Distributed Computing manuscript No.
(will be inserted by the editor)

Distributed Computation and Reconfiguration in Actively Dynamic Networks

Othon Michail · George Skretas · Paul G. Spirakis

Received: date / Accepted: date

Abstract We study here systems of distributed entities that can actively modify their communication network. This gives rise to distributed algorithms that apart from communication can also exploit network reconfiguration to carry out a given task. Also, the distributed task itself may now require a global reconfiguration from a given initial network G_s to a target network G_f from a desirable family of networks. To formally capture costs associated with creating and maintaining connections, we define three edge-complexity measures: the *total edge activations*, the *maximum activated edges per round*, and the *maximum activated degree of a node*. We give (poly)$\log(n)$ time algorithms for the task of transforming any G_s into a G_f of diameter (poly)$\log(n)$, while minimizing the edge-complexity.

Our main lower bound shows that $\Omega(n)$ total edge activations and $\Omega(n/\log n)$ activations per round must be paid by any algorithm (even centralized) that achieves an optimum of $\Theta(\log n)$ rounds. We give three distributed algorithms for our general task. The first runs in $O(\log n)$ time, with at most $2n$ active edges per round, a total of $O(n \log n)$ edge activations, a maximum degree $n-1$, and a target network of diameter 2. The second achieves bounded degree by paying an additional logarithmic factor in time and in total edge activations. It gives a target network of diameter $O(\log n)$ and uses $O(n)$ active edges per round. Our third algorithm shows that if we slightly increase the maximum degree to polylog(n) then we can achieve $o(\log^2 n)$ running time.

Keywords distributed algorithms · dynamic networks · reconfiguration · transformation · polylogarithmic time · edge complexity

Declarations

Funding: EPSRC project "EP/P02002X/1 (Algorithmic Aspects of Temporal Graphs)

Conflicts of interest/Competing interests: Not applicable

Availability of data and material: Not applicable

Code availability: Not applicable

1 Introduction

1.1 Dynamic Networks

The *algorithmic theory of dynamic networks* is a relatively new area of research, concerned with studying the algorithmic and structural properties of networked systems whose structure changes with time. One way to classify dynamic networks is based on *who controls the network dynamics*. In *passively* dynamic networks the changes are external to the algorithm, in the sense that the algorithm has no control over them. Such dynamics are usually modeled by sequences of events determined by an *adversary scheduler*. This is for example the case

Supported in part by the NeST initiative and the EPSRC project "EP/P02002X/1 (Algorithmic Aspects of Temporal Graphs).". A preliminary version of the results in this paper has appeared in [25].

Othon Michail · George Skretas · Paul G. Spirakis ·
Department of Computer Science, University of Liverpool,
Ashton Street, Liverpool L69 3BX, United Kingdom
Tel.: +44 (0)151 795 4275

Paul G. Spirakis
Computer Engineering and Informatics Department, University of Patras, Greece

E-mail:
Othon.Michail@liverpool.ac.uk, G.Skretas@liverpool.ac.uk, P.Spirakis@liverpool.ac.uk

when the computing entities must operate in a dynamic environment, such as when being carried by a set of transportation units. In other applications, the entities can *actively* control the dynamics of their network, as is the case in mobile or reconfigurable robotics and peer to peer networks. *Hybrid* cases or cases of *partial control* are less studied (cf. [15] for a relevant study).

Another level of classification comes from *who controls the algorithm*. This gives rise to two main families of models. One is *fully centralized*, in which a central controller has global view of the system. In case of active network dynamics, the centralized algorithm typically designs a dynamic network by exploiting its full knowledge about the system in a way that aims to optimize some given objective function. If network dynamics are passive then the goal is typically to achieve some global computation task, like foremost journeys or dissemination, which may either be possible to compute *offline* under full information about the evolution of the network or required to compute *online* under limited or no knowledge about the future network structure. Similar objectives hold for the *fully distributed* case, in which every node in the network is an independent computing entity, like an automaton or Turing machine, typically equipped with computation and communication capabilities, and in the case of active dynamics, with the additional capability to locally modify the network structure, like *activating* a connection to a new neighbor or *eliminating* an existing connection. One may also consider *partial distributed control*, in which only k out of n nodes are occupied by computing entities, but again not much is known about this family of models.

1.2 An Actively Dynamic Distributed Model

In this paper, we consider an *actively dynamic and fully distributed* system. In particular, there are n computing entities starting from an *initial connected network* drawn from a family of initial networks. The entities are typically equipped with unique IDs, can compute locally, can communicate with neighboring entities, and can activate connections to new neighbors locally or eliminate some of their existing connections. All these take place in lock step through a standard synchronous message passing model, extended to include the additional operations of edge activations and deactivations within each round.

The goal is, generally speaking, to program all the entities with a distributed algorithm that can transform the initial network G_s into a *target network* G_f from a family of target networks. The idea is that starting from a G_s not necessarily having a good property, like small diameter, the algorithm will be able to "efficiently" reach a G_f satisfying the property. This gives rise to two main objectives, which in some cases might be possible to satisfy at the same time. One is to transform a given G_s into a desired target G_f and the other is to exploit some good properties of G_f in order to more efficiently solve a distributed task, like computation of a global function through information dissemination.

Even when edge activations are extremely local, meaning that an edge uv can only be activated if there exists a node w such that both uw and wv are already active, there is a straightforward algorithmic strategy that can successfully carry out most of the above tasks. In every round, all nodes activate all of their possible new connections, which corresponds to each node u connecting with all nodes v_i that were at distance 2 from u in the beginning of the current round. By a simple induction, it can be shown that in any round r the neighborhood of every node has size at least 2^r, which implies that a clique K_n is formed in $O(\log n)$ rounds. Such a clique can then be used for global computations, like electing the maximum UID as a leader, or for transforming into any desired target network G_f through eliminating the edges in $E(K_n) \setminus E(G_f)$. All these can be performed within a single additional round.

Even though sublinear global computation and network-to-network transformations are in principle possible through the *clique formation* strategy described above, this algorithmic strategy still has a number of properties which would make it impractical for real distributed systems. As already highlighted in the literature of dynamic networks, (i.e., [20]), activating and maintaining a connection does not come for free and is associated with a cost that the network designer has to pay for. Even if we uniformly charge 1 for every such active connection, the clique formation incurs a cost of $\Theta(n^2)$ total edge activations in the worst case and always produces instances (e.g., when K_n is formed) with as many as $\Theta(n^2)$ active edges in which all nodes have degree $\Theta(n)$.

Our goal in this work is to formally define such cost measures associated with the structure of the dynamic network and to give improved algorithmic strategies that maintain the time-efficiency of clique formation, while substantially improving the edge complexity as defined by those measures. In particular, we aim at minimizing the edge complexity, given the constraint of (poly)logarithmic running time. Observe at this point that without any restriction on the running time, a standard distributed dissemination solely through message passing over the initial network, would solve global computation without the need to activate any edges. However, linear running times are considered insuffi-

cient for our purposes (even when the goal is to solve traditional distributed tasks). Moreover, strategies that do not modify the input network cannot be useful for achieving network-to-network transformations.

1.3 Contribution

We define three cost measures associated with the edge complexity of our algorithms. One is the *total number of edge activations* that the algorithm performed during its course, the second one is the *maximum number of activated edges in any round* by the algorithm, and the third one is the *maximum activated degree of a node in any round*, where the maximum activated degree of a node is defined only by the edges that have been activated by the algorithm.

Our ultimate goal in this paper is to give *(poly)logarithmic time* algorithms which, starting from *any* connected network G_s, transform G_s into a G_f of *(poly)logarithmic diameter* and at the same time *elect a unique leader*. Such algorithms can then be composed with any algorithm B that assumes an initial network of (poly)logarithmic diameter and has access to a unique leader and unique ids. In case of a static network algorithm B, this for example yields (poly)logarithmic time information dissemination and computation of any global function on inputs. In case of an actively dynamic network algorithm B, it gives (poly)logarithmic time transformation into any target network from a given family which depends on restrictions related to the edge complexity.

We restrict our focus on *deterministic* algorithms, that is, the computational entities do not have access to any random choices. Moreover, our algorithms never break the connectivity of the network of active edges as this would result in components that could never be reconnected based on the permissible edge activations. Temporary disconnections within a round may be permitted but can always be avoided by first activating all new edges and then deactivating any edges for the current round.

There is a clear tradeoff between time and edge complexity and we formally capture that with the lower bounds presented in Section 7. In particular, we first prove that $\Omega(\log n)$ is a lower bound on time following from an upper bound of 2 on the distance of new connections and the $\Theta(n)$ worst-case diameter of the initial network. Then we give an $\Omega(n)$ lower bound on total edge activations and $\Omega(n/\log n)$ activations per round for any centralized algorithm that achieves an optimal $\Theta(\log n)$ time. Our main lower bound is a total of $\Omega(n \log n)$ total edge activations that any logarithmic time deterministic distributed *comparison based* algorithm must pay. This is in contrast to the $\Theta(n)$ total edges that would be sufficient for a centralized algorithm and is due to the distributed nature of the systems under consideration.

We then proceed to our main positive results. In particular, we give three algorithms for transforming *any* initial connected network G_s into a network G_f of (poly)logarithmic diameter and at the same time electing a unique leader. Each of these algorithms makes a different contribution to the time vs. edge complexity trade-off. All of our main algorithms are built upon the following general strategy. For each of them, we define a different *gadget network* and the algorithms are developed in such a way that they always satisfy the following invariants. In any round of an execution, the network is the union of committees being such gadget networks of varying sizes and some additional edges including the initial edges and other edges used to join the committees. Initially, every node forms its own committee and the algorithms progressively merge pairs or larger groups of committees based on the rule that the committee with the greater UID dominates. If properly performed, this ensures that eventually only one committee remains, namely, the committee of the node u_{max} with maximum UID in the network. The diameter of all our gadgets is (poly)logarithmic in their size, which facilitates quick merging and ensures that the final committee of u_{max} satisfies the (poly)$\log(n)$ diameter requirement for G_f. The algorithms also ensure that, by the time the committee of u_{max} is the unique remaining committee, u_{max} is the unique leader elected.

Our algorithms must achieve (poly)logarithmic time and they do so by satisfying the invariant that surviving committees always grow exponentially fast. This growth is *asynchronous* in our algorithms for the following reason. In a typical configuration (of a phase) the graph of mergings forms a spanning forest F of committees such that any tree T in F is rooted at the committee that will eventually consume all committees in $V(T)$. Given that those trees may have different sizes (even up to $V(T) = \Theta(n)$), the rounds in which various committees finish merging may be different, but we can still show that their amortized growth is exponential.

Our first algorithm, called `GraphToStar` and presented in Section 4, uses a star network as a gadget. Its running time is $O(\log n)$ and it uses at most $2n$ active edges per round and an optimal total of $O(n \log n)$ edge activations. The target network G_f that it outputs is a spanning star, thus, achieving a final diameter of 2.

Our second algorithm, called `GraphToWreath` and presented in Section 5, uses as a gadget a graph we call a *wreath* which is the union of a ring and a complete binary tree spanning the ring. The main improve-

ment compared to `GraphToStar` is that it maintains a bounded maximum degree throughout its course (given a bounded-degree G_s). It does this at the cost of increasing the running time to $O(\log^2 n)$ and the number of total edge activations to $O(n \log^2 n)$. The active edges per round remain $O(n)$. The target network G_f that it outputs is a complete binary spanning tree (after deleting the original edges and the spanning ring), thus, the algorithm achieves a final diameter of $O(\log n)$.

Our third algorithm, called `GraphToThinWreath` and presented in Section 6, shows that if we slightly increase the maximum degree to polylog(n) then we can achieve a running time of $o(\log^2 n)$ (more precisely, $O(\log^2 n / \log\log^k n)$, for some constant $k \geq 1$).

If our model can be compared to models from the area of overlay networks construction (see Section 2 for a discussion on this matter), then `GraphToWreath` is, to the best of our knowledge, the first deterministic bounded-degree $O(\log^2 n)$-time algorithm and `GraphToThinWreath` is the first deterministic polylog(n)-degree $o(\log^2 n)$-time algorithm for the problem of transforming any connected G_s into a polylog(n) diameter G_f.

2 Further Related Work

Temporal Graphs. The algorithmic study of temporal graphs was initiated by Berman [9] and Kempe *et al.* [17], who studied a special case of temporal graphs in which every edge can be available at most once. The problem of designing a cost-efficient temporal graph satisfying some given connectivity properties was introduced in [21]. The design task was carried out by an offline centralized algorithm starting from an empty edge set. Subsequent work [14], motivated by epidemiology applications, considered the centralized algorithmic problem of re-designing a given temporal graph through edge deletions in order to end up with a temporal graph with bounded temporal reachability, thus keeping the spread of a disease to a minimum. Our work is related to the temporal network (re-)design problem but our model is fully distributed, allows for both edge activations and deletions, and our families of target networks are different than those considered in the above papers.

Distributed Computation in Passively Dynamic Networks. Probably the first authors to consider distributed computation in passively dynamic networks were Angluin *et al.* [4–6]. Their population protocol model, considered originally the computational power of a population of n finite automata which interact in pairs passively either under an eventual fairness condition or under a uniform random scheduling assumption. A variant of population protocols in which the automata can additionally create or destroy connections between them was introduced in [22, 26]. It was shown that in that model, called network constructors, complex spanning networks can be created efficiently despite the computational weakness of individual entities. The closest to our approach from this area is [27], in which the authors showed how to transform any connected initial network into a spanning line which can then be exploited to achieve global computation on input values and termination. The main difference though is that in all these models pairwise interactions are chosen asynchronously by a scheduler, and connections can be created between any pair of nodes during their interaction independently of the current network structure and the distance between them.

Other papers [18, 23, 28] have studied distributed computation in worst-case dynamic networks using a traditional message-passing model and typically operating through local broadcast in the current neighborhood. Our communication model is closer to those models but network dynamics there are always passive and their main goal has been to revisit the complexity of classical distributed tasks under a worst-case adversarial network.

Finally the work by Casteigts *et al.* [12] is a unifying framework for different dynamic network models and our model falls more closely under the umbrella of the graph-centric evolution discussed by the paper.

Construction of Overlay Networks. There is a rich literature on the distributed construction of overlay networks. A typical assumption is that there is an overlay (active) edge from a node u to a node v in a given round iff u has obtained v's UID through a message. Without further restrictions, the overlay in round r would always correspond to the union of r consecutive transitive extensions starting from the original edge set. The main restriction imposed in the relevant literature is a polylogarithmic (in bits) communication capacity per node per round, which also implies that in every round $O(\log n)$ new overlay connections per node are permitted.

Our model and results, even though different in motivation, in the complexity measures considered, and in the restrictions we impose, appear to have similarities with some of the developments in this area. Unlike our work, where our complexity measures are motivated by the cost of creating and maintaining physical or virtual connections, the algorithmic challenges in overlay networks are mainly due to restricting the communication capacity of each node per round to a polylogarithmic total number of bits.

Research in this area started with seminal papers such as Chord of Stoica *et al.* [31] and the Skip graphs of Aspnes and Shah [7]. Probably the first authors to have considered the problem of constructing an overlay network of logarithmic diameter were Angluin *et al.* [3]. Their algorithm is randomized with $O((d+W)\log n)$ running time w.h.p., where W is the maximum size of a unique UID. Then Aspnes and Wu [8] gave a randomized $O(\log n)$ time algorithm for the special case in which the initial network has outdegree 1. A very recent work by Götte *et al.* [16] has improved the upper bound of [3] to $O(\log^{3/2} n)$, w.h.p. It is a randomized algorithm which uses a core deterministic procedure that has some similarities to our algorithmic strategy of maintaining and merging committees (called "supernodes" there) whose size increases exponentially fast. Their model keeps the polylogarithmic restriction on communication and the polylogarithmic maximum degree.

To the best of our knowledge, the only previous deterministic algorithm for the problem is the one by Gmyr *et al.* [15]. Our algorithmic strategies appear to have some similarities to their "Overlay Construction Algorithm", which in their work is used as a subroutine for monitoring properties of a passively dynamic network. Unlike our model, their model is hybrid in the sense that algorithms have partial control over the connections of an otherwise passively dynamic network. Due to using different complexity measures and restrictions it is not totally clear to us yet whether a direct comparison between them would be fair. Still, we give some first observations. Their algorithm has the same time complexity, i.e., $O(\log^2 n)$, with our `GraphToWreath` algorithm, while our `GraphToStar` algorithm achieves $O(\log n)$ and our `GraphToThinWreath` $o(\log^2 n)$. Their overlays appear to maintain $\Theta(n \log n)$ active connections per round, while our algorithms maintain $O(n)$. Their maximum active degree is polylogarithmic, the same as `GraphToThinWreath`, while `GraphToStar` uses linear and `GraphToWreath` always bounded by a constant. Their model restricts the communication capacity of every node to a polylogarithmic number of bits per round, whereas we do not restrict communication.

Scheideler and Setzer [30] recently studied the (centralized) computational complexity of computing the optimum graph transformation and gave **NP**-hardness results and a constant-factor approximation algorithm for the problem.

Programmable Matter. There is a growing interest in studying the algorithmic foundations of systems that can change their physical properties through local reconfigurations [1, 2, 10, 13, 24]. A prominent such property is changing their shape. Typical examples of systems in this area are reconfigurable robotics, swarm robotics, and self-assembly systems [11, 19, 29]. In most of these settings, modification of structure can be represented as a dynamic network, usually called *shape*, with additional geometric restrictions coming from the shape and the local reconfiguration mechanism of the entities. The goal is to transform a given initial shape into a desired target shape through a sequence of valid local moves. Our network transformation problem can be viewed as a non-geometric abstraction of these geometric transformation problems. Apart from being motivated by this area, we also hope that the abstract algorithmic principles of network reconfiguration might promote our understanding of the geometrically constrained cases.

3 Preliminaries

3.1 Model

An actively dynamic network is modeled in this work by a temporal graph $D = (V, E)$, where V is a static set of n nodes and $E \subseteq \binom{V}{2} \times \mathbb{N}$ is a set of undirected time-edges. In particular, $E(i) = \{e : (e, i) \in E\}$ is the set of all edges that are *active* in the temporal graph at the beginning of round i. Since V is static, $E(i)$ can be used to define a snapshot of the temporal graph at round i, which is the static graph $D(i) = (V, E(i))$.

The temporal graph D of an execution is generated by local operations performed by the nodes of the network, starting from an initial graph $G_s = D(1)$. Throughout this paper, G_s is assumed to be connected. A node u can *activate an edge* with node v in round i, if $uv \notin E(i)$ and there exists a node w such that both uw and wv are active at the beginning of round i. A node u can *deactivate an edge* with node v in round i, provided that $uv \in E(i)$. An active edge remains active indefinitely unless a node that is incident to that edge deactivates it. There is at most one active edge between any pair of nodes, that is multiple edges are not allowed. If a node attempts to activate an edge which is already active, the action has no effect and the edge remains active; similarly for deactivating inactive edges. Moreover, if a node u decides to activate an edge with a node v in round i and v decides to activate an edge with u in the same round, then only one edge is activated between them. In case u and v disagree on their decision about edge uv, then their actions have no effect on uv. We define $E_{ac}(i)$ as the set of all edges that were activated in round i and $E_{dac}(i)$ as the set of all edges that were deactivated in round i. Then $E(i+1) = (E(i) \cup E_{ac}(i)) \setminus E_{dac}(i)$.

We define set $N_1^i(u)$ of node u, where $v \in N_1^i(u)$ iff $uv \in E(i)$ which means that set $N_1^i(u)$ contains the neighbors of node u in round i. Additionally, set $N_2^i(u)$ of node u, where $w \in N_2^i(u)$ iff there exists $v \in V$ s.t. $v \in N_1^i(u)$ and $v \in N_1^i(w)$ and $w \notin N_1^i(u)$. That is, set $N_2^i(u)$ of node u in round i contains the nodes at distance 2 which we will refer to as *potential neighbors*. We will omit the i index for rounds, when clear from context.

Each node $u \in V$ is identical to every other node v but for the unique identifier *(UID)* that each node possesses. Each node u starts with a UID that is drawn from a namespace $\mathcal{U}$. The maximum UID is represented by $O(\log n)$ bits. An algorithm is called *comparison based* if it manipulates the UIDs of the network using comparison operations $(<, >, =)$ only. All of the algorithms and lower bounds presented in this paper are comparison based.

The nodes represent agents equipped with computation, communication, and edge-modification capabilities and they operate in synchronous rounds. In each round all agents perform the following actions in sequence and in lock step: send messages to their neighbors, receive messages from their neighbors, activate edges with potential neighbors, deactivate edges with neighbors, update their local state.

We note that a node may choose to send a different message to different neighbors in a round and that the time needed for internal computations is assumed throughout to be $O(1)$. We do not impose any restriction on the size of the local memory of the agents, still the space complexity of our algorithms is within a reasonable polynomial in n.

3.2 Problem Definitions and Performance Measures

In this paper, we are mainly interested in the following problems.

Leader Election. Every node u in graph $D = (V, E)$ has a variable $status_u$ that can be set to a value in {Follower, Leader}. An algorithm A solves leader election if the algorithm has terminated and exactly one node has its status set to Leader while all other nodes have their status set to Follower.

Token Dissemination. Given an initial graph $D = (V, E)$ where each node $u \in V$ starts with some unique piece of information (token), every node $u \in V$ must terminate while having received that unique piece of information from every other node $v \in V \setminus \{u\}$. W.l.o.g. we will consider that unique information to be the UID of each node throughout the paper.

Depth-d Tree. Given any initial graph G_s from a given family, the distributed algorithm must reconfigure the graph into a target graph G_f, such that G_f is a rooted tree of depth d with a unique leader elected at the root.

Apart from studying the running time of our algorithms, measured as their worst-case number of rounds to carry out a given task, we also introduce the following edge complexity measures.

Total Edge Activations. The total number of edge activations of an algorithm is given by $\sum_{i=1}^{T} |E_{ac}(i)|$, where T is the running time of the algorithm.

Maximum Activated Edges. It is defined as $\max_{i \in [T]} |E(i) \setminus E(1)|$, that is, equal to the maximum number of active edges of a round, disregarding the edges of the initial network.

Maximum Activated Degree. The maximum degree of a round, if we again only consider the edges that have been activated by the algorithm. Let $\Delta(G)$ denote the maximum degree of a graph G. Then, formally, the maximum activated degree is equal to $\max_{i \in [T]} \Delta(D(i) \setminus D(1))$, where the graph difference is defined through the difference of their edge sets.

In this paper, instead of measuring the maximum activated degree we will focus on preserving the maximum degree of input networks from specific families. For example, one of our algorithms solves the Depth-d Tree problem on any input network and, if the input network has bounded degree, then it guarantees that the degree in any round is also bounded.

3.3 Basic Subroutines

We will now provide algorithms that transform initial graphs into graphs with small diameter and which will be used as subroutines in our general algorithms. The first called TreeToStar transforms any initial rooted tree graph into a spanning star in $O(\log n)$ time with $O(n \log n)$ total edge activations and $O(n)$ active edges per round, provided that the nodes have a sense of orientation on the tree (i.e., can distinguish which of their neighbors is "closer" to the root of the tree). In every round, each node activates an edge with the potential neighbor that is its grandparent and deactivates the edge with its parent. This process keeps being repeated by each node until they activate an edge with the root of the tree.

Proposition 1 *Let T be any tree rooted at u_0 of depth d. If the nodes have a sense of orientation on the tree,*

then algorithm TreeToStar transforms T into a spanning star centered at u_0 in $\lceil \log d \rceil \leq \log n$ rounds. TreeToStar has at most $2n-3$ active edges per round.

Our next algorithm called LineToCompleteBinaryTree transforms any line into a binary tree in $O(\log n)$ time, with $O(n \log n)$ total edge activations, $O(n)$ active edges per round and the degree of each node is at most 4, provided that the nodes have a common sense of orientation. In each round, each node activates an edge with its grandparent and afterwards it deactivates its edge with its parent. This process keeps being repeated by each node until they activate an edge with the root of the tree or if their grandparent has 2 children.

Proposition 2 *Let T be any line rooted at u_0 of diameter d. If the nodes have a sense of orientation on the line, then algorithm LineToCompleteBinaryTree transforms T into a binary tree centered at u_0 in $\lceil \log d \rceil \leq \log n$ time. LineToCompleteBinaryTree has at most $2n-3$ active edges per round, $n \log n$ total edge activations and bounded degree equal to 3.*

3.4 General Strategy for Depth-d Tree

All algorithms developed in this paper solve the Depth-d Tree problem starting from any connected initial network G_s from a given family. Our aim is to always achieve this in (poly)logarithmic time while minimizing some of the edge-complexity parameters. There is a natural trade-off between time and edge complexity and each of our algorithms makes a different contribution to this trade-off. In particular, by paying for linear degree, our first algorithm manages to be optimal in all other parameters. If we instead insist on bounded degree, then our second algorithm shows that we can still solve Depth-d Tree within an additional $O(\log n)$ factor both in time and total edge activations. Finally, if the bound on the degree is slightly relaxed to (poly)$\log(n)$, our third algorithm achieves $o(\log^2 n)$ time.

All three algorithms are built upon the same general strategy that we now describe. For each of them we choose an appropriate gadget network, which has the properties of being "close" to the target network G_f to be constructed and of facilitating efficient growth. For example, the G_f of our first algorithm is a spanning star and the chosen gadget is a star graph, while the G_f of our second algorithm is a complete binary tree and the chosen gadget is the union of a ring and a complete binary tree spanning that ring (called a *wreath*).

Our algorithms satisfy the following properties. The nodes are always partitioned into committees, where each committee is internally organized according to the corresponding gadget network of the algorithm and has a unique leader, which is the node with maximum UID in that committee. Initially, every node forms its own trivial committee and committees increase their size by competing with nearby committees. In particular, committees select and, if possible, merge with the maximum-UID committee in their neighborhood. Prior to merging, such selections may give rise to pairs of committees, in which case merging is immediate, but also to rooted trees of committees where all selections are oriented towards the root and merging has to be deferred. In the latter case, the winning committee will eventually be the root of the tree, at which point all other committees of the tree will have merged to it. In all cases, merging must be done in such a way that the gadget-like internal structure of the winning committee is preserved. This growth guarantees that eventually there will be a single committee spanning the network. At that point, the leader of that committee (which is always the node with maximum UID in the network) is an elected unique leader. Moreover, the gadget-like internal structure of that committee can be quickly transformed into the desired target network, due to the by-design close distance between them. For example, in the algorithm forming a star no further modification is required, while in the algorithm forming a complete binary tree, a ring is eliminated from a wreath so that only the tree remains.

Our algorithms are designed to operate in asynchronous phases, with the guarantee that in every phase pairs of committees merge and trees of committees halve their depth. This can be used to show that in all our algorithms a single committee will remain within $O(\log n)$ phases. Each phase lasts a number of rounds which is within a constant factor of the maximum diameter of a committee involved in it, which is in turn upper bounded by the diameter of the final spanning committee. The latter is always equal to the diameter of the chosen gadget as a function of its size. The total time is then given by the product of the number of phases and the diameter of the chosen gadget. For example, in our first algorithm the gadget is a star and the running time (in rounds) is $O(1) \cdot O(\log n)$, in our second algorithm the gadget is a wreath of diameter $O(\log n)$ and the running time is $O(\log n) \cdot O(\log n) = O(\log^2 n)$, while in our third algorithm the gadget is a modified wreath, called ThinWreath, of diameter $o(\log n)$ and the running time is $o(\log n) \cdot O(\log n) = o(\log^2 n)$. Given that every node activates at most one edge per round, the total number of edge activations of our algorithms is within a linear factor of their running time.

4 An Edge Optimal Algorithm for General Graphs

Our first algorithm, called `GraphToStar`, solves the Depth-d Tree problem, for $d = 1$. In particular, by using a star gadget it transforms any initial graph G_s into a target spanning star graph G_f. Its running time is $O(\log n)$ and it uses an optimal number of $O(n \log n)$ total edge activations and $O(n)$ active edges per round. Optimality is established by matching lower bounds, presented in Section 7.

Algorithm GraphToStar

Each committee $C(u)$ is a star graph where the center node u is the leader of the committee and all other nodes are followers. The leader node of each committee is the node with the greatest UID in that committee. The UID of each committee is defined by the UID of that committee's leader. The winning committee in the final graph, denoted $C(u_{max})$, is the one with the greatest UID in the initial graph. Every node starts as a leader and forms its own committee as a single node. The original edges of G_s are assumed to be maintained until the last round of the algorithm and the nodes can always distinguish them. The algorithm proceeds in phases, where in every phase each committee $C(u)$ executes in one of the following modes, always executing in selection mode in phase 1.

- **Selection:** If $C(u)$ has a neighboring committee $C(z)$ such that $UID_z > UID_u$ and $C(z)$ is not in *pulling mode*, then, from its neighboring committees not in pulling mode, $C(u)$ *selects* the one with the greatest UID; call the latter $C(v)$. It does this, by u first activating an edge e_1 with a potential neighbor in $C(v)$. Then u activates an edge with v, deactivates the previous edge e_1, and $C(u)$ enters either the merging or pulling mode. In particular, if $C(v)$ did not select, then $C(u)$ and $C(v)$ form a pair and $C(u)$ enters the merging mode. If on the other hand $C(v)$ selected some $C(w)$, then $C(u)$ enters the pulling mode. Otherwise, $C(u)$ did not select. If $C(u)$ was selected then it enters the waiting mode, else it remains in the selection mode. If $C(u)$ has no neighboring committees, then it enters the termination mode.
- **Merging:** Given that in the previous phase the leader of $C(u)$ activated an edge with the leader of $C(v)$, each follower x in $C(u)$ activates the edge xv and deactivates the edge xu. The result is that $C(u)$ and $C(v)$ have merged into committee $C(v)$, which remains a star rooted at v now spanning all nodes in $V(C(u)) \cup V(C(v))$. Therefore, $C(u)$ does not exist any more.
- **Pulling:** Given that in the previous phase the leader of $C(u)$ activated an edge with the leader of $C(v)$ and the leader of $C(v)$ activated an edge with the leader of $C(w)$, u activates uw, deactivates uv, and $C(u)$ remains in pulling mode. If, instead, the leader of $C(v)$ did not activate in the previous phase, then $C(u)$ enters the merging mode. On the other hand, given that in the previous phase the leader of $C(u)$ activated an edge with the leader of $C(v)$ and in the current phase, committee $C(v)$ does not exist anymore, this means that v is currently in some committee $C(w)$, and u activates uw and $C(u)$ enters the merging mode.
- **Waiting:** If $C(u)$ has no neighboring committees, $C(u)$ enters the termination mode. If in the previous phase no committee $C(v)$ activated an edge with u, then $C(u)$ enters the selection mode. Otherwise $C(u)$ remains in the waiting mode.
- **Termination:** $C(u)$ deactivates every edge in $E(G_s) \setminus E(C(u))$. In particular, each follower x in $C(u)$ deactivates all active edges incident to it but xu.

Correctness

Lemma 1 *Algorithm* `GraphToStar` *solves Depth-1 Tree.*

Proof It suffices to prove that in any execution of the algorithm, one committee eventually enters the termination mode and that this committee can only be $C(u_{max})$. If this holds, then by the end of the termination phase $C(u_{max})$ forms a spanning star rooted at u_{max} and u_{max} is the unique leader of the network. This satisfies all requirements of Depth-1 Tree.

A committee *dies* (stops existing) only when it merges with another committee by entering the merging mode. First observe that there is always at least one *alive* committee. This is $C(u_{max})$, because entering the merging mode would contradict maximality of u_{max}. We will prove that any other committee eventually dies or grows, which due to the finiteness of n will imply that eventually $C(u_{max})$ will be the only alive committee.

In any phase, but the last one which is a termination phase, it holds that every alive committee $C(u)$ is in one of the selection, merging, pulling, and waiting modes. If $C(u)$ is in the merging mode, then by the end of the current phase it will have died by merging with another committee $C(v)$. It, thus, remains to argue about committees in the selection, pulling, and waiting modes.

We first argue about committees in the pulling mode. Denote their set by $\mathcal{C}_{pull}$. Observe that, in any

given phase, the committees in pulling mode form a forest F, where each $C(u) \in \mathcal{C}_{pull}$ belongs to a pulling tree T of F. Any such pulling tree mimicks the execution of the TreeToStar algorithm (from Proposition 1) on the leaders of committees $C(u)$ and satisfies the invariant that its root committee C_r is always in the waiting mode and C_r's children are in the merging mode. In every phase, C_r's children merge with C_r and their children become the new children of C_r and enter the merging mode. It follows that all non-root committees in T will eventually merge with C_r. Thus, all committees in pulling mode eventually die.

It remains to argue about committees in the selection and waiting modes. We start from the waiting mode. Any committee $C(u)$ in waiting mode is a root of either a pulling tree in the forest F or of a star of committees in which all leaf-committees are merging with $C(u)$. In both cases, $C(u)$ eventually exits the waiting mode and enters the selection mode. This happens as soon as all other committees in its pulling tree or star have merged to it, thus $C(u)$ has grown upon its exit.

Now, a committee $C(u)$ in the selection mode can enter any other mode. As argued above, if it enters the merging or pulling modes it will eventually die and if it enters the waiting mode it will eventually grow. Thus, it suffices to consider the case in which it remains in the selection mode indefinitely. This can only happen if all current and future neighboring committees of $C(u)$, including the ones to eventually replace neighbors in pulling mode, have a UID smaller than UID_u. But each of these must have selected a neighboring $C(w)$, such that $UID_w > UID_u$, otherwise it would have selected $C(u)$. Any such selection results in $C(w)$ (or a z, such that $UID_z > UID_w$ in case w belongs to a tree) becoming a neighbor of $C(u)$, thus contradicting the indefinite local maximality of UID_u. □

Time Complexity

Let us move on to proving the time complexity of our algorithm. At the beginning, we are going to ignore the number of rounds within a phase, and we are just going to study the maximum number of phases before a single committee is left. We define $|C(u)_s|$ to be the *size* of committee $C(u)$ in phase s, which is the number of nodes in committee $C(u)$ in phase s.

Lemma 2 *Consider committee $C(u)$ that is in waiting mode between phases s and $s+j$. If the size of every committee in phase s is at least 2^k, then the size of committee $C(u)$ once it enters the selection mode in phase $s+j+1$ is at least 2^{k+j-2}.*

Proof Any committee $C(u)$ in waiting mode is a root of either (i) a pulling tree in the forest F or (ii) a star of committees in which all leaf-committees are merging with $C(u)$.

For case (i): root committee $C(u)$ is always in waiting mode and every other committee $C(v)$ of T is either in pulling or merging mode. It follows that all non-root committees $C(v)$ in the pulling tree will eventually merge with $C(u)$ in some phase $s+j$. W.l.o.g. assume that every committee $C(v)$ that belongs to the pulling tree T entered pulling or merging mode in phase s and every committee $C(v)$ will have merged with committee $C(v)$ by phase $s+j$. Every committee $C(v)$ will stay in pulling mode for $i < j$ phases and in merging mode for 1 phase. Consider the leaders v of every committee $C(v)$ and note that while in pulling mode, the leaders are mimicking the execution of the TreeToStar algorithm, where the leader of $C(u)$ is the root of the tree, and the leaders of $C(v)$ are the non-root nodes of the tree. We know by Proposition 1, that the running time of the algorithm is $\log d$, where d is the depth of the tree. Thus, if every committee $C(v)$ enters the pulling mode in phase s and the last committee $C(v)$ to exit the pulling mode is in phase $s+i$, $s+i-s = \log d \implies i = \log d$. This means that the depth of tree T is 2^i. Since the depth of the pulling tree T is 2^i, the tree T must contain at least 2^i committees. Additionally note that after the last committee $C(v)$ exits the pulling mode is in phase $s+k$, in phase $s+k+1$ it enters the merging mode and in phase $s+k+2$ every committee $C(v)$ has merged with committee $C(u)$. Thus, $s+i+2 = s+j \implies i = j-2$ and the size of $C(u)$ in phase $s+j+1$ is $|C(u)_{s+j+1}| \geq 2^k * 2^i = 2^{k+j-2}$.

For case (ii): root committee $C(u)$ is in waiting mode and has at least one leaf committee in phase s. After the leaf committee merges in 1 phase, committee $C(u)$ has size $|C(u)_{s+1}| \geq |C(u)_s| + |C(u)_s| \geq 2^k + 2^k = 2^{k+1}$. □

Lemma 3 *If committee $C(u)$ stays in the selection mode for $p \geq 4$ consecutive phases, then $C(u)$ has a neighboring committee $C(v) \in \mathcal{C}_{pull}$ that belongs to a pulling tree T for at least p phases.*

Proof Let us assume that committee $C(u)$ stays in the selection mode for $p \geq 4$ consecutive phases while having a neighbor $C(v)$ that does not belong to pulling tree T.

- If $C(v)$ does not belong to a pulling tree in phase k, then it cannot be in pulling mode.
- If $C(v)$ is in selection mode in phase k and $C(v)$ does not select $C(u)$ and $C(u)$ does not select $C(v)$, then $C(v)$ has a neighbor $C(w)$ where $UID_w > UID_v > UID_u$ and $C(v)$ selected $C(w)$. Then $C(v)$ enters the merging mode in phase $k+1$

and gets merged with $C(w)$. In phase $k+2$ committee $C(w)$ becomes a neighbor of $C(u)$ and $C(w)$ enters the selection mode. Therefore, since $UID_w > UID_u$, $C(u)$ would select $C(w)$ in phase $k+2$, and enter either the pulling or merging mode. Thus, a contradiction.
– If $C(v)$ is in waiting mode in phase k, it cannot be the root of a pulling tree, and is the root of a star. Therefore in phase $k+1$ it will enter the selection mode and based on the analysis of the previous paragraph, in phase $k+3$ $C(u)$ will exit the selection mode. Thus, a contradiction.

□

Lemma 4 *Let us assume that the minimum size of a committee in phase s is 2^k. If committee $C(u)$ stays in the selection mode from phase s to phase $s+p$, where $p \geq 4$, then in phase $s+p+1$ it will select or get selected by a committee $C(v)$ of size at least 2^{k+p-2}.*

Proof From Lemma 3 it follows that, since $C(u)$ is in the selection mode for at least 4 phases, there exists a neighbor $C(v)$ that belongs to a pulling tree T. W.l.o.g. assume that $C(w)$ is the root of the pulling tree T and $C(w)$ has been in waiting mode between phases s and $s+p$. Note also that in phase $s+p+1$, committees $C(u)$ and $C(w)$ are neighboring committees and both are in selection mode. Thus, $C(u)$ will exit the selection mode in phase $s+p+1$, because either $C(u)$ will select $C(w)$ or $C(w)$ will select $C(u)$. Since $C(w)$ was in waiting mode for p phases, the size of $C(w)$ is at least 2^{k+p-2} (based on Lemma 2). □

Lemma 5 *Assume that the minimum size of every committee in phase s is 2^k and that every committee will have exited the selection mode in phase $s+p$ at least once. The size of all winning committees (committees that still exist) in phase $s+p+1$ is at least 2^{k+p-2}.*

Proof Trivially, if $p \leq 4$ the winning committee has size at least 2^{k+1} in phase $p+1$ since it has merged with at least one other committee. From Lemma 4 it follows that if $p \geq 4$ the winning committee between $C(w)$ and $C(u)$ will have size at least 2^{k+p-2} in phase $s+p+1$. □

Lemma 6 *After $O(\log n)$ phases, there is only a single committee left in the graph.*

Proof We trivially assume that committee $C(u_{max})$ has size $|C(u_{max})_1| = 1$ in phase 1. Based on Lemma 5, after $O(\log n)$ phases, $C(u_{max})$ has size $|C(u_{max})_{O(\log n)}| \geq 2^{1+O(\log n)-c} \geq 2^{\log n} \geq n$. Therefore, committee $C(u_{max})$ must contain every single node of G. □

Lemma 7 *Each phase consists of at most 2 rounds.*

Proof Based on the description of the algorithm, the selection phase lasts 2 rounds and the rest of the phases last 1 round. □

Edge Complexity

It is very simple to prove the edge complexity for the algorithm. Note that in each round i each node activates at most 1 edge and based on Lemma 6 the algorithm runs for $O(\log n)$ phases which means that there are $O(n \log n)$ total edge activations. Furthermore, if a node had activated an edge u in round i, and it activates another edge v in round $i+1$, then it deactivates edge u. Therefore, each node cannot have more than 2 active edges that it has activated itself at any time and since we have n nodes in the network, there can ever be at most $2n$ active edges per round. Since the structure of every committee is a star, the maximum activated degree is $O(n)$.

Theorem 1 *For any initial connected graph G_s, the* `GraphToStar` *algorithm solves the Depth-1 Tree problem in $O(\log n)$ time with at most $O(n \log n)$ total edge activations, $O(n)$ active edges per round and $O(n)$ maximum activated degree.*

5 Minimizing the Maximum Degree on General Graphs

In this section we will create an algorithm that minimizes the maximum activated degree to a constant but has $O(\log^2 n)$ running time and $O(n \log^2 n)$ total edge activations.

For this algorithm, our committees must have at least $\Omega(\log n)$ diameter in order to have a constant degree and therefore merging two different committees in constant time while keeping a specific structure proves to be complicated. The new gadget of our committees is going to be a graph we call *wreath*. A wreath graph is a graph that has both a ring subgraph and a complete binary tree subgraph. We are going to use the edges of the ring subgraph to merge committees and the binary tree subgraph to exchange information between the nodes of the graph. First, let us define the structure of the wreath graph.

Definition 1 (Wreath graphs) A graph $D = (V, E)$ belongs to the class of *wreath* graphs if it has two subgraphs $D_r = (V, E_r)$ and $D_b = (V, E_b)$, where D_r belongs to the class of ring graphs, D_b belongs to the class of complete binary tree graphs, and $E = E_r \cup E_b$.

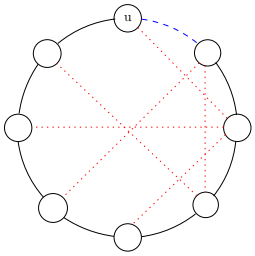

Fig. 1 A wreath graph with 8 nodes. The ring subgraph consists of the normal (black) and dashed (blue) edges. The complete binary tree consists of the dotted (red) and dashed (blue) edges. Node u is the root of the complete binary tree.

The $O(\log n)$ diameter that the wreath graph possesses will allow the leaders of committees $C(u)$ to communicate with neighboring committees $C(v)$ in $O(\log n)$ time. Additionally, the merging phase of each pair of committees will require only $O(\log n)$ time. The algorithm is almost identical to the `GraphToStar` as far as the high level strategy is concerned. Committees select neighboring committees and merge with them. The main difference is that when a tree with root v is formed, we cannot use the pulling mode since this would increase the degree significantly. We provide an example of two committees merging in Fig. 2. In this example, committee $C(u)$ is merging with committee $C(v)$ and the merging happens through nodes x and y, see Fig. 2(a). The committees on each tree merge in a single ring that includes all committees in $O(1)$ time (ring merging mode), see Fig. 2(b),2(c). After this, v deactivates one of its incident edges in order to create a line subgraph, see Fig. 2(d). Once this happens, each node on the line executes an asynchronous version of the LineToCompleteBinaryTree subroutine in $O(\log n)$ time using the orientation of the new ring, where root v is the root of the line. Once the subroutine is finished, the complete binary tree subgraph of the wreath graph is ready. Therefore we have managed to merge a tree graph of multiple committees into a single committee. Fig. 2 does not include the asynchronous version of the LineToCompleteBinaryTree subroutine since it is quite involved to illustrate.

Algorithm `GraphToWreath`

The structure of each committee/node is the same as the `GraphToStar` algorithm apart from the fact that each committee $C(u)$ is a wreath graph. Every node is able to distinguish between the edges of the binary tree and the edges of the ring by marking them and it can also distinguish its clockwise neighbor and counterclockwise neighbor on the ring. Our algorithm proceeds in phases, where in every phase each committee $C(u)$ executes in one of the following modes, always executing in selection mode in phase 1.

- **Selection:** If $C(u)$ has a neighboring committee $C(z)$ such that $UID_z > UID_u$ and $C(z)$ is not in *Ring Merging mode* or *Tree Merging mode* then, from its neighboring committees not in ring merging or tree merging mode, $C(u)$ *selects* the one with the greatest UID; call the latter $C(v)$. If $C(u)$ selected $C(v)$ or $C(u)$ was selected, $C(u)$ enters the Ring Merging mode. If $C(u)$ did not select anyone and it was not selected by anyone, it stays in the selection mode. If $C(u)$ has no neighboring committees, $C(u)$ enters the termination mode.
- **Ring Merging:** Given that in the previous phase, $C(u)$ selected $C(v)$, committee $C(u)$ merges its ring component with the ring component of $C(v)$ by the following method: Let $k \in C(u)$ and $l \in C(v)$, such that edge kl is active. k activates an edge with the clockwise neighbor of l, call it l_1, and l activates an edge with the clockwise neighbor of k, call it k_1. Then they deactivate edges kk_1, ll_1, and kl. The two rings have now merged into a single ring.
  Given that in the previous phase, $C(u)$ was selected by $C(k)$, committee $C(k)$ merges its ring component with the ring component of $C(u)$. $C(u)$ enters the tree merging mode.
- **Tree Merging:** Every node x in $C(u)$ executes one round of an asynchronous version of the LineToCompleteBinaryTree algorithm, which extends the LineToCompleteBinaryTree algorithm with extra wait states. If there exists node x that has not terminated the asynchronous LineToCompleteBinaryTree algorithm, $C(u)$ stays in the Tree Merging mode. If all nodes x have terminated the asynchronous LineToCompleteBinaryTree algorithm, all nodes x have now merged with committee $C'(u)$ whose leader is the root of the complete binary tree and $C'(u)$ enters the selection mode. $C(u)$ does not exist anymore.
- **Termination:** Each follower x in $C(u)$ deactivates every edge apart from the edges that define the spanning complete binary tree of $C(u)$.

5.1 Low Level Description of Modes

In this subsection, we are going to describe the low level details of each mode since the communication process is much more complicated than the `GraphToStar` algorithm.

Selection. Consider committee $C(u)$. Each follower x in committee $C(u)$ sends a message $\{myUID_x, maxNeighborUID, maxNeighborDiameter\}$ to its

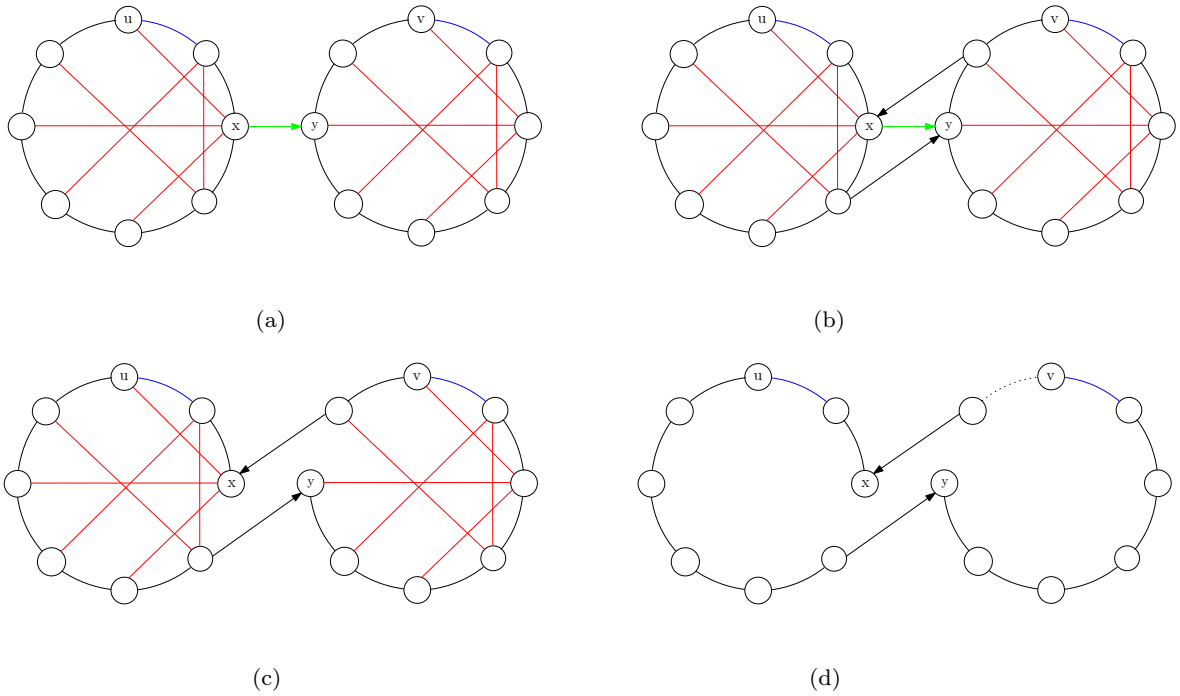

Fig. 2 Example where committee $C(u)$ is merging with committee $C(v)$ and the merging happens through nodes x and y. Figure (a) shows the initial connections. Figure (b) shows that ring merging process where x and y activate edges with the counterclockwise neighbors of each other. Figure (c) shows the deactivation of edges from x and y in order to form the cycle which includes the black and blue edges. Figure (d) shows that node v deactivates its incident edge (dotted line) in order to turn the cycle into a line where the asynchronous version of the LineToCompleteBinaryTree subroutine will be executed.

leader u via the binary tree subgraph, see Fig. 3. Variable $myUID_x$ contains the UID of node x, $maxNeighborUID$ contains the UID of the neighboring committee with the greatest UID among all neighboring committees that x has an edge with, and $maxNeighborDiameter$ contains the diameter of that committee.

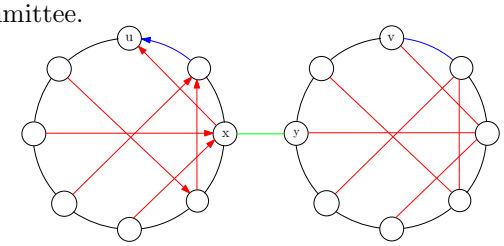

Fig. 3 Every follower in $C(u)$ sends a message with the information of its neighboring committees to leader u via the complete binary tree. For example, follower x sends the information for committee $C(v)$

.

After committee leader u receives all triplets, u knows the UID of all neighboring committees. If $\exists\ maxNeighborUID > UID_u$, $C(u)$ selects the neighboring committee $C(v)$ with the greatest $maxNeighborUID$ and broadcasts a message to x to initiate the connection with that committee. Since, it is possible that multiple followers x sent the same $maxNeighborUID$, u picks the one with the greatest UID_x. If $\nexists\ maxNeighborUID > UID_u$, committee $C(u)$ does not select another committee. Either way, after the selection, u waits to see whether another committee has selected $C(u)$. Committee leader u knows the maximum waiting time since it just received the maximum diameter of all neighboring committees.

After follower x receives the initiation message, it sends a connection message to the leader v of the neighbouring committee $C(v)$ via followers x and y though the binary tree subgraphs. See Fig. 4. After leader v receives all possible requests, it sends back an approval message to all nodes y with a timestamp that defines in which round the merging should happen. See Fig. 5.

Therefore every committee $C(u)$ can understand which committee $C(v)$ it has selected and whether any committees $C'(v)$ have selected $C(u)$. This means that $C(u)$ knows which mode it should enter after the selection phase.

Ring Merging. Assume that multiple committees $C(v_1), C(v_2), ..., C(v_i)$ for $i = 1, ..., n-1$ have selected committee $C(u)$ in the selection phase, via followers $y_1, y_2, ..., y_i$ respectively, whose neighbour $x \in C(u)$ will

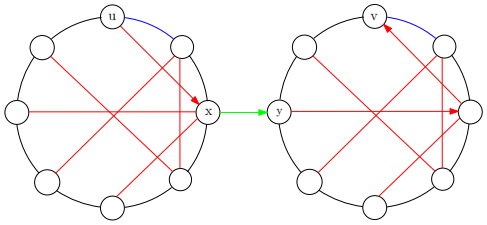

Fig. 4 Leader u sends a message to follower x to initiate a connection with committee v. Follower x sends the request to follower y who propagates it to leader v

.

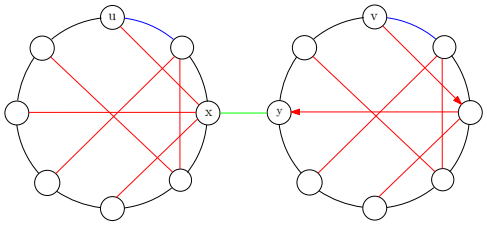

Fig. 5 Leader v sends the approval message to follower y to initiate the merging with committee u

.

initiate the connection. See Fig. 6(a) for an example. Followers $y_1, y_2, ..., y_i, x$ execute the following steps in order to complete the ring merging mode.

- Follower x sends a message to followers $y_1, y_2, ..., y_i$ to rearrange themselves into an *inner-circle* by activating edges $\{x, y_1\}$, $\{y_1, y_2\}, \{y_2, y_3\}, ..., \{y_{i-1}, y_i\}, \{y_i, x\}$ and deactivating edges $\{y_1, x\}, \{y_2, x\}, ..., \{y_i, x\}$. See Fig. 6(b).
- Each follower activates an edge with the clockwise neighbor of its inner-circle outgoing neighbor. See Fig. 6(c).
- Each follower deactivates an edge with its clockwise neighbor, as well as the edges of the inner-circle. See Fig. 6(d).

Note that the orientation of the new ring is the same as the orientation of committee $C(u)$ and all nodes in the committee have the same orientation.

Tree Merging. Note that we cannot use the LineToCompleteBinary tree algorithm from Section 3.3 to merge the tree component of the committees since that algorithm assumes that every node starts the execution at the same time. But in our case, we have multiple committees that are merging together with different sizes and therefore the nodes are not synchronized. Thus, we introduce an asynchronous version of the algorithm where nodes can start the execution at different rounds.

Every node x executes the asynchronous LineToCompleteBinaryTree algorithm which works as follows. If node x was a committee leader, then $leader_x = true$ else $leader_x = false$. The acronym EA stands for Edge Activations and DEA stands for Edge Deactivations.

Algorithm 1 Asynchronous LineToCompleteBinaryTree

```
▷state : EA, DEA, awake, leader
▷initial state of node : EA = 0, DEA = 0, Awake = false
if node receives awake signal OR leader = true then
    Awake = true
end if
if awake = true then
    Broadcast awake
    if grandparent has only 1 child then
        if EA_my = DEA_my = EA_father = DEA_father then
            Activate edge with grandparent
            EA_my ++
        end if
        if EA_my = DEA_my + 1 = EA_child then
            Deactivate edge with parent
            DEA_my ++
        end if
    end if
end if
```

We are going to give an intuition on how this algorithm works. First, the leader of each committee broadcasts an awake signal to its own committee. Once a node awakes, it starts executing the asynchronous LineToCompleteBinaryTree algorithm. Since nodes have different waking points, we cannot use the synchronous LineToCompleteBinaryTree that requires synchronized clocks from each node. Therefore, we are going to use other properties that are present for every node in the synchronous LineToCompleteBinaryTree which are: (i) Every node x has the same total number of activations as its parent. (ii) Every node has the same number of total activations as total deactivations. The asynchronous version tries to mimic that by having every node activate an edge, only when its parent has the same total number of edge activations as itself. Similarly, for the deactivations, every node checks that its child has the same deactivations as itself before deactivating an edge. This way, the synchronous LineToCompleteBinaryTree is simulated by the asynchronous version.

Correctness

Lemma 8 *Algorithm* `GraphToWreath` *solves Depth-*$\log n$ *Tree.*

Proof It suffices to prove that in any execution of the algorithm, one committee eventually enters the termination mode and that this committee can only be

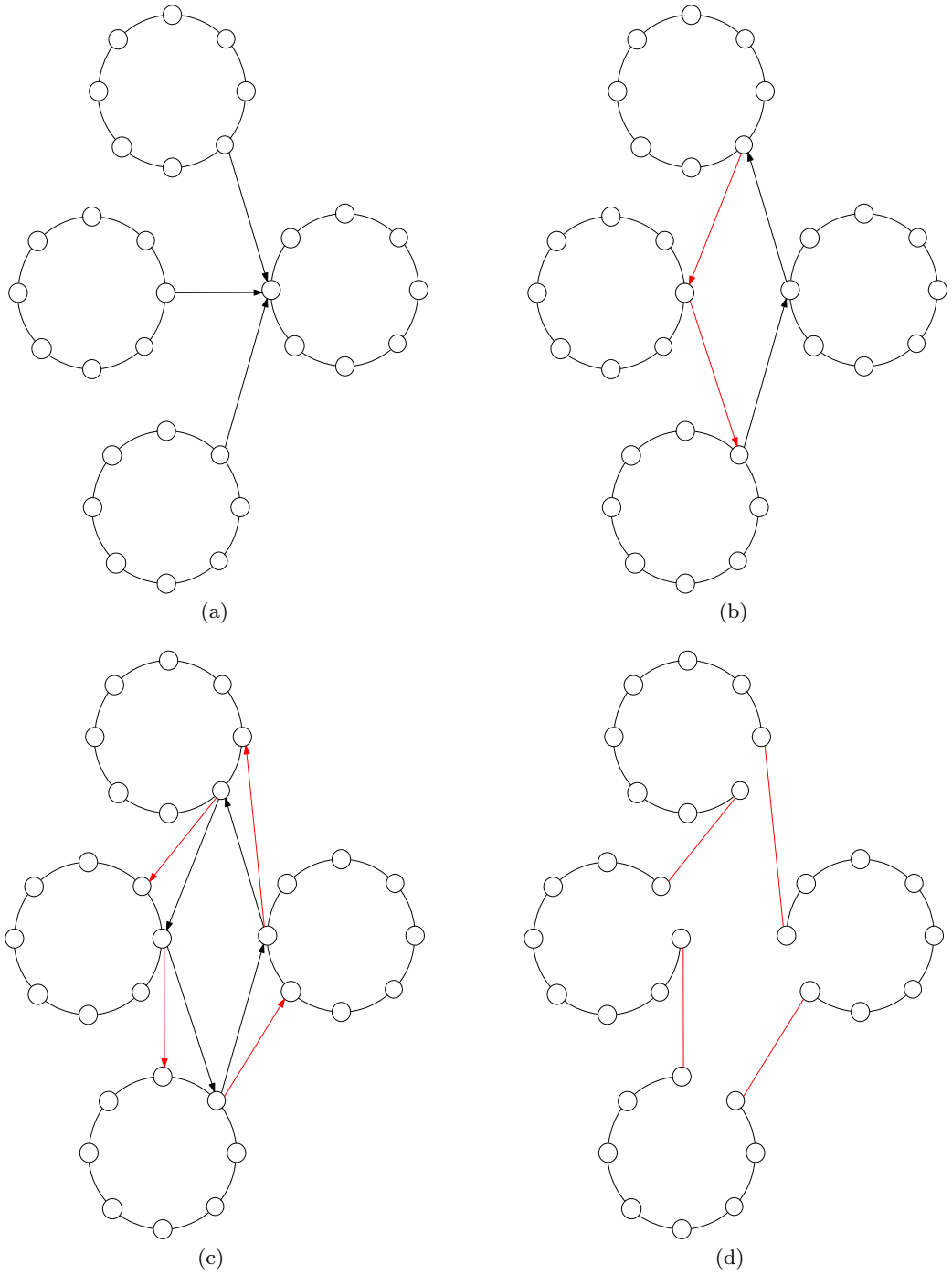

Fig. 6 Example where 3 committees $C(v_1), C(v_2), C(v_3)$ have selected committee $C(u)$. Figure (a) shows the initial connection. In figure (b) committees rearrange themselves into an inner-circle. In figure (c) each committee activates an edge with the clockwise neighbor of its inner-circle outgoing neighbor. In figure (d) each committee deactivates an edge with its clockwise neighbor(based on the committee orientation), as well as the edges of the inner-circle.

$C(u_{max})$ where u_{max} is the highest UID in the network. If this holds, then by the end of the termination phase $C(u_{max})$ forms a complete binary spanning tree rooted at u_{max} and u_{max} is the unique leader of the network. This satisfies all requirements of Depth-$\log n$ Tree.

A committee *dies* only when it merges with another committee by entering the tree merging mode. First observe that there is always at least one *alive* committee. This is $C(u_{max})$, because when it enters the tree merging mode, it is always the root of the complete binary tree. We will prove that any other committee eventually dies or grows, which due to the finiteness of n will imply that eventually $C(u_{max})$ will be the only alive committee.

In any phase, but the last one which is a termination phase, it holds that every alive committee $C(u)$ is in one of the selection, ring merging, and tree merging modes. If $C(u)$ is in the ring merging mode then it will enter the tree merging mode and if its leader is not the root of the complete binary tree, then by the end of the current phase it will have died by merging with another committee $C'(u)$. It, thus, remains to argue about committees in the selection mode.

Now, a committee $C(u)$ in the selection mode can enter the tree merging mode. As argued above, if it enters the ring merging and tree merging modes in sequence it will either die or it will eventually grow. Thus, it suffices to consider the case in which it remains in the selection mode indefinitely. This can only happen if all current and future neighboring committees of $C(u)$ have a UID smaller than UID_u. But each of these must have selected a neighboring $C(w)$, such that $UID_w > UID_u$, otherwise it would have selected $C(u)$. Any such selection, results in $C(w)$ becoming a neighbor of $C(u)$, thus contradicting the indefinite local maximality of UID_u. □

Time Complexity

Let us move on to proving the time complexity of our algorithm. At the beginning, we are going to ignore the number of rounds within a phase, and we are just going to study the maximum number of phases before a single committee is left.

Lemma 9 *After $O(\log n)$ phases, there is only a single committee left in the graph.*

Proof Note that there is a direct correspondence between the modes in the `GraphToWreath` algorithm and the `GraphToStar` algorithm.

Both selection modes are used to decide the selections between the neighboring committees. The difference between the two algorithms is that each selection phase has a different running time. In particular, The `GraphToStar` selection phase required 2 rounds while the selection phase of the `GraphToWreath` requires $O(\log n)$ rounds due to the diameter of the Wreath graph that each committee has. Therefore Lemma 3 that talks about the selection waiting time still holds.

The ring mode is always an intermediate phase between the selection phase and the tree merging phase that lasts for $O(1)$ rounds. The purpose of this mode is to turn the tree T created by the committees in the selection phase into a cycle so that the LineToCompleteBinaryTree subroutine can work. The pulling mode in the `GraphToStar` implements the TreeToStar subroutine, while the tree merging mode in the `GraphToWreath` implements the asynchronous version of the LinetoCompleteBinaryTree. Both subroutines are used to merge the Trees T of depth t created by the committees in $O(\log t)$ time and recall from the basic subroutines subsection that the TreeToStar and the LineToCompleteBinaryTree have the same running time. Therefore both algorithms require the same amount of phases. Therefore Lemmas 2, 4 and 5 that show the growth of each committee still hold.

Note that there is no merging or waiting mode in the `GraphToWreath` since those modes have also been implemented by the merging tree mode.

Since all modes that have been implemented in the `GraphToWreath` have equivalent modes in the `GraphToStar` with similar running times and growths for the committes, the `GraphToWreath` algorithm requires at most $O(\log n)$ phases. □

Lemma 10 *Each phase in the `GraphToWreath` algorithm, requires at most $O(\log n)$ rounds.*

Proof First, we argue that the selection phase requires $O(\log n)$ rounds since each committee $C(u)$ has to exchange information with its neighboring committees in order to decide which committee $C(w)$ it is going to merge with and whether any other committee $C(v)$ will decide to merge with $C(u)$. This requires time that is upper bounded by the diameter of each committee. Based on the low level description of the selection mode, the leader of committee $C(u)$ learns the UID of every neighboring committee in $\log d$ rounds and initiates the connection with the chosen neighboring committee $C(w)$ in another $\log d$ rounds, where d is the diameter of committee $C(u)$. Another $\log d_w$ rounds are required in order for committee $C(w)$ to accept and initiate the connection with committee $C(u)$, where d_w is the diameter of $C(w)$. Since $\log d \leq \log n$ and $\log d_w \leq \log n$, the selection mode requires $O(\log n)$ rounds.

The ring merging phase requires $O(1)$ rounds since every committee has to merge its ring component with

committees $C(v)$ and the running time does not depend on the size of each committee participating.

Each tree merging mode implements one round of the asynchronous LineToCompleteBinaryTree. Note here that the asynchronous version of this algorithm has the same running time as the synchronous version if we consider round 0 to be the first round in which all nodes are awake. Additionally, since every committee leader broadcasts the awake message to its own committee, the time needed for all nodes to be awake is $\log(\max d) < \log n$. Thus, the running time of the asynchronous LineToCompleteBinaryTree is $O(\log n)$. □

Edge Complexity

The analysis for the total edge activations is simple. The algorithm runs for $O(\log^2 n)$ rounds and each node activates at most 1 edge per round. Therefore the total edge activations are $O(n \log^2 n)$.

Let us consider the maximum incident edges that a node can have, excluding the edges of the initial graph. Each node has up to 2 edges for the ring component of the wreath and 2 for the binary tree component of the wreath graph. Based on the low level description of the `GraphToWreath` algorithm, a node can have 1 active edge used for the ring merging phase. Additionally, it can have 2 active edges for the execution of the LineToCompleteBinaryTree. Therefore the number of active edges per round is $O(n)$ and the maximum degree of each node is $7 + c$, where c is the degree of each node in the original graph.

Theorem 2 *For any initial connected graph with constant degree, the* `GraphToWreath` *algorithm solves Depth-*$\log n$ *Tree problem in* $O(\log^2 n)$ *time with* $O(n \log^2 n)$ *total edge activations,* $O(n)$ *active edges per round and* $O(1)$ *maximum activated degree.*

6 Trading the Degree for Time

In this section, we provide another algorithm aiming at $O(\frac{\log n}{\log \log n})$ time for the merging but we are going to allow the maximum degree to reach $O(\log^2 n)$. This requires a new graph for the committees where the diameter of the shape is $O(\frac{\log n}{\log \log n})$, so that the communication within the committees is $O(\frac{\log n}{\log \log n})$ and a new way to merge the committees in $O(\frac{\log n}{\log \log n})$ time. For this algorithm only, we also make the assumption that all nodes know the size of the initial graph. This yields an interesting open problem on whether we can modify the algorithm so that it will not require knowledge of the initial network.

The new graph is very similar to the Wreath graph and we call it *ThinWreath*. The main difference is that instead of having a complete binary tree component, it has a complete polylogarithmic degree tree component with diameter $O(\frac{\log n}{\log \log n})$. The $O(\frac{\log n}{\log \log n})$ diameter that the ThinWreath graph possesses will allow the leaders of neighboring committees to communicate in $O(\frac{\log n}{\log \log n})$ time.

Algorithm `GraphToThinWreath`

The structure of each committee is the same as in `GraphToStar` algorithm, apart from the fact that each committee $C(u)$ is a ThinWreath graph. We also assume that the nodes know the size of the initial graph. Our algorithm proceeds in phases, where in every phase each committee $C(u)$ executes in one of the following modes, always executing in selection mode in phase 1.

- **Selection:** If $C(u)$ has a neighboring committee $C(z)$ such that $UID_z > UID_u$ and $C(z)$ is in selection mode, then, $C(u)$ *selects* its neighboring committee with the greatest UID; call the latter $C(v)$. If $C(u)$ was selected by another committee, $C(u)$ enters the Matchmaker mode. If $C(u)$ was not selected and $C(u)$ selected $C(v)$, $C(u)$ enters the Matched mode. If $C(u)$ did not select anyone and it was not selected by anyone, it stays in the selection mode. If $C(u)$ has no neighboring committees, it enters the termination mode.
- **Matchmaker:** If multiple committees had selected $C(u)$ in the previous phase, committee $C(u)$ matches those committees in pairs. If the number of committees that selected $C(u)$ is odd, one committee is matched with $C(u)$. $C(u)$ enters the Matched mode.
- **Matched:** If committee $C(u)$ selected committee $C(v)$ in the last selection phase, committee $C(u)$ is matched with another committee. Committee $C(u)$ enters the Ring Merging mode.
- **Ring Merging:** Given that in the previous phase, $C(u)$ was matched with $C(v)$, committee $C(u)$ merges its ring component with the ring component of $C(v)$ where the winning committee is $C(u)$ if $UID_u > UID_v$, otherwise $C(v)$ is the winning committee. Either way, committee $C(u)$ enters the Leader Merging mode.
- **Leader Merging:** Given that in the previous mode, committee $C(u)$ lost to committee $C(w)$, the leader of $C(u)$ activates an edge with the leader of $C(w)$. If committee $C(w)$ has lost to some other committee $C(z)$ in the previous phase, $C(u)$ enters the Tree Merging mode. If $C(u)$ did not lose to any other committee, $C(u)$ enters the Tree Merging mode where u is the root.
- **Tree Merging:** The leader of $C(u)$ executes one round of the asynchronous LineToCompletePoly-

logarithmicTree algorithm, which is similar to the asynchronous LineToCompleteBinaryTree algorithm with a termination criterion of $\log n$ children instead of 2. If there exists node x that has not terminated the asynchronous LineToCompletePolylogarithmicTree algorithm, $C(u)$ stays in the Tree Merging mode. If all nodes x have terminated the asynchronous LineToCompletePolylogarithmicTree algorithm, all nodes $x \in C(u)$ have now merged with committee $C'(u)$ whose leader of $C(u)$ is the root of the complete polylogarithmic tree and $C'(u)$ enters the selection mode. Committee $C(u)$ does not exist anymore.

– **Termination:** Each follower $x \in C(u)$ deactivates every edge apart from the edges that define the complete polylogarithmic spanning tree subgraph.

6.1 Low Level Description of the Modes

We will now describe the low level operation of the modes. The selection and ring merging modes are identical to the equivalent modes of the `GraphToWreath` algorithm and therefore will not be described here.

Matchmaker. Before we begin the description of this mode, we would like to give some insight on the Matchmaker/Matched modes and what they are trying to achieve. After the selection mode, the graph of committees consists of directed trees, directed lines and pairs. We want to break up the directed trees into lines and pairs since directly merging the committees using the directed trees might result in having a final committee with linear degree due to the structure of the directed tree. We break up the committees by pairing up the multiple committees $C(v)$ that have selected committee $C(u)$. The difficulty here arises from the fact that committees $C(v)$ are not neighbours and they have to use committee $C(u)$ in order to become neighbours by activating edges on $C(u)$. While doing this, we have to make sure that these edge activations don't violate the maximum degree of $O(\log n)$.

In this mode, we know that at least one committee $C(v)$ has selected committee $C(u)$. Leader u sends a synchronisation message with a timestamp to all leaders v which dictates when the Matched mode algorithm should begin. This timestamp is equal to $3 \cdot d$ where d is the diameter of the neighbouring committee with the highest UID among all neighbouring committees of $C(u)$. This guarantees that the message can reach every leader v in $2 \cdot d$ time and d time for the v leaders to send the message back to their followers. After this, committee $C(u)$ enters the Matched mode.

Matched. In this mode, after leader v receives the synchronization message from leader u, leader v sends the timestamp to follower y to begin the Matched algorithm. Once follower y receives the message, it starts executing the following algorithm on the round specified by the timestamp. Followers $x \in C(u)$ are responsible for Matching followers y.

Algorithm 2 Matched

```
▷state : round, Matched
▷initial state of node : round = request, Matched = {1, myUID},
if round == request then
    Send Matched to follower x
end if
if round == receive then
    Receive Matched' = {Match, UID} from follower x
    if Match == 0 then
        Activate edge with parent of x
        Deactivate edge with x
    end if
    if Match == 1 then
        myMatch = C(UID)
        round = terminate
    end if
end if
if round == terminate then
    Terminate with committee myMatch as its Matched committee
end if
if round == request then
    round = receive
else
    round = request
end if
```

Follower x acts as a matchmaker in this mode. In every round, each follower y_i asks the current neighbour x to be matched with another follower y_j. If multiple followers y_i send a $Matched$ message, follower x matched them in pairs, using their UIDs in ascending order and sends $Matched = \{1, UID\}$ back to each follower y_i where UID is the committee that each follower y_i is matched with. See Figs. 7(c),7(d). If only one follower y_i sends a $Matched$ message then follower x sends back $Matched = 0, UID$ to inform it that no matches are present. See Figs. 7(a). After that follower y_i moves on to the next level of the polylogarithmic tree by activating an edge with the parent of follower x and looks again for a match. See Fig. 7(b).

In short, followers y_i might start at different levels of the polylogarithmic tree of $C(u)$. In each round, they activate an edge with the next level until they find a match at their current level. Note that all followers y_i will find a match, since they have a common destination which is the root of committee $C(u)$.

Let us now consider the maximum activated degree of each follower x during the Matched algorithm. In each round, followers y_i might activate an edge with follower x while coming through the lower levels. Each follower x has at most $\log n$ children and it is not possible for more than 1 follower y_i to come through each child since, if one child had multiple requests from followers y_i, they would get matched together and terminate as in Fig. 7(d). Therefore the maximum activated degree of each follower x can increase by at most $\log n$. Finally note that at this point, the graph of committees consists of directed lines and pairs.

Leader Merging. In this mode, provided that committee $C(v)$ has smaller UID than $C(u)$, leader v activates an edge with leader u by activating edges on the polylogarithmic trees of $C(v)$ and $C(u)$. This process is bound by the diameter of the committees. If we focus on any directed lines in the graph of committees, we can see that we have created a path that consists only of the leaders of the committees in the directed line.

Tree Merging. Every committee leader v executes the asynchronous LineToCompletePolylogarithmicTree algorithm which is the same as the asynchronous LineToCompleteBinaryTree algorithm except that termination criteria requires that the grandfather of each node has $\log n$ children instead of 2 children.

Note here that the whole merging process is finished for this phase. Our final graph consists of a ring graph created by the ring merging process and a collection of wreath graphs with leader u as the root, created by the Leader/Tree merging process. Because of the Tree merging process, there is a polylogarithmic tree consisting of leaders u and v with diameter $O(\frac{\log n}{\log\log n})$. Additionally each leader v is the root of its own polylogarythmic tree with diameter $O(\frac{\log n}{\log\log n})$ from the previous phase. Therefore the diameter of the collection of wreath graphs is $O(\frac{\log n}{\log\log n})$.

6.2 `GraphToThinWreath` Proof

For this algorithm's proof, it is not possible to use the same strategy as the previous algorithms. This is because, while we can prove that this algorithm also requires $O(\log n)$ phases as the previous algorithms, all of our modes require $O(\frac{\log n}{\log\log n})$ but for the tree merging mode which requires $O(\log n)$ and therefore similar analysis would yield $O(\log^2 n)$ running time. Our new strategy is to show that after $O(\log n)$ rounds in which at least one committee is in the tree merging mode in each round, there is only a single committee left in the graph.

Correctness

Lemma 11 *Algorithm* ***GraphToThinWreath*** *solves the Depth-*$\frac{\log n}{\log\log n}$ *Tree problem.*

Proof Since the selection mode of the `GraphToThinWreath` algorithm is identical with the `GraphToWreath` algorithm, we argue that there will be a single committee left in the final graph. This committee consists of a ring subgraph and multiple thinwreath subgraphs. Based on the low level description of the tree merging mode, the diameter of the graph is $O(\frac{\log n}{\log\log n})$. □

Time Complexity

Lemma 12 *After* $O(\log n)$ *tree merging rounds, there is only a single committee left in the graph.*

Proof We define a tree merging round to be a round in which at least one committee is in the tree merging mode. For the purposes of this proof, we are going to consider each tree merging round to be its own phase. Consider the rounds in which a committee is in the tree merging mode. Observe that in any such round, the leaders of those committees form a forest F, where each committee belongs to a tree of F. Any such tree executes the asynchronous LineToCompletePolylogarithmicTree algorithm. This structure is identical to the structure in the pulling mode of the `GraphToStar` algorithm. The only difference between the pulling mode and the tree merging mode is that they are running different algorithms. But, the asynchronous LineToCompletePolylogarithmicTree and the TreeToStar algorithm have the same running time and both of them merge the trees of committees into single committees. Therefore the two algorithms will have the same number of rounds. Based on Lemma 6, there at most $O(\log n)$ phases for the `GraphToStar` algorithm to terminate and every phase includes at most one round of pulling mode and subsequently there are at most $O(\log n)$ rounds of pulling mode. Therefore the `GraphToThinWreath` algorithm can have at most $O(\log n)$ tree merging rounds before a single committee is left in the graph. □

Lemma 13 *The* ***GraphToThinWreath*** *algorithm has* $O(\frac{\log^2 n}{\log\log n})$ *running time.*

Proof In order for a committee to enter the tree merging mode, it has to go through some or all of the other modes of the algorithm which have $O(\frac{\log n}{\log\log n})$ running time since all of them are bound by the diameter of the committee. Therefore, for every tree merging round, there can be at most $O(\frac{\log n}{\log\log n})$ rounds from the other modes. Then based on Lemma 12, the running time of the algorithm is $O(\frac{\log n}{\log\log n}) \cdot O(\log n) = O(\frac{\log^2 n}{\log\log n})$. □

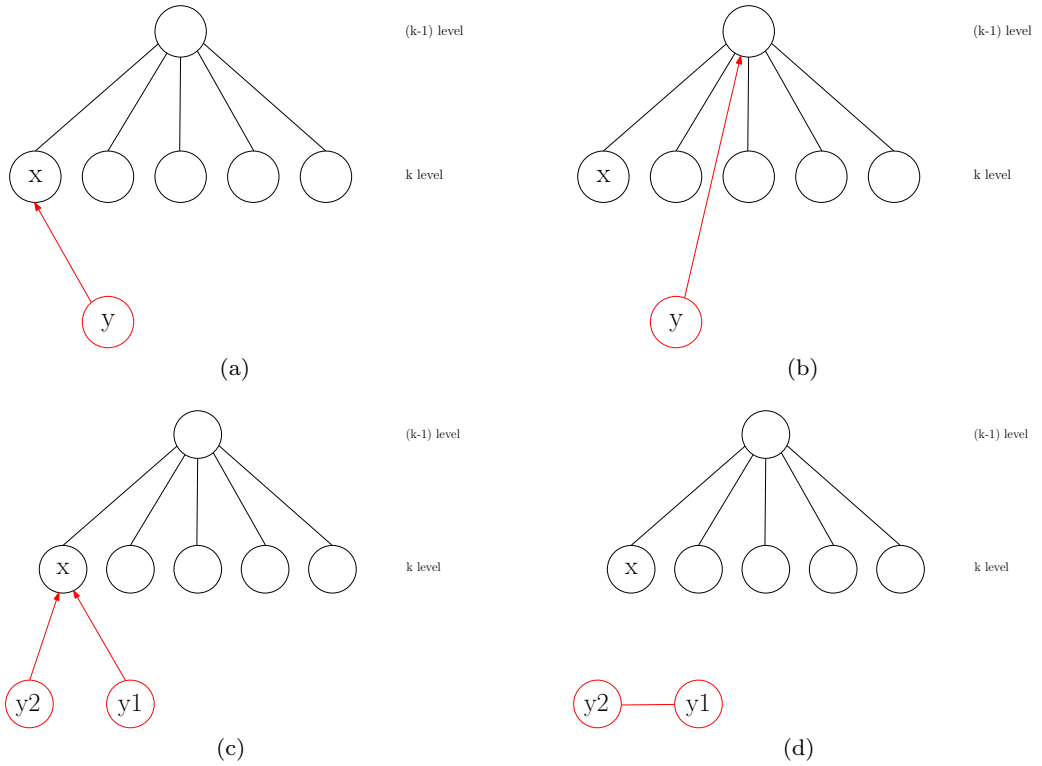

Fig. 7 Figures showing the Matched mode. Figure (a,b) show that committee $C(v)$ goes up one level on the binary tree of committee $C(u)$ if follower y finds no match through follower x. On the other hand, figures (c,d) show that if two committees are in the same round on the same follower x, they get matched together.

Edge complexity

Let us consider the maximum possible edges added on each node throughout each phase. Based on the low level description of the modes, each mode adds at most $O(\log n)$ number of edges to each node. From Lemma 12, we implicitly know that there can be at most $\log n$ phases until the algorithm terminates. Therefore, the maximum degree of each node is $O(\log^2 n)$. Similarly, since in every round, each node activates at most 1 edge, the maximum edges activated are $O(n \cdot \frac{\log^2 n}{\log \log n})$. Finally, since in every mode, every edge activation is followed by a deactivation, the maximum number of activated edges in $O(n)$.

Theorem 3 *For any initial connected graph with polylogarithmic degree, the* `GraphToThinWreath` *algorithm solves Depth-*$\frac{\log n}{\log \log n}$ *Tree in* $O(\frac{\log^2 n}{\log \log n})$ *time with* $O(n \log^2 n)$ *total edge activations,* $O(n)$ *active edges per round and* $O(1)$ *maximum activated degree.*

7 Lower Bounds for the Depth-log n Tree Problem

We will now shift our focus into proving lower bounds for our model. We are going to provide lower bounds for both a centralized model and a distributed one because we want to show that there is an important difference between the two of them.

7.1 Centralized Lower Bounds

In the centralized setting, everything we have previously defined in the model subsection stays the same but now every node also has complete knowledge of the graph and a centralized controller can decide what each node will do in each round.

We begin by defining the *potential* of a UID to a node v. The potential describes how far the UID is from node v. We are going to use this definition to measure how fast the identifier can be transmitted throughout the graph.

Definition 2 We define the *potential* of a UID_u to v as its minimum "distance" from v. The distance is defined as follows: Consider all nodes w in the network that know UID_u. Compute the length of the shortest path between each node w and node v. The minimum length among all shortest paths is the distance between UID_u and node v. We denote the potential of UID_u to v by $\phi_{u,v}$.

Note that in any initial graph $D = (V, E)$, $\forall u, v \in V, \phi_{u,v} \leq n-1$. Consider any pair of nodes u, v, where $\phi_{u,v} = k$. There are two ways to reduce $\phi_{u,v}$ in each round i:

• **Information Propagation**. Consider all nodes w that currently know UID_u. Compute the shortest path between all pairs of w and v and pick node w that yields the smallest shortest path. Node w can send UID_u to one of its neighbors y that belong to the shortest path between w and v to reduce $\phi_{u,v}$ by 1.

• **Reduce Shortest Paths**. Consider all nodes w that currently know UID_u. Compute the shortest path between all pairs of w and v and pick node w that yields the smallest shortest path with $size = k$. Now consider all pairs of nodes x, y that are potential neighbors and also belong to the shortest path between w and v. Activating xy between one pair of x, y reduces $\phi_{u,v}$ by 1. Activating multiple xy between different pairs in one round can reduce $\phi_{u,v}$ even more but at most by $k/2$.

Observation 1 *In order for an algorithm to solve the Depth-$\log n$ Tree Problem, $\forall u, v \in V, \phi_{u,v} \leq \log n$.*

Lemma 14 *Any transformation strategy based on this model requires $\Omega(\log n)$ time to solve the Depth-$\log n$ tree problem if the initial graph G_s is a spanning line.*

Proof Consider a spanning line where, for simplicity, we call the node that resides at the "left" endpoint of the line u and the node that resides at the "right" endpoint of the line v. According to Observation 1, in order for an algorithm to solve the Depth-$\log n$ tree problem, $\phi_{u,v} \leq \log n$. In the initial graph, $\phi_{u,v} = n-1$. We know that by using edge activations, we can reduce $\phi_{u,v}$ by half in each round, and by using Information Propagation we can reduce $\phi_{u,v}$ by 1 in each round. Therefore in order for $\phi_{u,v} = \log n$, any algorithm would require at least $\Omega(\log n)$ rounds. □

Lemma 15 *Any transformation strategy based on this model that solves the Depth-$\log n$ Tree problem in $O(\log n)$ time requires $\Omega(n)$ edge activations.*

Proof Let us again consider a spanning line as the initial graph. W.l.o.g. let us assume that the size of the network is odd. Let us call u the node that is the "left" end point of the line and v the "right" endpoint of the line.

Let us assume that in some round i, where $i \leq \log n$, that $\phi_{u,v} \leq \log n$. We can produce the following equation based on the two rules that allow us to reduce the potential: $InitialPotential - \#EdgeActivations - \#MessagesSent \leq \log n$. The maximum value of $MessagesSent$ is $\log n$ and $InitialPotential = n-1$ and if we add those in the previous equation we get $\#EdgeActivations \geq n - 1 - 2\log n$ and therefore, in order for $\phi_{u,v} \leq \log n$ at least $n-1-2\log n$ edges have to have been activated. □

Lemma 16 *Any transformation strategy based on this model that solves the Depth-$\log n$ Tree problem in $O(\log n)$ time, requires $\Omega(n/\log n)$ edge activations per round.*

Proof From Lemma 15 we know that in order to have $\phi_{u,v} = 0$, in $\log n$ time, there must be at least $EdgeActivations = n$ edge activations. Now, since we are trying to find the minimum number of edge activations per round possible, we can easily do this by dividing the total number of edge activations with the number of rounds. Therefore $EdgeActivationsPerRound \geq \frac{EdgeActivations}{Rounds} \geq \frac{\Omega(n)}{\log n}$. □

Since we have just proven that $\Omega(n)$ edge activations are required in order to solve the Depth-$\log n$ problem given any initial graph, we are now going to prove that $\Theta(n)$ edges are sufficient in order to solve it. First, we are going to informally prove it for the special case of the spanning line graph and afterwards we are going to prove it for general graphs.

Consider a spanning line with nodes $u_1, u_2, \ldots, u_j$ for $j = 1, 2, \ldots, n$. For simplicity, assume that u_1 is the "left" endpoint of the line, u_2 is the neighbor of u_1 etc, u_3 is a neighbor of u_2 etc. In each round i, we activate edge u_j, u_{j+2^i} $\forall$ $\{u_j | (j \mod (2^i) = 1) \wedge (j + 2^i \leq n)\}$. After $\log n$ rounds, the diameter of the shape is equal to $\log n$. Let us now proceed to analyzing the total edge activations. By definition of the algorithm, in each round i, $\frac{n}{2^i}$ edges are activated. Since the algorithm runs for $\log n$ rounds, we have $\sum_{i=1}^{\log n} \frac{n}{2^i} = n-1$ total edge activations. We call this algorithm CutInHalf.

Theorem 4 *Given any initial graph $D = (V, E)$, the Depth-$\log n$ problem can be solved in $O(\log n)$ time, with $\Theta(n)$ total edge activations.*

Proof Since we are in a centralized setting, we are first going to perform some global computations that are going to output the specific edges that have to be activated in order for the diameter of the shape to drop to $\log n$. We consider any initial graph $D = (V, E)$ and we pick an arbitrary node called u. First, we compute a spanning tree that starts from node u. Afterwards we compute an Eulerian tour starting from u. This way we can create a virtual ring $D' = (V', E')$ that has $|V'| \leq 2|V|$ and $|E'| \leq 2|E|$. Now in this ring, node u deactivates one of its incident edges and the graph is now a line. We can now execute the CutInHalf algorithm to solve the Depth-$\log n$ Tree problem in $O(\log n)$ time, with $\Theta(n)$ total edge activations. □

7.2 Distributed Lower Bound

In this part, we are going to show that there is a difference in the minimum total edge activations required for solving the Depth-$\log n$ problem between the centralized and the distributed model. At this point, we would like to remind the reader that an algorithm is called *comparison based* if it manipulates the UIDs of the network using comparison operations $(<, >, =)$ only. Our main theorem will show that any deterministic distributed comparison based algorithm requires $\Omega(n \log n)$ total edge activations to solve the Depth-$\log n$ Tree problem in $O(\log n)$ time. Consider two nodes, called u and v , that have received increasing order UIDs that are larger than both u and v during the execution of a deterministic comparison based algorithm. Since nodes are only allowed to compare UIDs between them, the results of the comparison of u and v are exactly the same and thus, u and v must have the same behaviour until they find a different result by comparing receiving UIDS. We are going to use this behaviour to show that any algorithm must activate $\Omega(n \log n)$ total edge activations.

Definition 3 Let $U = u_1, u_2, \ldots, u_k$ be a sequence of UIDs of length k. We say that U is an *increasing order sequence* if, for all $i, j, 1 \leq i, j \leq k$, we have $i \leq j$ iff $u_i \leq u_j$.

Definition 4 Let A be a *comparison-based* algorithm executing on an increasing order ring graph. Let i and j be two nodes in the ring graph. We say that i and j are in corresponding states if the UIDs that they both have received from counterclockwise neighbors are a decreasing order sequence and the UIDs they have received are an increasing order sequence and vice versa. Two nodes in corresponding states in round i must have the same behaviour during the execution of an algorithm in round i

Definition 5 We define the increasing order ring R as follows. Suppose we have an increasing order sequence U of UIDs to be assigned on a ring with n nodes. We assign the smallest UID from $U = u_1, u_2, \ldots, u_k$ to an arbitrary node and we continue assigning increasing UIDs clockwise (or counterclockwise). We call this an increasing order ring.

Definition 6 We define a round of an execution/algorithm to be active if at least one message is sent in it or an edge is activated in it.

Definition 7 We define the *k-expo-neighborhood* of node i in ring R of size n, where $0 \leq k \leq n/2$, to consist of the $2 \cdot 2^k + 1$ nodes $i - 2^k, \ldots, i + 2^k$, that is, those that are within distance at most 2^k from node i (including i itself).

Lemma 17 *Consider an increasing order ring of size n. Let d_{min} be the initial distance between node d and the node with the minimum UID called d_0. Let d_{max} be the initial distance between node d and the node with maximum UID called d_{n-1}. Let i and j be two nodes in A, where i_{min}, i_{max} is the minimum distance between i and d_0, d_{max} respectively, and j_{min}, j_{max} is the minimum distance between j and d_0, d_{max} respectively. Let A be a comparison-based algorithm executing in the ring. Then, nodes i and j must be in corresponding states for at least k rounds, where $2^k = \min(\max(i_{min}, i_{max}), \max(j_{min}, j_{max}))$.*

Proof Note here that nodes i and j are in corresponding states as long as $(((\phi_{d_0,i} > 0) \vee (\phi_{d_{n-1},i} > 0)) \wedge ((\phi_{d_0,j} > 0) \vee (\phi_{d_{n-1},j} > 0))$. In simple terms, i and j are in corresponding states as long as both of them do not know both UID_{d_0} and $UID_{d_{n-1}}$ which follows from definition 4. This means that i and j will stop being in corresponding states once one of them learns both d_0 and d_max. By definition, 2^k is the potential between i, j and d_0, d_{max} and we know that the potential of a UID can only be decreased by information propagation and reducing shortest paths, where information propagation reduces the potential by at most 1 per round and reducing shortest paths reduces it by half. Thus, since the initial potential is 2^k, by applying the potential reduction methods, any algorithm would need at least $k - \log k$ rounds so that $(((\phi_{d_0,i} = 0) \vee (\phi_{d_{n-1},i} = 0)) \wedge ((\phi_{d_0,j} = 0) \vee (\phi_{d_{n-1},j} = 0))$. □

Observation 2 *Any transformation strategy based on this model that solves the Depth-$\log n$ Tree problem in $O(\log n)$ time in an increasing order ring, requires at least $\log n$ active rounds.*

Theorem 5 *Any deterministic distributed algorithm that solves the Depth-$\log n$ Tree problem in $O(\log n)$ time, requires $\Omega(n \log n)$ total edge activations.*

Proof Consider an increasing order ring R with n nodes and algorithm A that solves the Depth-$\log n$ problem. Consider the node with the greatest UID in the network, called u_{max}, the node with the smallest UID in the network, called u_1, and the antipodal node of u_{max} called u_c.

First of all, note that in the first round, all nodes except from u_1 and u_{max} are in corresponding states. We can generalize this statement by using Lemma 17 to state that in round i, each node whose i-expo-neighborhood does not include both u_1,u_{max} is in a corresponding state with each such node. Therefore those

nodes behave the same way e.g. if in round i, one of those c nodes activates an edge, then all c nodes activate an edge. For this proof, we define a round of algorithm A to be *live* if the c nodes activate at least one edge in it, we also define a round of algorithm A to be *asleep* if none of the c nodes activate an edge in it.

We already know that we need at least $\log n$ active rounds to connect u_{max} with u_c from Lemma 2. Our goal here is to prove that $\log n$ of those active rounds also have to be live rounds.

For simplicity, we define the set C where node $u \in C$ if u is in the same corresponding state as u_c (including u_c), the set A where node $u \in A$ if u is not in the same corresponding state as u_c.

Consider an arbitrary round i, where the shortest path between u_{max} and u_c is $|P| = k$. This shortest path can be split into two different paths. The one called P_A that includes nodes $u \in A$ and the one called P_C that includes nodes $v \in C$. Essentially, the potential $\phi_{u_{max},a} \geq |P_C|$ since otherwise, some node $v \in C$ would know $UID_{u_{max}}$ which is impossible by definition of set C. Let us divide our analysis between asleep and live rounds and study how much the potential can be reduced in each round.

• **Asleep rounds.** In each asleep round a, only nodes $u \in A$ can activate edges and $|P_C|$ can only be reduced by at most $l+1$ where l is the total number of live rounds before round a. We can reduce it l by having $u \in A$ activating an edge with each potential neighbor $v \in C$, and reduce it 1 by having u send $UID_{u_{max}}$ to all $v \in C$.

• **Live rounds.** In each live round l, all nodes can activate an edge so we can reduce $|P_C|$ by $l+1$ by following the above strategy and additionally, use edge activations between nodes $v \in C$ so that $|P_C|$ is reduced by at most half.

Note here, that Asleep rounds are not enough to reduce the potential to 0 in order to solve the Depth-$\log n$ problem. After $O\log(n)$ asleep rounds, $\phi_{u_{max},a} \geq InitialPotential - (\log n)(l+1) = \frac{n}{2} - (\log n)(l+1)$. Therefore we need at least $\log n$ live rounds to solve the Depth-$\log n$ problem.

We are now examining how many edges are activated in each live round. Before we do that, we list some abbreviations: CN: the number of nodes in the original graph, NRL: number of nodes that were removed in previous live rounds, NRA: number of nodes removed in previous asleep rounds. Recall that in each live round l , at least 1 node $v \in C$ activates an edge and by Lemma 17, all nodes $v \in C$ activate an edge. The number of nodes $v \in C$ in round i are $|u| \geq \#CN - NRL - NRA = (n-2) - (\sum_{i=1}^{l-1} 2^i)(\sum_{i=1}^{a} -i(l-1)) - a(l-1)$. The number of edges activated in each round l are at least $|C| \geq |u|$. Therefore the total number of edge activations in live rounds after $\log n$ rounds is at least $(n-2) - (\sum_{i=1}^{\log n} 2^i)(\sum_{i=1}^{\log n} -i(l-1)) - a(l-1) = \Theta(\log n)$. □

8 Conclusion and Open Problems

In this work we considered a distributed model for actively dynamic networks. The model can achieve global distributed computation and network reconfiguration in (poly)logarithmic time, but trivial solutions incur an impractical cost, which is related to the creation and maintenance of edges in the dynamic network generated by the algorithm. We defined natural cost measures associated with the edge complexity of actively dynamic algorithms. It turns out that there is a natural trade-off between the time and edge complexity of algorithms. By focusing on the apparently representative task of transforming any initial network from a given family into a target network of (poly)logarithmic diameter, which can then be exploited for global computation or further reconfiguration, we obtained non-trivial insight into this trade-off.

Our model is inspired by recent developments in the algorithmic theory of dynamic networks and in the theory of reconfigurable robotics. Still, it turns out to be very close to the interesting area of overlay network construction. It is not clear yet what is the formal relationship between the polylogarithmic restriction on communication in overlay networks and our efforts to minimize the total number of edge activations in our algorithms. This remains an interesting question for future research.

There is also a number of technical questions specific to our model and the obtained results. We do not know yet what are the ultimate lower bounds on time for different restrictions on the maximum degree. For maximum degree bounded by a constant our best upper bound is $O(\log^2 n)$ and if bounded by (poly)$\log(n)$ this drops slightly by an $O(\log\log n)$ factor. Can any of these be improved to $O(\log n)$, that is, matching the $\Omega(\log n)$ lower bound on time? It would also be valuable to investigate randomized algorithms for the same problems, like those already developed in overlay networks.

Finally, there are many variants of the proposed model and complexity measures that would make sense and might give rise into further interesting questions and developments. Such variants include anonymous distributed entities which are possibly restricted to treat their neighbors identically even w.r.t. actions (e.g., through local broadcast) and alternative poten-

tial neighborhoods, e.g., activating edges at larger distances.

References

1. Akitaya, H. A., Arkin, E. M., Damian, M., Demaine, E. D., Dujmović, V., Flatland, R., Korman, M., Palop, B., Parada, I., van Renssen, A., and Sacristán, V. Universal reconfiguration of facet-connected modular robots by pivots: The $O(1)$ musketeers. *Algorithmica 83*, 5 (2021), 1316–1351.
2. Almethen, A., Michail, O., and Potapov, I. Pushing lines helps: Efficient universal centralised transformations for programmable matter. *Theoretical Computer Science 830-831* (2020), 43 – 59.
3. Angluin, D., Aspnes, J., Chen, J., Wu, Y., and Yin, Y. Fast construction of overlay networks. In *17th ACM symposium on Parallelism in Algorithms and Architectures (SPAA)* (2005), pp. 145–154.
4. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M. J., and Peralta, R. Computation in networks of passively mobile finite-state sensors. *Distrib. Comput. 18*, 4 (2006), 235–253.
5. Angluin, D., Aspnes, J., Eisenstat, D., and Ruppert, E. The computational power of population protocols. *Distrib. Comput. 20*, 4 (2007), 279–304.
6. Aspnes, J., and Ruppert, E. An introduction to population protocols. In *Middleware for Network Eccentric and Mobile Applications*. Springer, 2009, pp. 97–120.
7. Aspnes, J., and Shah, G. Skip graphs. *ACM Transactions on Algorithms (TALG) 3*, 4 (2007), 37.
8. Aspnes, J., and Wu, Y. $o(\log n)$-time overlay network construction from graphs with out-degree 1. In *11th International Conference On Principles Of Distributed Systems (OPODIS)* (2007), pp. 286–300.
9. Berman, K. A. Vulnerability of scheduled networks and a generalization of Menger's theorem. *Networks 28*, 3 (1996), 125–134.
10. Berman, S., Fekete, S. P., Patitz, M. J., and Scheideler, C. Algorithmic Foundations of Programmable Matter (Dagstuhl Seminar 18331). *Dagstuhl Reports 8*, 8 (2019), 48–66.
11. Bourgeois, J., and Goldstein, S. C. Distributed intelligent mems: Progresses and perspectives. In *International Conference on ICT Innovations* (2011), pp. 15–25.
12. Casteigts, A., Flocchini, P., Quattrociocchi, W., and Santoro, N. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems 27*, 5 (2012), 387–408.
13. Derakhshandeh, Z., Dolev, S., Gmyr, R., Richa, A. W., Scheideler, C., and Strothmann, T. Amoebot - a new model for programmable matter. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures* (New York, NY, USA, 2014), SPAA '14, Association for Computing Machinery, p. 220–222.
14. Enright, J., Meeks, K., Mertzios, G. B., and Zamaraev, V. Deleting edges to restrict the size of an epidemic in temporal networks. *Journal of Computer and System Sciences 119* (2021), 60–77.
15. Gmyr, R., Hinnenthal, K., Scheideler, C., and Sohler, C. Distributed monitoring of network properties: The power of hybrid networks. In *44th International Colloquium on Automata, Languages, and Programming (ICALP)* (2017), pp. 137:1–137:15.
16. Götte, T., Hinnenthal, K., and Scheideler, C. Faster construction of overlay networks. In *26th International Colloquium on Structural Information and Communication Complexity (SIROCCO)* (2019), pp. 262–276.
17. Kempe, D., Kleinberg, J., and Kumar, A. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences 64*, 4 (2002), 820–842.
18. Kuhn, F., Lynch, N., and Oshman, R. Distributed computation in dynamic networks. In *42nd ACM Symposium on Theory of Computing (STOC)* (2010), pp. 513–522.
19. McEvoy, M. A., and Correll, N. Materials that couple sensing, actuation, computation, and communication. *Science 347*, 6228 (2015), 1261689.
20. Mertzios, G., Michail, O., and Spirakis, P. Temporal network optimization subject to connectivity constraints. *Algorithmica 81* (04 2019).
21. Mertzios, G. B., Michail, O., and Spirakis, P. G. Temporal network optimization subject to connectivity constraints. *Algorithmica 81*, 4 (2019), 1416–1449.
22. Michail, O., Chatzigiannakis, I., and Spirakis, P. G. Mediated population protocols. *Theoretical Computer Science 412*, 22 (2011), 2434–2450.
23. Michail, O., Chatzigiannakis, I., and Spirakis, P. G. Naming and counting in anonymous unknown dynamic networks. In *15th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)* (2013), pp. 281–295.
24. Michail, O., Skretas, G., and Spirakis, P. G. On the transformation capability of feasible mechanisms for programmable matter. *Journal of Computer and System Sciences 102* (2018), 18–39.
25. Michail, O., Skretas, G., and Spirakis, P. G. Distributed computation and reconfiguration in actively dynamic networks. In *Proceedings of the 39th Symposium on Principles of Distributed Computing* (New York, NY, USA, 2020), PODC '20, Association for Computing Machinery, p. 448–457.
26. Michail, O., and Spirakis, P. G. Simple and efficient local codes for distributed stable network construction. *Distrib. Comput. 29*, 3 (2016), 207–237.
27. Michail, O., and Spirakis, P. G. Connectivity preserving network transformers. *Theoretical Computer Science 671* (2017), 36–55.
28. O'Dell, R., and Wattenhofer, R. Information dissemination in highly dynamic graphs. In *Joint Workshop on Foundations of Mobile Computing (DIALM-POMC)* (2005), pp. 104–110.
29. Piranda, B., and Bourgeois, J. Designing a quasi-spherical module for a huge modular robot to create programmable matter. *Autonomous Robots 42* (2018), 1619–1633.
30. Scheideler, C., and Setzer, A. On the complexity of local graph transformations. In *46th International Colloquium on Automata, Languages, and Programming (ICALP)* (2019), pp. 150:1–150:14.
31. Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review 31*, 4 (2001), 149–160.