Fully Polynomial-Time Distributed Computation in Low-Treewidth Graphs

Taisuke Izumi

Graduate School of Information Science and Technology, Osaka University
Osaka, Japan

t-izumi@ist.osaka-u.ac.jp

Naoki Kitamura†

Graduate School of Information Science and Technology, Osaka University
Osaka, Japan

n-kitamura@ist.osaka-u.ac.jp

Takamasa Naruse

Graduate School of Engineering, Nagoya Institute of Technology
Aichi, Japan

t.naruse.333@stn.nitech.ac.jp

Gregory Schwartzman†

Japan Advanced Institute of Science and Technology (JAIST)
Ishikawa, Japan

greg@jaist.ac.jp

ABSTRACT

We consider global problems, i.e. problems that take at least diameter time, even when the bandwidth is not restricted. We show that all problems considered admit efficient solutions in low-treewidth graphs. By “efficient” we mean that the running time has polynomial dependence on the treewidth, a linear dependence on the diameter (which is unavoidable), and only a polylogarithmic dependence on n, the number of nodes in the graph. We present the following results in the CONGEST model (where τ and D denote the treewidth and diameter of the graph, respectively):

• Exact single-source shortest paths (Actually, the more general problem of computing a distance labeling scheme) for weighted and directed graphs can be computed in \(O(\tau^2 D + \tau^5)\) rounds. This is the first exact algorithm for the directed single-source shortest paths problem in low-treewidth graphs attaining a \(O(\tau^{O(1)} D)\)-round running time.

• Exact bipartite unweighted maximum matching can be computed in \(O(\tau^4 D + \tau^7)\) rounds. This is the first algorithm for a non-trivial graph class that achieves a worst case running time sublinear in the input size.

• The weighted girth can be computed in \(O(\tau^2 D + \tau^5)\) rounds for both directed and undirected graphs. Our results are the first to imply an exponential separation between the complexity of computing girth and diameter for a non-trivial graph class.

Although the above problems are seemingly unrelated, we derive all of our results using a single unified framework. Our framework consists of two novel technical ingredients. The first is a fully polynomial-time distributed tree decomposition algorithm, which outputs a decomposition of width \(O(\tau^2 \log n)\) in \(O(\tau^2 D + \tau^3)\) rounds (where \(n\) is the number of nodes in the graph). The second ingredient, and the technical highlight of this paper, is the novel concept of a stateful walk constraint, which naturally defines a set of feasible walks in the input graph based on their local properties (e.g., augmenting paths). Given a stateful walk constraint, the constrained version of the shortest paths problem (or distance labeling) requires the algorithm to output the shortest constrained walk (or its distance) for a given source and sink vertex. We show that this problem can be efficiently solved in the CONGEST model by reducing it to an unconstrained version of the problem.

CCS CONCEPTS:

• Theory of computation → Distributed algorithms; • Mathematics of computing → Graph algorithms.

KEYWORDS

distributed algorithm, treewidth, shortest path, matching

ACM Reference Format:

Taisuke Izumi, Naoki Kitamura, Takamasa Naruse, and Gregory Schwartzman. 2022. Fully Polynomial-Time Distributed Computation in Low-Treewidth Graphs. In Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA ’22), July 11–14, 2022, Philadelphia, PA, USA. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3490148.3538590

1 INTRODUCTION

1.1 Background and Motivation

The treewidth is one of the most important graph parameters and has received a huge amount of attention in the context of centralized algorithms [12]. Informally speaking, it represents the graph’s similarity to a tree (e.g., a tree has treewidth 1, a cycle has treewidth 2, a clique on \(n\) nodes has treewidth \(n-1\)). In the context of centralized algorithms, a vast class of computationally hard problems is efficiently solvable in low-treewidth graphs. Furthermore, many real-world data sets are indeed low-treewidth graphs [46].
In this work, we focus on the CONGEST model of distributed computation (see Section 2.1 for a formal definition). The inherent bandwidth limitation in the CONGEST model precludes any efficient centralized solution by aggregating the entire topological information of the network, and thus our algorithms must make do with only local information. The usefulness of the treewidth parameter in the CONGEST is mostly due to the recent framework of low-congestion shortcuts \([24, 29, 31]\), which provides efficient group communication for a collection of subgraphs \([24]\). Based on this framework, several algorithms have achieved near-optimal running times for various fundamental problems in low-treewidth graphs \([23, 24, 29, 31]\). For example, minimum-spanning tree, minimum-cut approximation, and approximate undirected single-source shortest paths \([24, 29, 30, 50]\).

While the low-congestion shortcut framework is a valuable tool for designing CONGEST algorithms (and is also used in this paper), it is a general framework, not limited to any specific graph class. Thus, it leaves many intriguing open questions for the family of low-treewidth graphs. For example, the problems of computing an efficient tree decomposition and designing efficient algorithms for fundamental problems based on tree decomposition make explicit use of the structure of low-treewidth graphs. Currently, the only relevant result is due to Li \([43]\), which presents a CONGEST algorithm with a running time of \(O(t^2D)\) that computes a tree decomposition of width \(O(t)\) with applications to the distributed computation of optimal solutions for classic NP-hard problems (e.g., vertex cover) whose running time exponentially depends only on the width of the computed decomposition.

### 1.2 Our Results

We focus on the study of fully polynomial-time distributed computation in low-treewidth graphs. Where "fully polynomial-time" means that the running time of algorithms depends polynomially on the treewidth of the input graph, linearly on its unweighted diameter \(D\) and only has polylogarithmic dependence on \(n\). This can be seen as a distributed analogue of the recent work of \([16]\), which considers problems whose non-parametrized complexity has a super-linear dependence on the input size and presents algorithms whose running time depends polynomially on the treewidth and only linearly on the input size. As most of the problems which admit an improved running time for low-treewidth graphs are global problems, they admit the universal lower bound of \(\Omega(D)\) rounds in the distributed setting. Where the term "universal" means that the lower bound holds for any instance. Hence our analog of linear dependence on the input size as a linear dependence on \(D\) is very natural.

Our results are not the first to achieve a fully polynomial-time dependence on the treewidth. Specifically, the shortcut-based MST and approximate min-cut algorithms mentioned in Sec. 1.1 require \(\tilde{O}(\sqrt{n} + D)\) rounds, which beats the \(\tilde{O}(\sqrt{n} + D)\)-round lower bound for general graphs \([14]\). However, due to the general nature of the shortcut framework, it does not take full advantage of the structure of specific graph classes. This is exemplified in the recent work of \([26, 44, 48]\) on planar graphs. While planar graphs admit efficient shortcut-based algorithms \([23, 24]\), it is possible to achieve improved results and tackle new problems by leveraging techniques that are specific to planar graphs. Our research can be seen as a low-treewidth counterpart of these results. The main contribution of this paper is a single algorithmic framework from which we are able to derive fully polynomial-time algorithms in low-treewidth graphs for a set of seemingly unrelated problems. In what follows, we explain the details of our results.

**Distance Labeling and Single-Source Shortest Paths.** Distance labeling (DL) is the problem of assigning vertices with short labels such that it is possible to compute the distance from \(u\) to \(v\) only by using their labels. The standard single-source shortest paths problem (SSSP) is easily reduced to distance labeling: the source node simply distributes its label to all other nodes. We present a randomized algorithm for exact directed DL, which correctly constructs all of the labels in \(\tilde{O}(t^2 + r^2)\) rounds with high probability (whp)\(^2\). It is known that the undirected weighted shortest-path problem require \(\Omega(\sqrt{n} + D)\) rounds \([14]\) for general graphs, even for approximate solutions. The first improvement of this bound for low-treewidth graphs is due to Haeupler and Li \([30]\). Their algorithm applies to any undirected graph that admits good shortcuts, including low-treewidth graphs (the running time and approximation factor depend on the quality of the shortcut). However, the approximation factor achieved is super-constant and their results do not extend to directed graphs. Concurrently and independently of our work, the approximation ratio and running time of \([30]\) was recently improved to \((1 + \epsilon)\) and \(\tilde{O}(tDn^{o(1)})\) \([50]\). However, their results do not apply for exact distance computation nor to directed graphs.

**Exact Maximum Matching.** We present a randomized algorithm that computes exact unweighted maximum matching in bipartite graphs running in \(\tilde{O}(t^2 + r^2)\) rounds. While the maximum matching problem has received much attention in the context of distributed approximation \([4, 6, 39, 45]\), the complexity of finding an exact solution is yet unknown. For general graphs, \([8]\) were the first to present a non-trivial algorithm with a running time of \(O(s_{\text{max}}^2)\) rounds, where \(s_{\text{max}}\) is the size of the maximum matching. This was recently improved to \(O(s_{\text{max}}^{3/2})\) \([38]\). For the case of bipartite graphs, an algorithm by Ahmadi et al. \([4]\) is the only result for exact maximum matching. It achieves a running time of \(O(s_{\text{max}})\) rounds (and thus the worst-case bound is \(O(n)\), even in low-treewidth graphs). We present the first algorithm for a non-trivial graph class which achieves a running time sublinear in \(n\).

**Weighted Girth.** We present a randomized algorithm that computes the weighted girth, \(g\), of a directed or undirected input graph in \(\tilde{O}(t^2 + r^2)\) rounds with high probability. The best known upper bound for computing \(g\) in general graphs is \(O(\min\{gn^{1/\theta(1/\theta), n}\})\) rounds \([10]\), and a lower bound of \(\tilde{O}(\sqrt{n} + D)\) rounds is also known for unweighted and undirected graphs \([19]\). The lower bound holds even for a \((2 - \epsilon)\)-approximation of \(g\). Planar graphs admit an efficient solution for computing \(g\) in \(\tilde{O}(D^2)\) rounds \([44]\). Our approach is structural and is based on a very simple (randomized) reduction of girth computation to a distance labeling scheme. Our techniques

---

\(^2\)Throughout this paper, the term "with high probability" means that the probability is at least \(1 - 1/nc^2\) for an arbitrary large constant \(c > 0\).
are novel and we believe that they may be applicable to other graph classes.

Our result is the first separation between the complexity of diameter computation and girth computation in undirected and unweighted graphs. All previously known results exhibit similar complexity bounds for both problems, i.e., $\tilde{O}(n^{\Theta(1)})$-round lower bounds for general graphs, and $\tilde{O}(D^{\Theta(1)})$-round upper bounds for planar graphs [44, 48]. On the other hand, there exist hard instances of constant diameter and logarithmic treewidth for which computing the diameter in the CONGEST model requires $O(n)$ rounds [1], contrasted with our algorithm for computing girth. That is, graphs of logarithmic treewidth and diameter are the first non-trivial graph class that exhibits an exponential separation in the round complexity for these two fundamental problems.

1.3 Our Framework

All of our algorithms are a direct result of a single unified framework. The key technical ingredients of our framework are twofold: A new fully polynomial-time tree-decomposition algorithm, and the novel concept of a stateful walk constraint. In this subsection, we outline their ideas and their applications for our algorithms.

Fully Polynomial-Time Tree Decomposition. All of our results require the existence of an efficient tree decomposition algorithm with a small width. Unfortunately, the best known tree-decomposition algorithm [43] has a running time that exponentially depends on the treewidth, and thus is too slow for our needs. Thus, we develop a fully polynomial-time CONGEST algorithm for tree decomposition, which runs in $\tilde{O}(r^2D + r^3)$ rounds and computes a tree decomposition of width $\tilde{O}(r^2)$. The algorithmic ideas are based on the fully-polynomial time (centralized) tree decomposition algorithm by Fomin et al. [16], with several nontrivial modifications that allow for an efficient implementation in the CONGEST model. While a direct implementation of [16] in the CONGEST model is straightforward, this will result in a round complexity of $\tilde{O}(r^{O(1)}D)$, where the exponent of $r$ is (at least) 7. We introduce novel ideas which allow us to substantially improve the dependence on $r$. Our distance labeling result is obtained by combining this tree decomposition algorithm with several techniques by Li and Parter [44] which were introduced in the context of distance labeling for planar graphs.

Stateful Walk Constraint. The second ingredient of our framework is to extend the applicability of distributed directed shortest paths algorithms (including distance labeling schemes) to a more general type of shortest walks. We consider a constrained version of SSSP (or DL), where a subset $C$ of all walks in the input graph is given. This problem requires that each edge $e$ in $G$ knows the length of the shortest walk in $C$ from a source vertex, $s$, to $e$ (or construct a labeling scheme that allows computing the length of the shortest walk in $C$ connecting the two vertices, using only their labels). This problem is not meaningful if $C$ is explicitly given to each node, and our focus is the scenario where $C$ is given in an implicit and distributed manner.

We introduce a natural class of walk constraints, which we call stateful walk constraints. Roughly speaking, a stateful walk constraint is a set $C$, of walks such that each node $u$ can locally decide if a walk leaving $u$ is contained in $C$ or not, using only a small amount of additional information (referred to as the state of a walk). This class captures many natural walks with combinatorial constraints, such as alternating walks (used in our matching algorithm). We show that the constrained versions of directed SSSP and DL under a stateful-walk constraint can be reduced to the corresponding unconstrained versions with a running-time overhead depending on the size of the state space associated with $C$.

Let us outline how to apply the above framework to the problems of matching and girth. We show that each of these problems can be reduced to finding shortest walks under some stateful-walk constraint. Our maximum matching algorithm, i.e., alternating path finding is one of the most natural applications of the framework. By combining the stateful-walk framework with a specific property of augmenting paths in low-treewidth graphs (presented in [35]), we derive our algorithm. For girth computation, our key idea is to use the framework to exclude walks that “fold onto themselves”, that is, the second half of the walk is the inversion of the first half. This leaves us with a set of walks which upper bound the girth. Finally, we use a probabilistic sampling of edge labels combined with the above to derive our algorithm.

The above applications demonstrate the expressive power and versatility of our framework. Finally, we would like to emphasize that the framework is not limited to low-treewidth graphs, but applies to general graphs. The authors believe that this framework is potentially useful in the design of efficient CONGEST algorithms for a wider class of problems.

1.4 Related Work

Distance computation problems are at the core of distributed graph algorithms. Recently there has been a vast number of results for both exact and approximate distance computation problems [1, 2, 7, 9–11, 13, 15, 17, 19, 25, 27, 30, 32–34, 36, 40, 42, 47, 49]. The state-of-the-art bounds for general graphs are $\tilde{O}(\sqrt{D} + D)$ rounds for $(1 + \epsilon)$-approximate SSSP [7], and $\tilde{O}(\sqrt{D^{1/4} + D})$ rounds for exact SSSP [11]. These results hold for weighted and directed graphs. A tight runtime bound for exact SSSP is still an open problem. For computing the girth, a near optimal approximation algorithm for unweighted girth is known [49]. It outputs the girth with an additive error of one (i.e., $(2 - 1/\Theta(g))$-multiplicative approximation) in $O(\sqrt{n}\log n + D)$ rounds.

Planar graphs are also an intriguing class of graphs, and are closely related to our results. Although our algorithms are applicable to planar graphs, as planar graphs have treewidth of $O(D)$, for distance computation problems the existing algorithms tailored for planar graphs [44, 48] achieve a running time with a better dependence on $D$.

Distance labeling schemes were first proposed by Gavoille et al. [22], and their centralized construction was studied for many graph classes [18, 20–22, 37]. Distance labeling is closely related to (approximate) distance oracles, which are centralized data structures for representing distance matrices that support quick access. There are a few results that consider distributed variants of approximate distance oracles [13, 36, 40, 41] for general graphs. However, all of them consider only approximate oracles, and essentially require the construction time to polynomially depend on $n$. The best bound for
bounded treewidth graphs is the shortcut-based construction [31], whose approximation factor is polylog(n).

While most previous results for maximum matching focus on approximate solutions [4, 6, 39, 45], the problem of finding an exact solution has been receiving more and more attention recently [3, 8, 38]. On the negative side, the lower bound of $\tilde{\Omega}(\sqrt{n}D)$ rounds is implied from the result of [4]. This lower bound is recognized as a strong barrier: It has been shown that the well-known approach of reduction from two-party communication complexity does not work for obtaining any stronger lower bound [5]. It is also known that exact maximum matching does not have any local solution: There exists a hard instance where $D = \Theta(n)$ that exhibits an $\Omega(n)$-round lower bound [8].

1.5 Organization of the Paper

In Section 2 we introduce the concept of tree decomposition and the CONGEST model. In Section 3, we present our tree decomposition algorithm in the CONGEST model. In Section 4 we show how to solve the distance labeling problem by using our tree decomposition algorithm. In Section 5 we introduce the concept of stateful walks, and show how to reduce the problem of finding (shortest) stateful walks into the standard directed reachability or the shortest paths problem. Sections 6 and 7 are devoted to the applications of our algorithm. In Section 5 we introduce the concept of stateful walks, and show how to reduce the problem of finding (shortest) stateful walks into the standard directed reachability or the shortest paths problem. Sections 6 and 7 are devoted to the applications of our algorithm. In Section 5 we introduce the concept of stateful walks, and show how to reduce the problem of finding (shortest) stateful walks into the standard directed reachability or the shortest paths problem. Sections 6 and 7 are devoted to the applications of our algorithm.

2 PRELIMINARIES

2.1 Model and Notations

Let us now define the CONGEST model of distributed computation. We model distributed systems as an undirected and unweighted graph, $G$, on $n$ nodes, where the nodes are computational units and edges are communication links. We assume that nodes have unique $O(\log n)$ bit IDs. Communication between nodes happens in synchronous rounds. In each round, each node sends a (possibly different) $O(\log n)$-bit message to each neighbor and, within the same round, receives all messages from the neighbors. After receiving the messages, it performs some local computation. We assume that nodes have unbounded computational power, and when analyzing our algorithms, we only care about the communication cost of the algorithm. That is, the number of communication rounds it takes to complete. For any graph $G$, we denote its vertex and edge sets by $V(G)$ and $E(G)$ respectively.

While we also deal with directed and weighted multigraphs as input instances, the communication network itself is modeled as a simple undirected unweighted graph (i.e., the orientation, weight, and multiplicity of the edges connecting two vertices do not affect the communication capability between them). More precisely, given an input instance $G$, we denote by $[G]$ the graph which is obtained by omitting all orientations of $E(G)$, by merging the multiedges connecting the same two vertices into a single one, and by removing all self-loops. Then $[G]$ is the communication network implied by $G$. Given a graph $H$, we denote by $D(H)$ the undirected diameter of $H$ (i.e., $D(H)$ is the diameter of $[H]$). For the input graph $G$, we use $D$ instead of $D(G)$. For any rooted tree $T$ and a vertex $v \in V(T)$, we denote by $T(v)$ the subtree of $T$ rooted by $v$. We also denote by $\text{ch}(T,v)$ the set of the children of $v$ in tree $T$.

2.2 Tree Decomposition and Treewidth

Let $G = (V(G),E(G))$ be an undirected and unweighted graph. A tree decomposition of an undirected and unweighted graph $G$ is a pair $\Phi = (T, \{B_x\}_{x \in V(T)})$, where $T$ is a tree, referred to as decomposition tree, and each vertex $x \in V(T)$ is associated with a subset $B_x \subseteq V(G)$ of vertices in $G$ (called bag $x$) satisfying the following conditions:

(a) $V(G) = \bigcup_{x \in V(T)} B_x$.
(b) Any edge in $G$ is covered by at least one bag, i.e., for all $(u,v) \in E(G)$, there exists $x \in V(T)$ such that $u, v \in B_x$ holds.
(c) For any $u \in V(G)$, the subgraph of $T$ induced by the bags containing $u$ is connected.

The width of a tree decomposition $\Phi = (T, \{B_x\}_{x \in V(T)})$ is defined as the maximum bag size minus one. The treewidth, $\tau$, of a graph $G$ is the minimum width over all tree decompositions of $G$. While the original definition of treewidth applies only to undirected graphs, we define the treewidth of a directed graph $G$ as the treewidth of $[G]$.

Throughout this paper, we assume that any decomposition tree $T$ is rooted, and each vertex in $V(T)$ is identified by a string over the alphabet $[0, n−1]$. Letting $x$ be any string over the alphabet $[0, n−1]$ and $i$ be any character, we define $x \cdot i$ as the string obtained by adding $i$ to the tail of $x$. The null string of length zero is denoted by $\psi$, which is the identifier of the root of $T$. Given a vertex $x \in V(T)$, $x \cdot i$ identifies the $i$-th child of $x$. We use the notation $x \subseteq y$ if $x$ is a prefix of $y$, and the notation $x \parallel y$ if neither $x \subseteq y$ nor $y \subseteq x$ holds. We denote the length of $x$ by $|x|$, which means the depth of vertex $x$ in $T$. We define $\text{Ar}_t(T)$ as the set of vertices of length $t$ in $V(T)$. For any tree decomposition $\Phi = (T, \{B_x\}_{x \in V(T)})$ and $v \in V(G)$, its canonical string $c_\Phi^G(v)$ is the shortest string such that $v \in B_{c_\Phi^G(v)}$ holds. Note that $c_\Phi^G(v)$ is uniquely determined because the set of bags containing $v$ forms a connected subgraph of $T$ (by condition (c) of the definition). The subscript $\Phi$ is often omitted when it is clear from context. Letting $x$ be a string of non-zero length in $V(T)$, we denote by $p(x)$ the string corresponding to the parent of $x$ (i.e., the string obtained by chopping the tail of $x$) in $T$. We define $\text{ch}_t(x)$ as the set of $i \in [0, n−1]$ such that $x \cdot i$ is a child of $x$, i.e., $\text{ch}_t(x) = \{i \mid x \cdot i \in V(T)\}$.

In the distributed setting, computing a tree decomposition means that each node $u \in V(G)$ outputs the IDs of the bags containing $u$.

2.3 Part-wise Aggregation

Throughout this paper, we often execute an algorithm, $\mathcal{A}$, on multiple subgraphs of the input graph independently and simultaneously. That is, given a collection $\mathcal{H} = \{H_1, H_2, \ldots, H_N\}$ of vertex disjoint connected subgraphs of the input graph $G$, we execute $\mathcal{A}$ on all $H_i \in \mathcal{H}$ in parallel. The primary obstacle in implementing this type of execution in the CONGEST model is that the diameter $D(H_i)$ for $H_i \in \mathcal{H}$ may be much larger than $D(G)$ (and can be $\Omega(n)$ in the worse case), and thus the running time of $\mathcal{A}$ in $H_i$ can depend on $n$ even if the running time of $\mathcal{A}$ depends only on the diameter of the input graph. The key technical ingredient for this section is a subroutine called part-wise aggregation [24], which is
defined as follows: Let $G = (V(G), E(G))$ be an undirected graph, $H = \{H_1, H_2, \ldots, H_k\}$ be a collection of connected vertex disjoint subgraphs of $G$, and $\tau$ be an associative binary function operating on a value domain $\mathcal{M}$ of cardinality poly$(n)$. Suppose that each node $v \in V(H_i)$ knows all the edges in $E(H_i)$ incident to $v$, and has a value $x_{ai}$ for $\forall v \in V(H_i)$. Every node in $H_i$ wants to learn the value $\bigoplus_{v \in V(H_i)} xa_i$, i.e., the aggregation with operator $\oplus$ over all the values $x_{ai}$ for $v \in V(H_i)$. It is known that bounded treewidth graphs admit a fast algorithm for part-wise aggregation [28, 29, 43], which runs in $O(\tau D)$ rounds.

3 FULLY POLYNOMIAL-TIME DISTRIBUTED TREE DECOMPOSITION

3.1 Balanced Separator

Our tree decomposition algorithm is based on the computation of balanced separators, which is a common technique used in many (centralized or distributed) tree decomposition algorithms. We first introduce the notion of a $(X, \alpha)$-balanced separator, which is a slight generalization of a conventional balanced separator.

Let $X$ be any subset of $V(G)$. For a given vertex subset $Y \subseteq V(G)$, we define $\mu_X(Y) = |Y \cap X|$. We also use a similar notation $\mu_X(H)$ for any subgraph $H \subseteq G$ to mean $\mu_X(V(H))$. The subscript $X$ is omitted if it is clear from the context. An $(X, \alpha)$-balanced separator, $S$, of an undirected graph $G$ is a vertex subset whose removal divides $G$ into $N$ connected components $G_1, G_2, \ldots, G_N$ such that $\mu_X(G_i)/\mu_X(G) \leq \alpha$ holds for any $1 \leq i \leq N$. A $(V(G), \alpha)$-balanced separator is simply called an $\alpha$-balanced separator of $G$.

It is well-known that any graph $G$ admits a $(1/2)$-balanced separator whose size is $\tau + 1$ (Lemma 7.19 of [12]), and that one can obtain a tree-decomposition algorithm of width $O(\tau \log n)$ from any balanced separator algorithm which outputs a separator $S$ of size $t$. The best known CONGEST algorithm for finding an $\alpha$-balanced separator of size $O(\tau)$ (for constant $\alpha < 1$) [43] has a running time that exponentially depends on $\tau$, and thus does not fit our goals. To get rid of the exponential dependency, we present a new CONGEST algorithm for computing balanced separators building on the ideas of the centralized algorithm by Fomin et al. [16] (referred to as FLPSW hereafter).

3.2 The Algorithm by Fomin et al.

We first present the outline of FLPSW. We assume $X = V(G)$ for simplicity, but the algorithm can handle an arbitrary $X$. FLPSW runs with a parameter $t$, and is guaranteed to output an $\alpha$-balanced separator of size $O(t^2)$ for $\alpha = 1 - \Theta(1) > 0$ when $\tau + 1 \leq t$. In the case when $\tau$ is unknown, one can combine FLPSW with a standard doubling estimation technique for $t$. To explain the algorithm we first introduce the notion of an $U_1-U_2$ vertex cut for $U_1, U_2 \subseteq V(G)$ as a generalization of the standard $s-t$ vertex cut, which is defined as a vertex subset $Z \subseteq V(G) \setminus (U_1 \cup U_2)$ such that $U_1$ and $U_2$ belong to different connected components in $G - Z$. If $U_1$ intersects with $U_2$ or some edge crosses between $U_1$ and $U_2$, the size of the $U_1-U_2$ vertex cut is defined as $\infty$.

Let $S$ be a $(1/2)$-balanced separator of $G$ of size at most $t$ (recall that it necessarily exists). The algorithm first constructs any rooted spanning tree, $T$, of $G$, and decomposes it into a set $T'$ of size $\Theta(n/t)$ subtrees of size $\Theta(n/t)$ such that only their root vertices are shared among two or more subtrees in $T$. In what follows, we refer to this type of decomposition as the splitting of $T$, and to each subtree as a split tree. Let $R$ be the set of the root vertices of split trees. We assume that the hidden constant in the cardinality of $T'$ is sufficiently large, e.g., $T' \geq 100t$. There are two cases to consider.

- (Case 1) $R$ does not intersect $S$: Since all trees in $T'$ are vertex disjoint except for $R$, at most $|S| = t$ trees in $T'$ intersect $S$. Then there exist two split trees $T_1, T_2 \in T'$ such that they belong to different connected components in $G - S$, i.e., the minimum $V(T_1)$-$V(T_2)$ vertex cut $Z$ has size at most $|S| = t$. The algorithm finds such a pair by computing the minimum $V(T_1)$-$V(T_2)$ vertex cut for all pairs $(T_1, T_2) \in T^2$. Once the pair is found, the algorithm outputs $Z$ as the separator. Note that $Z$ is a $(X, 1 - \Theta(1/t))$-balanced separator of $G$ because both $V(T_1)$ and $V(T_2)$ contain $\Theta(n/t)$ vertices. If the algorithm fails to find such a pair, it concludes that the first case does not apply, and proceeds to case 2.

- (Case 2) $R$ intersects $S$: The algorithm simply removes $R$ from $G$ and outputs it as the separator. The removal of $R$ results in the deletion of at least one vertex in $S$ from $G$.

FLPSW iterates the procedure above $2t$ times for the largest connected components of the remaining graph. After all iterations are complete, we obtain a $(1 - \Theta(1))$-balanced separator. That is, if the first case succeeds $t$ times, then $\Theta(1)$ fraction of vertices are separated. Otherwise, all vertices in $S$ are removed. Since each iteration adds $O(t)$ vertices to the output set, the total size of the output separator is $O(t^2)$. As stated in the introduction, it is relatively straightforward to implement FLPSW in $O(\tau O(1)) D$ rounds if we do not care about optimizing the exponent of $\tau$.

3.3 Our Algorithm

We present a modified version of FLPSW which admits a more efficient CONGEST implementation. The key differences between our algorithm and FLPSW are threefold. First, instead of solving the minimum $V(T_1)$-$V(T_2)$ vertex cut problem for all $(T_1, T_2) \in T^2$, we simply adopt a random sampling strategy for identifying a pair $(T_1, T_2)$ which has a cut of $O(t)$ vertices. When $S \cap R = \emptyset$ holds, this strategy is guaranteed to succeed with a constant probability. Since one pair $(T_1, T_2)$ is sampled per iteration, it suffices to solve $O(t)$ instances of the minimum vertex cut in total.

The second idea is a parallelization-friendly algorithm for tree splitting. More precisely, the algorithm manages a set of disjoint trees $T$, where initially $T = \{T\}$, and iteratively splits trees of large size in $T$, and then adds back the resulting split trees to $T$ if they are still large. This strategy admits an efficient CONGEST implementation because the splitting of two different trees in $T$ can be performed in parallel.

The third idea is to compute $O(t)$ instances of the minimum vertex cut simultaneously at the final step (FLPSW performs this computation sequentially). Utilizing a careful scheduling technique, we can execute $t$ independent instances of the minimum vertex cut problem in $O(\tau D + t^2 D)$ rounds, which is more efficient than sequential processing (which takes $O(t^2 D)$ rounds).

We present the centralized version of our algorithm (referred to as SEP hereafter). It works as follows:
(1) If $\mu(G) \leq 200t^2$, the algorithm outputs $X$ as a $(X, 14399/14400)$-balanced separator and halts.

When the algorithm does not halt at step 1, the algorithm iteratively applies the following steps (2 and 3) for $t = \lceil 301t/300 \rceil$ times to the graphs $G_1, G_2, \ldots, G_t$, where $G_1 = G$ and the rest of the sequence is generated within the following steps.

(2) At the beginning of the $t$-th iteration (for $G_t$), the algorithm constructs some spanning tree $T'$ of $G_t$, and then split $T'$ into several trees. In the $t$-th iteration, this splitting procedure, which we refer to as Split, maintains the two sets of trees $T$ and $T_i$, which initially store $T = \{T'\}$ and $T_i = \emptyset$. By a single invocation of Split, every tree $T \in T'$ is split into a set of trees of size at least $\mu(G)/(12t)$ and at most $5\mu(T)/6$. The original tree $T$ is removed from $T'$ after splitting. Each split tree is added to $T'$ if its size is more than $\mu(G)/(4t)$, or to $T_i$ otherwise. The splitting process terminates when $T'$ becomes empty.

The details of the procedure Split are as follows: For any $T \in T_i$, the algorithm finds the center vertex $c$ in $V(T)$, i.e., the vertex such that removing it decomposes $T$ into several subtrees of size at most $\mu(T)/2$. Now we regard $c$ as the root of $T$. Next, Split removes all subtrees $T(v)$ for $v \in ch(T, c)$ such that $\mu(T(v)) \geq \mu(G)/(12t)$ as split trees. Let $T'$ be the remaining tree. If $\mu(T') < \mu(G)/(12t)$, we pick any tree $T(v)$ split in the first step, and merge $T'$ into $T(v)$ (Fig. 1(a)). The size of $T' + T(v)$ is bounded by $\mu(G)/(12t) + \mu(T)/2 < \mu(T)/3 + \mu(T)/2 < 5\mu(T)/6$ (recall that any $T \in T'$ has a size at least $\mu(G)/(4t)$ and thus $\mu(G)/(12t) \leq \mu(T)/3$ holds). Otherwise, we further split $T'$ into several subtrees sharing $c$ as their roots. Let us fix some ordering of the children of $c$ in $T'$, denoted by $y_0, y_1, \ldots, y_{t-1}$, we define $V_{a,b} = \bigcup_{a \leq b} V(T'(y_k))$. The algorithm computes the indices $0 = q_0 < q_1 < \cdots < q_t = t - 1$ such that $\mu(G)/(12t) < \mu(V_{q_0,q_1}) < \mu(G)/(4t)$ holds for all $1 \leq h \leq t - 1$ and $\mu(V_{q_{t-1},q_t}) < \mu(G)/(4t)$ holds for $h = t'$. Then we split $T'$ into $t'$-connected subtrees induced by $V_{q_0,q_t+1} \cup \{c\}$. Since the subtree $T'(y)$ for any $y \in ch(T, c)$ has a size less than $\mu(G)/(12t)$, one can always obtain such a splitting. Each induced subtree is added to $T_i$ because its size is necessarily at most $\mu(G)/(4t)$ (Fig. 1(b)). It is easy to see that $T'$ becomes empty after $O(\log t)$ invocations of Split. At which point $T_i$ is a set of split trees covering $T'$, whose size is in the range $[\mu(G)/(12t), \mu(G)/(4t)]$.

(3) We denote the set of root vertices of subtrees in $T_i$ by $R_i$. If $R_i = \bigcup_{1 \leq i \leq t} R_i$ is a $(X, 14399/14400)$-separator of $G$, the algorithm outputs it and halts. Otherwise, we define $G_{t+1}$ as the heaviest connected component of $G_i - R_i$ with respect to $\mu$.

If the algorithm completes $t$ iterations of the steps above without halting, the following step is performed.

(4) For each $i \in [1, t]$, the algorithm chooses 95 ordered pairs uniformly at random from $T_i \times T_i$, and compute the $V(T_{\ell_1}) - V(T_{\ell_2})$ vertex cut for all chosen pairs. If the computed cut size is at most $t$, the cut vertices are added to the set $Z$. Finally, $Z$ is outputted if it is a $(X, 14399/14400)$-balanced separator. Otherwise $Z$ fails. When $Z$ fails for 5 log $n$ trials, it concludes that $\tau + 1 > t$ (and runs again after doubling $t$).

Figure 1: An illustration of the Split procedure. Subtrees circled by a red dotted line are split subtrees.

While the fundamental idea of Sep is similar to Flpsw, it is not straightforward to show the correctness of our algorithm. One significant technical challenge is that the subtree pairs chosen in Step (4) are not vertex disjoint. This is problematic as we must prove that the size of the largest connected component after removing the computed separator $Z$ becomes substantially smaller. The complete correctness proof (deferred to the full version) includes a new analysis that overcomes this matter. To implement Sep efficiently in the CONGEST model, we utilize the part-wise aggregation technique explained in Section 2.3, which is also known to provide efficient algorithms for minimum $U_1-U_2$ vertex cut and spanning tree construction running in $\tilde{O}(\tau^{O(1)} D)$ rounds [24, 29, 43]. Finally, we obtain the following lemma:

Lemma 1. Let $G$ be an undirected graph, and $X \subseteq V(G)$ be any vertex subset. There exists a randomized CONGEST algorithm that outputs a $(X, 14399/14400)$-balanced separator of size at most $400(\tau + 1)^2$ for $G$ in $\tilde{O}(\tau^2 D + \tau^3)$ rounds w.h.p.

3.4 Distributed Tree Decomposition based on Balanced Separators

We construct a tree decomposition of width at most $O(\tau^2 \log n)$, utilizing the balanced separator algorithm of Lemma 1. We refer to the constructed tree decomposition as $\Phi = (T, \{B_\ell\}_{\ell \in V(T)})$. As explained in Section 2.2, the subscript $x$ of each bag is a string over the alphabet $[0, n-1]$. Initially, let $G_0 = G$. There exists a standard strategy to obtain a decomposition from any balanced separator algorithm, which works as follows: We first compute
a balanced separator $S$ of $G = G_ϕ$. The set $S$ becomes the root bag $B_ϕ$ of the constructed tree decomposition. For each connected component $G_0, G_1, \ldots, G_{N-1}$ of $G - S$, we recursively construct their tree decompositions. Finally, we add $S$ to all of the bags in those decompositions, and connect their roots $1, 2, \ldots, N$ to the root $ϕ$ of the whole tree decomposition as children. Using an algorithm for computing a balanced separator of size $O(τ²)$, this strategy yields a tree decomposition of size $O(τ² \log n)$. However, adapting this strategy to the distributed setting is problematic, mainly due to the fact that the bag $B_ϕ$ is not a subset of the vertices of the corresponding graph $G_ϕ$. To avoid this, our algorithm utilizes a slightly modified strategy.

The algorithm recursively decomposes $G_ϕ$ for each string $x$ by fixing the corresponding bag $B_ϕ$. It first computes a $O(1)$-balanced separator $S_x$ of $G_ϕ$ using the algorithm of Lemma 1. If $|V(G_ϕ)| \leq 2|S_x|$, we define $B_ϕ = |V(G_ϕ)|$ and the recursion terminates. Otherwise, we define the bag $B_ϕ = V(G_ϕ) \cap (\cup_{x' \leq x} S_{x'})$. Let $G_ϕ^{x}_0, G_ϕ^{x}_1, \ldots, G_ϕ^{x}_{N-1}$ be the connected components of $G_ϕ - B_ϕ$. The graph $G^{x}_i$ (for $0 \leq i \leq N - 1$) is defined as $G^{x}_i = G^{x}_i + \{(u, v) | (u, v) \in (V(G^{x}_i \times B_ϕ) \cap E(G_ϕ))$. This decomposition strategy is illustrated in Fig. 2. This construction yields a tree decomposition of width $O(τ² \log n)$ and guarantees $B_ϕ \subseteq G_ϕ$ for any $x \in V(T)$.

Figure 2: An illustration of our tree decomposition.

A primary obstacle in adapting this decomposition strategy to the distributed setting is the fact that the collection of subgraphs, $G_t = \{G_x | x \in A_t\}$, for any $t$ is not necessarily vertex disjoint. For the recursive construction of the tree decomposition, we need to execute our balanced separator algorithm for all graphs in $G_t$ in parallel. To circumvent this obstacle, the algorithm computes the separator of $G^{x}_i$ (i.e. $G^{x}_i - B_{ϕ(x)}$) instead of $G^{x}_i$. Since $\{G^{x}_i | x \in A_t\}$ are the collection of connected and vertex disjoint subgraphs by definition, one can apply our separator algorithm to compute the separators of graphs $G^{x}_i$ for all $x \in A_t$ in parallel. Each separator $S^{x}_c$ for $G^{x}_i$ is easily transformed into the separator $S_{xc}$ for $G_{xc}$ by adding all vertices in $V(G_{xc}) \cap V(B_{ϕ(x)})$. The bag $B_{xc}$ is defined as $B_{xc} = V(G_{xc}) \cap (\cup_{x' \leq x} S_{xc'}) = B_{ϕ(x)} \cup S^{x}_c$, and thus its size is bounded by $O(τ² \log n)$. Our main theorem is stated as follows:

**Theorem 1.** For a given graph, $G = (V,E)$, there exists a tree decomposition algorithm in the CONGEST model, which constructs a tree decomposition, $Φ = (T, \{B_x\}_{x \in V(T)})$, of width $O(τ² \log n)$ whp. The depth of $T$ is $O(\log n)$ and the running time of the algorithm is $O(τ²D + τ²)$ rounds.

4 DISTRIBUTED DISTANCE LABELING IN LOW-TREewidth GRAPHS

4.1 Outline

Consider the weighted and directed input graph $G = (V(G), E(G))$ with edge cost function $c_G : E(G) \rightarrow \mathbb{N}$. The distance labeling problem is formally defined as follows:

**Definition 1 (Distance Labeling (DL)).** Distance labeling consists of a labeling function $la_G : V(G) \rightarrow \{0,1\}^*$ that depends on the input graph $G$ (which can be directed and weighted), and a common decoder function $dec : \{0,1\}^* \times \{0,1\}^* \rightarrow \mathbb{N}$. The decoder returns the distance $dc_G(u,v)$ from/to all labels $la_G(u)$ and $la_G(v)$. The problem requires that each node $v \in V(G)$ outputs its label $la_G(v)$.

Our distributed implementation for distance labeling adopts a similar approach to the algorithm of Li and Parter [44] for planar graphs, whose structure is a slightly modified version of the distance labeling scheme by Gavoille et al. [22]. Our implementation is a recursive algorithm utilizing tree decomposition, and can roughly be stated as follows: Let $G$ be a weighted directed input graph, $Φ = \{T, \{B_x\}_{x \in V(T)}\}$ be the (rooted) tree decomposition of $G$ constructed by the algorithm of Theorem 1. The algorithm recursively and independently constructs distance labels for each graph in $G_t$, and then each node $u$ in $G = G_ϕ$ learns the distances from/to all of the nodes in $B_ϕ$, and stores them in the label of $u$ constructed in $G_ϕ$ (where $G_ϕ$ is defined in Sec. 3.4). Consider computing the distance from $u$ to $v$. If the shortest path form $u$ to $v$ does not contain any vertex in $B_ϕ$, the distance obtained by the label of $u$ and $v$ for $G_ϕ$. Otherwise, it suffices to take the minimum of the distance from $u$ to $s$ plus that from $s$ to $v$ for every $s \in B_ϕ$, which can computed from the labels of $u$ and $v$.

We formally define the distance labeling constructed by our CONGEST algorithm. For simplicity of presentation, we assume that edge cost function $c_G$ is a mapping from $V(G) \times V(G)$ to $\mathbb{N} \cup \{\infty\}$, where we define $c_G(u,v) = \infty$ if $(u,v) \notin E(G)$. A distance set $dc_G(u,X)$ for $u \in V(G)$ and $X \subseteq V(G)$ is defined as the set of tuples $(u,v,dc_G(u,v))$ and $(u,v,dc_G(v,u))$ for all $v \in X$. We also define $B_ϕ(u,v) = (\cup_{x' \leq x} S_{xc'}) \cap B_ϕ$. The label $la_G(u)$ is defined as $la_G(u) = dc_G(u,B_ϕ(u))$. The decoder function $dec$ is defined as follows:

$$dec(la_G(u),la_G(v)) = \min_{s \in B_ϕ(u) \cap B_ϕ(v)} dc_G(u,s) + dc_G(s,v).$$

Using the tree decomposition algorithm of Theorem 1, the label size is bounded by $O(τ²)$ bits. The lemma below guarantees the correctness of this labeling scheme.

**Lemma 2.** For any $u,v \in V(G)$, $dec(la_G(u),la_G(v)) = dc_G(u,v)$ holds.

4.2 Distance Labeling Construction

We explain the construction of $la_G(u)$ for all $u \in V(G)$ in the CONGEST model. First we introduce the graph $H_x$ associated with each $B_x$ as follows:
We state the following theorem.

The key properties of the graph $H_x$ are stated in the following two lemmas.

**Lemma 3.** For any $u,v \in V(H_x)$, $d_{H_x}(u,v) = d_{G_u}(u,v)$ holds.

**Lemma 4.** Let $u$ and $v$ be any two vertices in $V(G_{uk}) \cup B_x$ for some $i \in \text{cht}(x)$. Then the following equality holds.

$$d_{G_u}(u,v) = \min\{d_{G_{s1}^{u,v}}(u,v), \min_{s,s' \in V(H_x)} (d_{G_{s1}^{u,v}}(u,s) + d_{H_x}(s,s') + d_{G_{s1}^{u,v}}(s',v))\}.$$

The construction of the labels follows a bottom-up recursion over the decomposition tree $T$. More precisely, the proposed algorithm constructs $\text{lag}_{G_x}(u)$ for all $u \in V(G_x)$, provided that $\text{lag}_{G_{s1}}(u)$ for all $u \in G_{s1}$ and $i \in \text{cht}(T,x)$ are available. The outline of the algorithm is as follows:

1. If $x$ is a leaf in $T$, each node $u \in V(G_x)$ broadcasts the information of the edges incident to $u$ in $G_x$ to the nodes in $G_x$, i.e., each node $u$ knows the entire information of $G_x$. Since the collection of graphs for every leaf $x$ is not vertex disjoint, we reduce this task to part-wise aggregation for $G'$. That is, every node $v$ in $V(G_x) \setminus V(G')$ "outsources" the task to some neighbor $v'$ belonging to $V(G')$ by sending its own information. Since $G_{s1}'$ for all leaves $x$ are vertex disjoint, the information exchange among the nodes in $V(G_{s1})'$ is implemented by part-wise aggregation. Finally, the outsourced neighbor $v'$ returns all the information obtained to $v$. By solving the all-pairs shortest paths problem locally, $u$ obtains the label $\text{lag}_{G_x}(u)$. If $x$ is not a leaf, the algorithm executes steps (2)-(4).

2. For all $i \in \text{cht}(x)$, the algorithm recursively constructs the distance labeling $\text{lag}_{G_{s1}^{i}}(u)$ for all $u \in V(G_{s1}^{i})$ utilizing $\Phi' = (T(x \bullet i), \{B_{s1}^{i} \setminus x \in \Phi'\})$ as the tree decomposition of $G_{s1}$. Since the node $x \bullet i$ is the root of $T(x \bullet i)$, $B_{s1}^{i} \subseteq B_{\Phi'}(u)$ trivially holds. That is, $\text{lag}_{G_{s1}^{i}}(u) = d_{G_{s1}^{i}}(u, B_{\Phi'}^{i}(u))$ necessarily contains the information of the distance set $d_{G_{s1}^{i}}(u, B_{\Phi'}^{i})$. Thus, each node $u \in B_{s1}$ can identify the edges incident to $u$ in $H_x$ is done via local computation.

3. Each node $u \in B_{s1}$ broadcasts the set of edges incident to $u$ in $H_x$ to all nodes in $G_x$, which is implemented using the same technique as the information exchange in step (1).

4. Using the information received at step (3), each node $u \in V(G_{s1})$ locally knows $H_x$. Following the formula of Lemma 4, $u$ updates the distance set $d_{G_{s1}^{i}}(u, B_{\Phi'}^{i}(u))$ to $d_{G_{s1}^{i}}(u, B_{\Phi'}^{i}(u))$, and learns the distance set $d_{G_{s1}^{i}}(u, B_{s1})$.

We state the following theorem.

**Theorem 2.** Let $G = (V(G), E(G))$ be any directed graph with edge cost function $c : E(G) \rightarrow \mathbb{N}$. Then there exists a randomized CONGEST algorithm that solves DL in $O(\log^2 n)$ rounds with probability at least $1 - 1/n^9$. The label size of each node is $O(\log^2 n)$ bits.

## 5 STATEFUL WALKS

### 5.1 Definition

Let $G = (V(G), E(G), y_G)$ be a directed multigraph, where $y_G$ is a mapping from each element in $E(G)$ to an ordered pair in $V(G) \times V(G)$ (as $G$ is a multigraph, we cannot assume that elements in $E(G)$ are of the form $(u, v)$, and thus $y_G$ is required). A walk $w = e_1, e_2, \ldots, e_t$ in $G$ is a sequence of edges in $E(G)$ such that for any $i \in [1, t-1], y_G(e_i) \in E_G(e_{i+1}) \{0\}$. To describe the vertices in the walk explicitly, it can also be represented as an alternating sequence of vertices and edges, $w = u_0, e_1, u_1, e_2, \ldots, u_{t-1}, u_t$. Since $y_G(\Phi_i) \in E_G(\Phi_{i+1}) \{0\}$, the set $\text{lag}_{G_{s1}}(u)$ represents the vertices and edges in $w$.

A walk constraint is a subset $C \subseteq G$. That is, a walk constraint limits the set of graph walks to the subset $C$. Given a walk constraint, we denote by $W_{G,C}(s,t)$ the set of all walks from $s$ to $t$ in $C$. If $W_{G,C}(s,t)$ is not empty, we say that $t$ is reachable from $s$ in $G$. The $C$-distance from $s$ to $t$, denoted by $d_{G,C}(s,t)$, is the shortest (weighted) length of all the walks in $W_{G,C}(s,t)$. We consider a variety of the single-source shortest paths problem, which requires that for a given constraint $C$ and source node $s$ each node $v \in V(G)$ outputs the shortest walk from $s$ to $v$ in $C$, as well as its length $d_{G,C}(s,v)$. As discussed previously, this problem becomes meaningful only when $C$ is provided to the nodes of the graph in an implicit and distributed manner. To formally define the above, we present the notion of a stateful walk constraint, followed by an intuitive description of the definition.

**Definition 2 (Stateful Walk Constraint).** Let $Q$ be any set containing two special elements $\bot$ and $\top$. A walk constraint $C \subseteq G$ is called stateful if it contains $\bot$ and there exists a function $M_C : \pi_{G,s} \rightarrow Q$ and a tuple of transition functions $\delta_C \subseteq \pi_{E,G} \times \pi_{E,G}$ associated with each edge $e \in E(G)$, where $\delta_{C,e}$ is a mapping from $Q$ to $\top$, satisfying the following three conditions:

1. $M_C(w) = \bot$ if and only if $w \in C$. In addition, $M_C(w) = \top$ if and only if $w = \bot$.

2. For any $w \in G$ terminating at $u$ and $e = (u,v) \in E(G)$, $\delta_C(e)(M_C(w)) = M_C(w \circ e)$ holds.

3. For any $e \in E(G)$, $\delta_C(e)(\bot) = \bot$.

We omit the subscript $C$ of $M$ and $\delta$ when it is clear from the context. The rough intuition of the definition above is as follows: Every walk $w \in W_G$ has a state in $Q$ (which is referred to as the state of $w$ hereafter). Starting from the length-zero walk $\bot$, which has the special state $\top$, the state of the walk changes as the length of the walk increases. The function $\delta$ determines how the state of a given walk $w$ changes by appending edge $e$ to its tail. The second condition implies that the state of $w \circ e$ is determined only by the state of $w$ and the edge $e$ and is independent of any other feature of $w$. The state $\bot$ is a "reject" state, which implies $w$ does not satisfy the constraint $C$ (i.e., $w \notin C$). Condition 3 implies that once a walk
w does not satisfy \( C ( i.e., M_C (w) = \bot) \), no addition of edges to w can make it satisfy C. Let \( E^\text{sat}_G (u) \) be the set of outgoing edges from u in G. Assuming each node u knows the transition function \( \delta_{C,e} \) for all \( e \in E^\text{sat}_G (u) \), a stateful walk constraint implies that each node u can locally decide the state of a walk w leaving u only from the state of the incoming prefix of the walk. For a stateful walk constraint \( C \), a walk with a state other than \( \bot \) (i.e. a walk in C) is called a stateful walk. We now present two concrete examples of stateful walks.

Example 1 \((c\text{-Colored Walk})\). Here we assume edges have colors, and we are interested in walks where no two consecutive edges are monochromatic. Consider the edge label domain \( \Sigma \) of cardinality \( c \) (i.e. color palette), and an assignment \( f : E(G) \rightarrow \Sigma \) of a color to each edge. A \( c \)-colored walk \( w = e_1, e_2, \ldots, e_l \) is a walk satisfying \( f(e_l) \neq f(e_{l+1}) \) for all \( 1 \leq i < l \). The set of all \( c \)-colored walks \( C_{\text{col}(c)} \subseteq W_G \) is a stateful walk constraint associated with the following triple \((Q, M, \delta)\): The state domain is \( Q = \{ \Sigma \} \cup \{ \bot, \top \} \). For any \( w \in W_G \), \( M(w) \) represents \( f(e) \) for the last edge in \( w \) if \( w \in C_{\text{col}(c)} \), \top if \( w = \phi \), and \bot otherwise. The state \( \delta_c(q) \) is \( f(e) \) if \( q \neq f(e) \), and \bot otherwise.

Example 2 \((\text{count}-c \text{ Walk})\). Here we assume edges are assigned a binary value \((i.e., zero \text{ or one})\), and we are interested in walks that contain at most \( c \) edges of value one. A \( c \)-walk \( w = e_1, e_2, \ldots, e_l \) is a walk satisfying \( \sum_{i \in [l]} f(e_i) \leq c \). The set of all \( c \)-walks, \( C_{\text{cnt}(c)} \), is a stateful walk constraint associated with the following triple \((Q, M, \delta)\): The state domain is \( Q = \{0, c\} \cup \{ \bot, \top \} \). For any \( w \in W_G \), \( M(w) \) represents \( \sum_{e \in w} f(e) \) if it is within the range \( [0, c] \), \top if \( w = \phi \), and \bot otherwise. The state \( \delta_c(q) \) is \( q + f(e) \) if \( q \not\in \{\bot, \top\} \) and \( q + f(e) \leq c \) hold, \( f(e) \) if \( q = \top \), and \bot otherwise.

As seen in the examples above, the specification of the function \( \delta \) typically relies only on the edge label \( f(e) \), but this characteristic is not mandatory.

Subsets of stateful walk constraints. Let \( C \) be a stateful walk constraint associated with the triple \((Q, M, \delta)\). We denote by \( C(q) \) the set of all walks with state \( q \) in C, and define \( C(Q') = \cup_{q \in Q} C(q) \) for \( Q' \subseteq Q \). By definition, \( C(Q') \) for any \( Q' \subseteq Q \) is a walk constraint (but not necessarily stateful). For example, while the stateful constraint for \( c \)-walks considers all walks of count at most \( c \), we can define exact \( c \)-walks (where the count is exactly \( c \)) as \( C(c) \subseteq C \).

5.2 Finding Stateful Walks

In this section, we show how to reduce the problem of finding shortest stateful walks to the problem of finding unconstrained shortest walks. We present a general framework for reducing the constrained version of the shortest paths problem, for any stateful constraint \( C \subseteq W_G \), into the unconstrained version in some auxiliary directed graph \( G_C \). The construction of \( G_C \) is defined as follows:

- If the input graph \( G \) is weighted (by an edge-cost function \( c : E(G) \rightarrow \mathbb{N} \)), for any \( u, v \in V(G) \), assign the cost \( c(u, v) \) to any edge \( E(G) \cap (U_G(u) \times U_G(v)) \).

The intuition of the above construction is that we wish to break down the vertex \( u \) into \( U_G(u) \) in order to distinguish walks entering \( u \) with different states. The vertex \((u, i)\) can be seen as the arrival vertex of any walk \( w \) to \( u \) with state \( i \). Since we add an edge between \((u, i)\) and \((o, j)\) if and only if \( \delta_{C,e}(i) \) is \( j \) holds, the walk \( w \in e \), which has state \( \delta_C(e) = j \) in the original graph \( G \), always reaches \((o, j)\). An illustration of this construction is presented in Figure 3. Note that the second condition is introduced in order to bound the diameter of \( [G_C] \) by \( O(D) \). The distance from any node in \( U_G(u) \) to any node in \( U_G(v) = 0 \) because there exists a path from \((u, \bot)\) to \((v, \bot)\) of length \( d_G(u, v) \) (recall that condition 2 of Definition 2 implies \((u, \bot), (v, \bot)\) \( \in E(G_C) \) for any \( u = (u, v) \in E(G) \)). We state the following lemma.

**Lemma 5.** Let \( G = (V(G), E(G), \gamma_G) \) be any multigraph with edge-cost function \( c : E(G) \rightarrow \mathbb{N} \), and \( C \subseteq W_G \) be a stateful walk constraint with associated triple \((Q, M, \delta)\). There exists a walk \( w \) of weighted length \( x \) from \( s \) to \( t \) with state \( q, (s, t) \in V(G), q \in Q \setminus \{ \bot \} \) in \( C \) if and only if there exists a walk \( w' \) of weight \( x \) from \( (s, v) \) to \((t, q) \) in \( G_C \).

For any stateful walk constraint \( C \), its state and two vertices \( s, t \in V_G \), this lemma allows us to compute the shortest walk in \( C(q) \) from \( s \) to \( t \) (and its distance) by computing the directed shortest path from \((s, v)\) to \((t, q)\) in \( G_C \). Letting \( p_{\text{max}} \) be the maximum edge multiplicity of the original graph \( G \), it is easy to simulate the execution of any CONGEST algorithm for \( [G_C] \) on top of the original communication graph \( [G] \) with \( O(|Q| \cdot p_{\text{max}}) \)-round overhead: Each node \( v \in V(G) \) is responsible for the simulation of the nodes in \( U_G(v) \). Consider the subgraph \( H(u, v) \) of \( G_C \) induced by \( U_G(u) \cup U_G(v) \). Each node in this subgraph has at most \( p_{\text{max}} \) outgoing edges. Thus, the total number of edges in \( H(u, v) \) is at most \( 2p_{\text{max}}|Q| \). A single communication round over the links in \( E(H(u, v)) \) can be achieved by \( O(|Q| \cdot p_{\text{max}}) \) communication rounds over the edge \((u, v) \in E(G_C) \). The total number of nodes in \( [G_C] \) is \( 4|Q|n \), and the diameter of \( G_C \) is \( O(D) \). It is easy to show that the treewidth of \( G_C \) is bounded by \( O(|Q| \cdot \tau) \). Given a tree decomposition of \( G \), we replace each vertex \( v \) in each bag by \( U_G(v) \). The resulting decomposition is obviously a tree decomposition of \( G_C \) and the bag size is multiplied by \( |Q| + 1 \). Consequently, any \( f(n, D, \tau) \)-round algorithm in \( G_C \) is simulated on the top of \( [G] \) within \( O(|Q| \cdot p_{\text{max}} f(n, D, \tau) (|Q| + 1)) \) rounds.

In our applications, we are interested in the constrained version of distance labeling schemes, which is formalized as follows:

**Constrained distance labeling (CDL(C))**: Let \( C \) be a stateful walk constraint with associated triple \((Q, M, \delta)\). It consists of a labeling function \( \lambda_{G_C} : V(G) \rightarrow \{0, 1\}^* \), that depends on the input graph \( G \), and a common decoder function \( d_{\text{dec}} : \{0, 1\}^* \rightarrow \mathbb{N} \). Both functions must satisfy \( d_{\text{dec}}(q, \lambda_{G_C}(u), \lambda_{G_C}(v)) = d_{G_C}(u, v) \) for any \( u, v \in V(G) \) and \( q \in Q \). The problem requires that each node \( v \in V(G) \) outputs its label \( \lambda_{G_C}(v) \).

Note that the input graph \( G \) can be directed and weighted. The problem CDL(C) in \( G \) is solved by any algorithm for (standard) distance
6 EXACT BIPARTITE MAXIMUM MATCHING

Let $G$ be an undirected unweighted graph. A matching $M \subseteq E(G)$ is a set of edges such that any two distinct edges do not share an endpoint. Given a matching $M$ of a graph $G$, we say that a vertex $u$ is unmatched if it is not an endpoint of any matching edge. The maximum matching problem requires the algorithm to output the maximum cardinality matching (by marking the edges in the solution). The maximum matching problem is known to be reducible to the task of finding an augmenting path, a simple path connecting two unmatched nodes where matching edges and non-matching edges appear alternately. Once an augmenting path is found, by flipping matching edges and non-matching edges, the size of the matching increases by one. The maximum matching is obtained by iterating this augmentation process until the current matching does not have any augmenting path.

An augmenting path can be seen as a simple 2-colored walk whose endpoints are both unmatched vertices, and thus it fits naturally into our stateful-walk framework (more precisely, the construction of $C_{\text{col}(2)}$-distance labeling following Example 1 and Theorem 3). This idea is incorrect for general graphs because the shortest 2-colored walk is not necessarily simple, but is valid for bipartite graphs: It is well-known that any shortest 2-colored walk is simple in bipartite graphs. However, we still have a few hurdles to overcome. The first issue is how does each unmatched node detect if there exists an augmenting path starting from itself? Since there might exist $\Omega(n)$ unmatched nodes, the trivial solution where all unmatched nodes broadcast their own $C_{\text{col}(2)}$-distance labels is very costly. The second issue is how to speed up the iterations of the matching update. The trivial sequential update takes $\Omega(n)$ iterations. We resolve these issues by a divide-and-conquer approach utilizing balanced separators. The key observation is that if the maximum matching is already computed for each connected component in $G - S$ independently (where $S \subseteq V(G)$ is any vertex subset), it suffices to check only the existence of augmenting paths with at least one endpoint in $S$. This observation is proved in the following simple proposition, which is a special case of a more general theorem presented in [35]:

Proposition 1 (Iwata et al. [35]). Let $G$ be any (undirected and unweighted) graph, $U \subseteq V(G)$ be a vertex subset, and $\mathcal{H} = \{H_1, H_2, \ldots, H_N\}$ be the set of connected components of $G - U$. Assume that the maximum matching $M_i$ for each connected component $H_i \in \mathcal{H}$ is already computed. Then, for any $v \in U$, the size of the maximum matching of the graph $G - (U \setminus \{v\})$ is at most $|U| + 1$. Any augmenting path in $G - (U \setminus \{v\})$ starts from $v$.

This proposition naturally induces a divide-and-conquer approach for bipartite maximum matching in low-treewidth graphs. That is, computing a balanced separator $S$, we recursively compute the maximum matching of each connected component of $G - S$. To obtain the maximum matching for the whole graph $G$, it suffices to consider only the augmenting paths starting from $S$. We state the following theorem.

Theorem 4. There exists a randomized CONGEST algorithm that computes the maximum matching for any bipartite graph $G$ in $\tilde{O}(\tau^2 D + \tau^3)$ rounds whp.
construction: The length of the shortest cycle containing a directed edge \((u, v) \in E(G)\) is determined by computing the distance from \(v\) to \(u\) in \(G\), which is obtained by exchanging the labels \(l_{uc}(u)\) and \(l_{uc}(v)\). To compute the girth, it suffices to execute this task for all edges, and take the minimum over all of the computed cycle lengths. In our setting, the running time of this algorithm is \(O((r + D^2 + r^2)\) rounds.

The case for undirected graphs is more challenging because the shortest path from \(v\) to \(u\) can contain the edge \((u, v)\), while such a case does not occur in directed graphs. This section provides a CONGEST algorithm that computes \(g\) in an undirected graph \(G\) using our framework. Let \(\Sigma = \{0, 1\}\) be the edge label domain. Recall that a walk \(w\) is called exact count-1 if \(w\) contains exactly one edge with label one. Since \(c_{\text{cnt}}(u) \subseteq c_{\text{cnt}}(u)\) is obviously the set of all exact count-1 walks, Theorem 3 allows us to compute the shortest length of exact count-1 walks from \(u\) to \(v\) for any two nodes \(u, v \in V(G)\) using their labels (i.e., \(\text{sd}_{c_{\text{cnt}}(u)}(1), \text{sla}_{c_{\text{cnt}}(u)}(u), \text{sla}_{c_{\text{cnt}}(u)}(v)\)) is the shortest length of exact count-1 walks from \(u\) to \(v\).

The following lemma is the key for our girth algorithm.

**Lemma 6.** Any shortest exact count-1 walk \(w\) starting and terminating at the same vertex \(v\) contains a simple cycle, and thus the weight of \(w\) is at least \(g\).

Note that the above holds for any assignment of binary edge labels. Assume a labeling function \(f\) such that some shortest cycle \(R = e_0, e_1, …, e_{l-1}\) has exactly one edge \(e_i \in E(R)\) which satisfies \(f(e_i) = 1\). Each node \(u\) computes the length of the shortest exact count-1 walk from \(u\) to \(u\). Let \(u\) denote this by \(g(u)\). As explained above, this is possible by using CDL\(c_{\text{cnt}}(u)\) (note that the shortest length of exact count-1 walks (i.e., \(c_{\text{cnt}}(u)\) -distance) from \(u\) to \(u\) is computed locally by the label \(\text{sla}_{c_{\text{cnt}}(u)}(u)\). As \(R\) is a shortest cycle in \(G\) with exactly one edge labeled \(1\), by Lemma 6, \(g(v) = g\) holds for every \(v \in V(R)\). Thus, we can compute the girth \(g = \min_{u,v \in V(G)} g(u)\) by standard aggregation over all nodes.

The final challenge is how to obtain the edge label function \(f\) satisfying the above condition. However, this can be resolved by a probabilistic label assignment. Let \(F\) be the set of the edges, \(e \in E(G),\) such that \(e\) is covered by at least one shortest cycle. Note that if exactly one edge in \(F\) has label one, the condition holds for at least one shortest cycle. To require that the condition holds with constant probability, it suffices to assign each edge with label one (independently of other edges) with probability \(p = \Theta(1/F)\). Repeating this process (assigning random labels, computing \(g\), keeping the minimum value of \(g\)) a logarithmic number of times, we can amplify the success probability to \((1-1/n)^\epsilon\). While the value of \(p\) is unknown to the algorithm, it can be estimated by a standard doubling technique. Consequently, we state the following theorem.

**Theorem 5.** There exists a randomized CONGEST algorithm that computes the girth, \(g\), of directed and weighted graph \(G\) in \(O((r^2D + r^2)\) rounds whp.

**ACKNOWLEDGMENTS**

The authors would like to thank Ami Paz for helpful comments and discussions.

**REFERENCES**

[1] Amir Abboud, Keren Censor-Hillel, and Seri Khoury. 2016. Near-Linear Lower Bounds for Distributed Distance Computations. Even in Sparse Networks.
[24] Mohsen Ghaffari and Bernhard Haeupler. 2016. Distributed Algorithms for Planar Networks II: Low-congestion Shortcuts, MST, and Min-Cut. In Proc. of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) 202–219.

[25] Mohsen Ghaffari and Jason Li. 2018. Improved Distributed Algorithms for Exact Shortest Paths. In Proc. of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC) 431–444.

[26] Mohsen Ghaffari and Merav Parter. 2017. Near-Optimal Distributed DFS in Planar Graphs. In 31st International Symposium on Distributed Computing (DISC 2017), Vol. 91. 211–2116. https://doi.org/10.4230/LIPIcs.DISC.2017.21

[27] Ofir Grossman, Seri Khoury, and Ami Paz. 2020. Improved Hardness of Approximation of Diameter in the CONGEST Model. In Proc. of 34th International Symposium on Distributed Computing (DISC) 19:1–19:16. https://doi.org/10.4230/LIPIcs.DISC.2020.19

[28] Bernhard Haeupler, D. Ellis Hershkowitz, and David Wajc. 2018. Round- and Message-Optimal Distributed Graph Algorithms. In Proc. of the 2018 ACM Symposium on Principles of Distributed Computing (PODC) 119–128. https://doi.org/10.1145/3212734.3212737

[29] Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. 2016. Near-Optimal Low-Congestion Shortcuts on Bounded Parameter Graphs. In Proc. of 30th International Symposium on Distributed Computing (DISC) 158–172.

[30] Bernhard Haeupler and Jason Li. 2018. Faster Distributed Shortest Path Approximations via Shortcuts. In Proc. of 32nd International Symposium on Distributed Computing (DISC) 33:1–33:14.

[31] Bernhard Haeupler, Jason Li, and Goran Zuzic. 2018. Minor Excluded Network Families Admit Fast Distributed Algorithms. In Proc. of the 2018 ACM Symposium on Principles of Distributed Computing (PODC) 465–474.

[32] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. 2016. A Deterministic Almost-tight Distributed Algorithm for Approximating Single-source Shortest Paths. In Proc. of the 48th Annual ACM Symposium on Theory of Computing (STOC) 489–498.

[33] Stephan Holzer and Roger Wattenhofer. 2012. Optimal Distributed All Pairs Shortest Paths and Applications. In Proc. of the 2012 ACM Symposium on Principles of Distributed Computing (PODC) 355–364.

[34] Chien-Chung Huang, Danupon Nanongkai, and Thatchaphol Saranurak. 2017. Distributed Exact Weighted All-Pairs Shortest Paths in $O(n^{5/3})$ Rounds. In 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS) 168–179.

[35] Yoichi Iwata, Tomoaki Ogasawara, and Naoto Ohsaka. 2018. On the Power of Treewidth of Real-World Graph Data. In ICDT (LIPIcs, Vol. 127) Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 12:1–12:18.

[36] Taisuke Izumi and Roger Wattenhofer. 2014. Time Lower Bounds for Distributed Distance Oracles. In Proc. of 18th International Conference on Principles of Distributed Systems (OPODIS) 60–75.

[37] Michal Katz, Nir A. Katz, and David Peleg. 2005. Distance Labeling Schemes for Well-separated Graph Classes. Discrete Applied Mathematics 145, 3 (2005), 384–402.

[38] Naoaki Kitamura and Taisuke Izumi. 2022. A Subquadratic-Time Distributed Algorithm for Exact Maximum Matching. IEEE Transactions on Information and Systems 105, 3 (2022), 634–645. https://doi.org/10.1587/transinf.2021edp7083

[39] Fabian Kuhn, Thomas Mosch Wright, and Roger Wattenhofer. 2006. The price of being near-sighted. In Proceedings of the seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). 1109557–1109666. https://doi.org/10.5555/1109557.1109666

[40] Christoph Lenzen and Boaz Patt-Shamir. 2013. Fast Routing Table Construction Using Small Messages: Extended Abstract. In Proc. of the 45th Annual ACM Symposium on Symposium on Theory of Computing (STOC) 381–390.

[41] Christoph Lenzen and Boaz Patt-Shamir. 2015. Fast Partial Distance Estimation and Applications. In Proc. of the 2015 ACM Symposium on Principles of Distributed Computing (PODC) 153–162.

[42] Christoph Lenzen and David Peleg. 2013. Efficient Distributed Source Detection with Limited Bandwidth. In Proc. of the 2013 ACM Symposium on Principles of Distributed Computing (PODC) 375–382.

[43] Jason Li. 2018. Distributed Treewidth Computation. arXiv (2018). arXiv:1805.10708

[44] Jason Li and Merav Parter. 2019. Planar Diameter via Metric Compression. In Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC) 152–163. https://doi.org/10.1145/3313276.3316358

[45] Zvi Lotker, Boaz Patt-Shamir, and Seth Pettie. 2015. Improved distributed approximate matching. Journal of the ACM (JACM) 62, 5 (2015), 1–17. https://doi.org/10.1145/2786755

[46] Silviu Maniu, Pierre Senellart, and Suraj Jog. 2019. An Experimental Study of the Treewidth of Real-World Graph Data. In ICDT (LIPIcs, Vol. 127) Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 12:1–12:18.

[47] Danupon Nanongkai. 2014. Distributed Approximation Algorithms for Weighted Shortest Paths. In Proc. of the 46th Annual ACM Symposium on Theory of Computing (STOC) 565–573.

[48] Merav Parter. 2020. Distributed Planar Reachability in Nearly Optimal Time. In Proc. of 34th International Symposium on Distributed Computing (DISC) 38:1–38:17. https://doi.org/10.4230/LIPIcs.DISC.2020.38

[49] David Peleg, Liam Roditty, and Elad Tal. 2012. Distributed Algorithms for Network Diameter and Girth. In Proc. of the 39th International Colloquium Conference on Automata, Languages, and Programming (ICALP) 660–672.

[50] Goran Zuzic, Gramoz Goranci, Mingqian Ye, Bernhard Haeupler, and Xiaorui Sun. 2022. Universally-Optimal Distributed Shortest Paths and Transshipment via Graph-Based IV-Oblivious Routing. In Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). 2549–2579. https://doi.org/10.1137/1.9781611977073.100