u$. This design means the node cannot send the message arbitrarily, and only the node which has the minimum number c_i can send the message. So lines 2-8 can finish in $O(D)$ rounds, and they satisfy the $\mathcal{CONGEST}$ model.

Lines 9-13 mean that we find the node which has the minimum number for every component within a distributed BFS process, and count the component's cost. It is obvious that they satisfy the $\mathcal{CONGEST}$ model.

So the **Algorithm 3** satisfies the $\mathcal{CONGEST}$ model. □

Lemma 10. *Algorithm 4 and Algorithm 5 satisfy the $\mathcal{CONGEST}$ model.*

Proof. In **Algorithm 4**, line 1 means initializing all nodes' level number, the message is $O(\log n)$ bits.

Lines 2-14 run a distributed BFS process, and every node calculates its level l_i and the height H of the BFS tree to its successors in every round.

Lines 15-26 run the process in the distributed BFS tree, every node only send the message $id_i, L(l_i)$ to its parent node. This message is $O(\log n)$ bits.

In **Algorithm 5**, lines 1-11 find the level l_1 that the cost does not exceed $n/2$ from level 0 to $l_1 - 1$, but the cost exceeds $n/2$ from level 0 to l_1. The messages delivered in this process are the cost $L(l_i)$ of level l_i and the cost sum_i from level 0 to l_i, they are $O(\log n)$ bits.

Lines 12-27 find the level l_0 that $L(l_0) + 2(l_1 - l_0) \leq 2\sqrt{k}$ and level l_2 that $L(l_2) + 2(l_2 - l_1 - l_0) \leq 2\sqrt{n-k}$. This process focuses on calculating and comparing by distributed BFS, it satisfies the $\mathcal{CONGEST}$ model.

In conclusion, **Algorithm 4** and **Algorithm 5** satisfy the $\mathcal{CONGEST}$ model. □

Lemma 11. *Algorithm 6 satisfies the $\mathcal{CONGEST}$ model .*

Proof. Lines 1-10 deal with the case that one of (y, w_i) and (v_i, y) is a tree edge. In this case, we need to compare $cost_i$ of node along the BFS tree's path with $2n/3$. The message we need to send is the cost along the path, so it is bounded by $O(\log n)$ bits. The message delivered in lines 11-21 is the same as the case as noted above. Thus, we conclude that **Algorithm 6** satisfies the $\mathcal{CONGEST}$ model. □

Theorem 4. *Algorithm 2 satisfies the $\mathcal{CONGEST}$ model.*

Proof. From lemma 9, lemma 10 and lemma 11, we can prove this theorem. □

Theorem 5. *Algorithm 7 satisfies the $\mathcal{CONGEST}$ model .*

Proof. Line 1 is a distributed BFS process to divide the graph G into k-outerplanar graphs. The messages is $O(\log n)$ bits.

Line 2 is initializing the recursive number t to 0.

Lines 3-13 execute a process for computing the girth in every k-outerplanar graph recursively. Line 4 is to compute the shortcuts through Lemma 7 with congestion $O(D \log D)$, this means the message need $O(D \log D)$ rounds to deliver for satisfying the $\mathcal{CONGEST}$ model. Line 5 is to utilize Algorithm 2, from Theorem 4, this process satisfies the $\mathcal{CONGEST}$ model. Lines 6-11 do a distributed BFS process for computing the girth. The messages delivered are the label, distance $d_u(x_j)$ and g_j, they can be packed in $O(\log n)$ bits. Line 12 is to keep the recursive process by making k plus one, it satisfies the $\mathcal{CONGEST}$ model.

Lines 15-28 are the same as lines 1-14, so they satisfy the $\mathcal{CONGEST}$ model.

Line 29 is a distributed BFS process, the message is g_j and can be packed in $O(\log n)$ bits.

Thus, we conclude that **Algorithm 7** satisfies the $\mathcal{CONGEST}$ model. □

B. Efficiency Analysis

Lemma 12. *Algorithm 3 takes $O(D)$ rounds.*

Proof. In **Algorithm 3**, line 1 is a distributed BFS process so that it takes $O(D)$ rounds. Lines 2-8 take $O(D)$ rounds. Because only one node which has minimum number can send the message in every round, $T_{total} = T_{min} = O(D)$. Lines 9-13 take $O(D)$ rounds, because they run a distributed BFS and count the numbers of nodes in the component. Thus, we conclude that **Algorithm 3** takes $O(D)$ rounds. □

Lemma 13. *Algorithm 4 and Algorithm 5 take $O(D)$ rounds.*

Proof. In **Algorithm 4**, line 1 is an initializing process so that it takes $O(1)$ rounds. Lines 2-14 take $O(D)$ rounds. Because they run a distributed BFS. Line 15 takes $O(1)$ rounds obviously. Lines 16-26 take $O(D)$ rounds because every node only send the message to its parent in the BFS tree. So they take $O(D)$ rounds for the source node get every level's cost $L(l_i)$. Thus, we conclude that **Algorithm 4** takes $O(D)$ rounds.

In **Algorithm 5**, line 1 is a distributed BFS process so that it takes $O(D)$ rounds. Lines 2-26 take $O(D)$ rounds. Because they run a distributed BFS from source node r_0 and compute the level l_0 and l_2. Thus, we conclude that **Algorithm 5** takes $O(D)$ rounds. □

Lemma 14. *Algorithm 6 takes $O(D)$ rounds.*

Proof. In **Algorithm 6**, lines 1-10 take $O(D)$ rounds. Because it compares the cost of nodes along the BFS tree path in turn. And the path' length is at most $2D + 1$ in the BFS tree. So it takes $O(D)$ rounds. Lines 11-21 take $O(D)$ rounds because the process executes on the two cycles in parallel, and every cycle takes $O(D)$ rounds. So lines 11-21 take $O(D)$ rounds. So **Algorithm 6** takes $O(D)$ rounds. □

Theorem 6. *Our distributed algorithm Algorithm 2 for planar separator theorem takes $O(D \cdot min\{D, \log n\})$ rounds.*

Proof. The first step is finding a planar embedding of graph G, it will cost $O(D \cdot min\{D, \log n\})$ rounds. And **Algorithms 3, 4, 5, 6** take $O(D)$ rounds. So the total rounds of the distributed algorithm for separator theorem are $O(D \cdot min\{D, \log n\}) + O(D) = O(D \cdot min\{D, \log n\})$ rounds. And this completes the proof of **Theorem 2**. □

Theorem 7. *Our distributed algorithm Algorithm 7 takes $O(D \log^2 n)$ rounds to compute the girth for an unweighted planar graph G.*

Proof. Line 1 means we run the distributed algorithm in parallel for every k-outerplanar graph between level hi and $2h + hi$ with $i = 0, 2, 4, 6, \ldots, 2(H - 2h) - 2$. We run the distributed BFS to divide the graph into k-outerplanar graphs. It takes $O(D)$ rounds.

Lines 2-11 give the recursive algorithm to compute the girth in every k-puterplanar graph.

Line 3 means we run the algorithm bounded by the recursion level RL, and $RL = O(\log n)$ because through the separator theorem, the subsets' size of the graph decreases by a $2/3$ factor in every recursive process.

Line 4 computes shortcuts of G_i, this procedure takes $O(D \log n)$ rounds, it makes the graph G_i has diameter $O(D \log D)$ with congestion $O(D \log D)$[13].

Line 5 computes the separator nodes of G_i, as we have mentioned in section V, it takes $O(D \log n)$ rounds.

Lines 6-11 compute the girth in the G_i through distributed BFS for every separator node in parallel, as mentioned above, it can finish in $O(2h + D) = O(D + D \log D) = O(D \log D)$ rounds because the diameter is $O(D \log D)$ in G_i, and the h is bounded by $O(D)$ through Lemma 6.

So lines 1-14 take $O((D + D \log n + D \log D + D \log n + D \log D) \log n) = O(D \log^2 n)$.

Lines 15-28 run the similar procedure, so lines 15-28 take $O(D \log^2 n)$ rounds as the same as lines 1-14.

Line 29 runs a distributed BFS, it can finish in $O(D)$ rounds.

So lines 1-29 take $O(D \log^2 n)$ rounds, which means Algorithm 7 takes $O(D \log^2 n)$ rounds and this completes the proof of **Theorem 3**. □

VIII. Lower Bound

For computing the girth exactly in the weighted planar graph, a trivial method is to collect all the edges into one node. This method takes $O(n)$ rounds and there does not exist a faster algorithm now. Unfortunately, the algorithm provided in this paper is hard to be applied to the weighted graph in a sublinear time. The reason is that our algorithm is heavily relied on the single source shortest path algorithm, which is tricky to be applied in a sublinear time. In this section, we prove that computing the girth in the weighted planar graph is hard by providing a lower bound construction and showing that computing the weighted girth exactly in planar graphs needs $\Omega(\sqrt{n}/\log n + D)$ rounds. The construction is inspired by [13] and the proving method is based on the mailing problem [22].

A. The mailing problem

For graph $G = (V, E)$ with two distinguished nodes sender s and receiver r, both the sender and receiver store b boolean variables each, $\lambda_1^s, \cdots, \lambda_b^s$ and $\lambda_1^r, \cdots, \lambda_b^r$, respectively. The mailing problem $Mail(G, s, r, b)$ takes s to deliver b variables to receiver r. In this section, we consider this problem on a weighted planar graph G with $b = \sqrt{n}$. The construction of our graph is different from the graph in [22], we will prove that based on this construction, the lower bound is $\Omega(\sqrt{n}/\log n + D)$.

B. The construction of the lower bound

The lower bound graph is constructed as Figure 6: There is a long path P consisted of xy nodes $P_1, \cdots, P_{xy}$, the edges between them have weight 1. The left half nodes of the path connect to node s and the right half are linked to node t, the weight of these edges are n^2. Then we add subgraphs to this construction. The subgraph consists of x levels of paths $path$ that each $path$ consists of xy nodes as shown in Figure 7. For every y node, we add edges of weight n^2 connecting each level. We also add y edges of weight n^2 connecting the subgraph to the long path P. There are total y subgraphs added in the construction level by level and no edges intersect. There are xy $paths$ and we denote them as $path_1, \cdots, path_{xy}$.

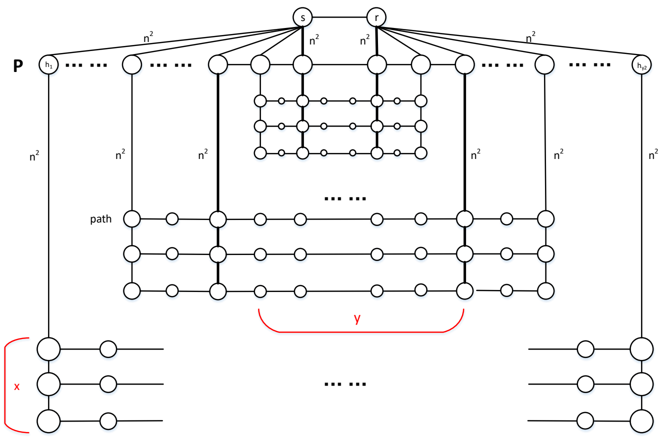

Fig. 6. This picture depicts the construction of the lower bound. Path P means the long path which consists of xy nodes. There are y subgraphs connected to path P, we omit some details in the picture.

We then construct the input variables $\lambda = \{0, 1\}^{xy}$ for sender s and receiver r. For each subgraph, we set the weight of the most left edge of each adjacent paths as W or W' where $W' > W$. The weight W corresponds to boolean variable 1 and W' corresponds to 0. Hence, the input variables $\lambda = \{0, 1\}^{xy}$ of s is determined by weight assignment of each subgraph. The construction for receiver r is the same as the left side. A simple depiction of each subgraph is shown in Figure 7. For this construction, we give **Lemma** 15 showing the relationship of hop diameter D and the number of nodes n.

Lemma 15. *The total number of nodes $n = x^2y^2 + xy + 2$ and the hop diameter D of our construction is at least $n^{1/4}$.*

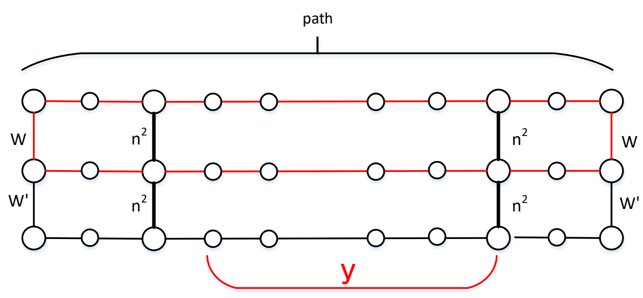

Fig. 7. An example of the subgraph. A subgraph consists of x paths, each path has xy nodes and there are $x+1$ edges connecting each pair of adjacent paths. The weights of the most left and right edges between paths are determined by input boolean variables λ.

Proof. From our construction, the total number of nodes is $x^2y^2 + xy + 2$ so that $xy = \Theta(n^{1/2})$. For any pair of nodes, the hop between them is at most $max\{2x + y/2, y\}$. Since $y \le D$, we can get $x \le D/4$. Thus, the hop diameter satisfies $D = \Omega(n^{1/4})$. □

There are many cycles in this construction. In **Lemma** 16, we show that the weighted girth is only determined by input variables $\lambda = \{0, 1\}^{xy}$ of s and r.

Lemma 16. *The weighted girth in the construction is* $2W + 2xy - 2$ *or* $2W' + 2xy - 2$.

Proof. We first consider cycles formed in subgraphs. For all the edges between paths except the edges of the most left and right, they can form cycles with weight of at least $2n^2 + 2y$. Since the weights of the most left and the right edges is W or W', we get the cycle of weight of $2W + 2xy - 2$ or $2W' + 2xy-2$, which is the minimal weighted cycle in the subgraphs.

We then consider cycles formed by subgraphs, long path P and nodes s and r. The minimum weighted cycle is formed by s, r and an edge in the long path, which is $2n^2 + 1$. Thus, we conclude that the weighted girth in the construction is $2W + 2xy - 2$ or $2W' + 2xy - 2$. □

C. The lower bound

From **Lemma** 16, we know that the weighted girth in this construction can be computed only when receiver r receives the boolean variables $\lambda_1^s, \cdots, \lambda_{xy}^s$ from s. Otherwise we cannot distinguish the minimum weight between $2W+2xy-2$ and $2W' + 2xy - 2$. A natural way to deliver these boolean variables is routing these bits along paths of each subgraph. But there is a shorter path through edge (s, r) so that a delivery algorithm may deliver boolean bits in a way mixed the shorter path and $paths$. To prove the lower bound for deliver xy-bits boolean variables, we take the idea from a classic problem $Mail(G, s, r, xy)$. We conclude the result of our lower bound in Theorem 8.

Theorem 8. *For a weighted planar graph* $G = (V, E)$, *any distributed algorithm needs* $\Omega(\sqrt{n}/\log n + D)$ *rounds to compute weighted girth in the* $\mathcal{CONGEST}$ *model.*

Before proving the theorem, we give the definition of T_t, which is an important ingredient of the proof. For every $1 \le t \le xy$, the tail of long path P^t is defined as

$$T_t(P^t) = \{P_i | t \le i \le xy\}.$$

Denote v_i^j as the j-th node in $path_i$, then we define the tail of $path^t$ as

$$T_t(path^t) = \{v_i^j | 1 \le i \le xy, t \le j \le xy\}.$$

Then the definition of T_t is

$$T_t = T_t(P^t) \cup T_t(path^t) \cup r.$$

We give Figure 8 for T_{t+1}.

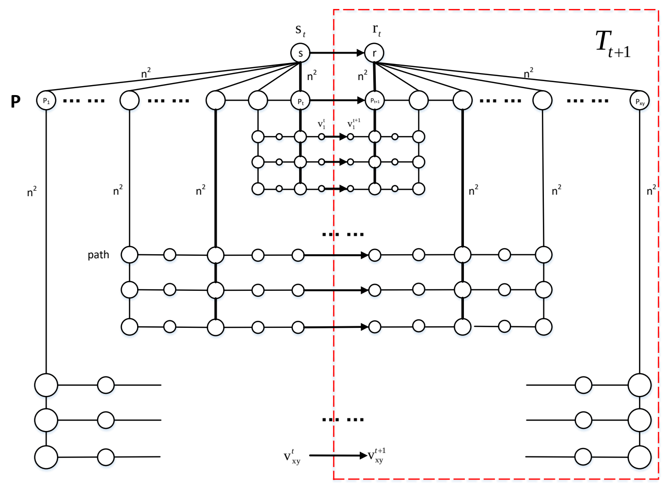

Fig. 8. An example of T_{t+1}, all the nodes contained in the red box are belonged to T_{t+1}. The arrows represent possible bits delivered from T_t to T_{t+1}.

Our proof is based on $Mail(G, s, r, xy)$ problem [22], so here we also define the execution state of v at round t as $\varphi(v, t, \lambda)$. Then for the set of nodes $U = \{v_1, \cdots, v_l\}$, a configuration

$$C(U, t, \lambda) = \{\varphi(v_1, t, \lambda), \ldots, \varphi(v_l, t, \lambda)\}$$

is a vector of the execution states of the nodes of U at round t. Denote $\mathcal{C}[U, t]$ as the collection of all possible configurations of set U at round t. Define $\rho(U, t) = |\mathcal{C}[U, t]|$. At the beginning of the execution, we need $x + 2$ rounds for s to know all the boolean variables λ. After node s has received all these variables, the execution starts and $t = 0$ in this round. Thus, $\rho(T_0, 0) = 1$ in the initial state.

Proof of Theorem 8. We prove the lower bound through showing that $\rho(T_t, t) \le (2^{\log n} + 1)\rho(T_{t+1}, t + 1)$ in rounds $0 \le t < xy$. From Figure 8, we know that for a fixed configuration $\hat{C} \in \mathcal{C}(T_t, t)$, the state in each v_i^t is a single state so that each v_i^t sends only one state over the edge (v_i^t, v_i^{t+1}). But for the edge (s, r), the node s is not in the set T_t so that it may be in any one of the possible states. Since each edge can deliver $O(\log n)$-bits messages, at most

$2^{\log n}+1$ states will be observed by r. Thus, we conclude that $\rho(T_t, t) \leq (2^{\log n}+1)\rho(T_{t+1}, t+1)$ in rounds $0 \leq t < xy$.

Then we prove that computing weighted girth in planar graphs needs $\Omega(\sqrt{n}/\log n + D)$ rounds. From $\rho(T_t, t) \leq (2^{\log n}+1)\rho(T_{t+1}, t+1)$, we have $\rho(T_t, t) \leq (2^{\log n}+1)^t$ for $0 \leq t < xy$. At the end of the execution, r will receive at least 2^{xy} different states so that $\rho(T_{t_{end}}, t_{end}) \geq 2^{xy}$. Combined with **Lemma** 15, we have $t_{end} \geq \sqrt{n}/\log n$. From **Lemma** 16, we know that r can distinguish weighted girth between $2W + 2xy - 2$ and $2W' + 2xy - 2$ if it received all boolean variables from s. Thus, we conclude that the weighted girth can be computed in planar graphs in $\Omega(\sqrt{n}/\log n + D)$ rounds. □

IX. Conclusion

In this paper, we present the first sublinear time $O(D \cdot min\{D, \log n\})$ distributed deterministic exact algorithm for planar separator theorem. We consider this algorithm as an useful method in designing the efficient distributed algorithms for planar graphs. Then we utilize the separator theorem to design the first distributed exact algorithm which computes the girth for the unweighted planar graphs in $O(D\log^2 n)$ rounds. Finally, we prove an $\Omega(\sqrt{n}/\log n + D)$ lower bound for computing the girth exactly of a weighted planar graph. All of our algorithms work under the $\mathcal{CONGEST}$ model. It would be interesting to utilize the distributed algorithm for separator theorem to slove the maximum matching, maximum independent set or other problems for planar graphs.

References

[1] Reinhard Diestel. *Graph theory*. Springer, Heidelberg, 2nd edition, 2000.

[2] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978.

[3] Hristo Djidjev. Computing the girth of a planar graph. In *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000, Geneva, Switzerland, July 9-15, 2000, Proceedings*, pages 821–831, 2000.

[4] Parinya Chalermsook, Jittat Fakcharoenphol, and Danupon Nanongkai. A deterministic near-linear time algorithm for finding minimum cuts in planar graphs. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 828–829, 2004.

[5] Oren Weimann and Raphael Yuster. Computing the girth of a planar graph in $O(n \log n)$ time. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, pages 764–773, 2009.

[6] Hsien-Chih Chang and Hsueh-I Lu. Computing the girth of a planar graph in linear time. In *Computing and Combinatorics - 17th Annual International Conference, COCOON 2011, Dallas, TX, USA, August 14-16, 2011. Proceedings*, pages 225–236, 2011.

[7] Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980.

[8] Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, 1987.

[9] Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics 36.2 (1979): 177-189.*, 1979.

[10] Noga Alon, Paul D. Seymour, and Robin Thomas. Planar separators. *SIAM J. Discrete Math.*, 7(2):184–193, 1994.

[11] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 363–372, 2011.

[12] Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks I: planar embedding. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 29–38, 2016.

[13] Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks II: low-congestion shortcuts, mst, and min-cut. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 202–219, 2016.

[14] Peter Ungar. A theorem on planar graphs. *Journal of the London Mathematical Society 1.4 (1951): 256-262.*, 1951.

[15] Hristo Nicolov Djidjev. On the problem of partitioning planar graphs. *SIAM Journal on Algebraic Discrete Methods 3.2 (1982): 229-240.*, 1982.

[16] John R. Gilbert, Joan P. Hutchinson, and Robert Endre Tarjan. A separator theorem for graphs of bounded genus. *J. Algorithms*, 5(3):391–407, 1984.

[17] Jacob Fox and János Pach. A separator theorem for string graphs and its applications. *Combinatorics, Probability & Computing*, 19(3):371–390, 2010.

[18] Jacob Fox and Jnos Pach. Separator theorems and turn-type results for planar intersection graphs. *Advances in Mathematics 219.3 (2008): 1070-1080.*, 2008.

[19] Paul Seymour Alon, Noga and Robin Thomas. A separator theorem for nonplanar graphs. *Journal of the American Mathematical Society 3.4 (1990): 801-808.*, 1990.

[20] David Peleg. Distributed computing: a locality-sensitive approach. *Society for Industrial and Applied Mathematics*, 2000.

[21] Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *ACM Symposium on Principles of Distributed Computing, PODC '12, Funchal, Madeira, Portugal, July 16-18, 2012*, pages 355–364, 2012.

[22] David Peleg and Vitaly Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM J. Comput.*, 30(5):1427–1442, 2000.