Distributed Computation and Reconfiguration in Actively Dynamic Networks*

Othon Michail
Department of Computer Science, University of Liverpool, UK
Othon.Michail@liverpool.ac.uk

George Skretas
Department of Computer Science, University of Liverpool, UK
G.Skretas@liverpool.ac.uk

Paul G. Spirakis
Department of Computer Science, University of Liverpool, UK
Computer Engineering and Informatics Department, University of Patras, Greece
P.Spirakis@liverpool.ac.uk

ABSTRACT

In this paper, we study 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 in order to carry out a given task. At the same time, 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 family of networks having some good properties, like small diameter.

To formally capture costs associated with creating and maintaining connections, we define three reasonable 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 general task of transforming any G_s into a G_f of diameter (poly)$\log(n)$, while minimizing the edge-complexity.

There is a natural trade-off between time and 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. On the positive side, 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 a running time of $o(\log^2 n)$.

This novel model of distributed computation and reconfiguration in actively dynamic networks and the proposed measures of the edge complexity of distributed algorithms, may open new avenues for research in the algorithmic theory of dynamic networks.

KEYWORDS

distributed algorithms, dynamic networks, reconfiguration, transformation, polylogarithmic time, edge complexity

ACM Reference Format:
Othon Michail, George Skretas, and Paul G. Spirakis. 2020. Distributed Computation and Reconfiguration in Actively Dynamic Networks. In *ACM Symposium on Principles of Distributed Computing (PODC '20), August 3–7, 2020, Virtual Event, Italy*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3382734.3405744

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 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. [14] 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 the *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

*All authors were supported by the NeST initiative. The last author was also supported by the EPSRC project EP/P02002X/1 (Algorithmic Aspects of Temporal Graphs).
Full version: https://arxiv.org/pdf/2003.03355.pdf

PODC '20, August 3–7, 2020, Virtual Event, Italy

ACM ISBN 978-1-4503-7582-5/20/08…$15.00
https://doi.org/10.1145/3382734.3405744

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 it 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 spanning clique K_n is formed in $O(\log n)$ rounds. Such a clique can then be used for global computations, like electing the maximum id 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, 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 insufficient 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.

To appreciate the difficulty in solving the above problem while optimizing the edge complexity, assume for a moment, a network as simple as a spanning line $u_0u_1 \cdots u_{n-1}$ with a pre-elected unique leader on one of its endpoints, say u_0. If we had global view of the system, then we would proceed in $\log n$ phases as follows. In every phase i, we would start from u_0 and activate edges by making hops of length 2 over the edges activated in the previous phase, thus, activating the edges $u_0u_{2^i}, u_{2^i}u_{2\cdot 2^i}, u_{2\cdot 2^i}u_{3\cdot 2^i}, \ldots$ in the current phase. This would give an edge for every 2^i consecutive nodes in phase i and a total of $O(n)$ edge activations. The diameter of the resulting network and the number of phases are both logarithmic

in n. Observe now that this basic construction essentially requires to determine which half of the nodes that activated an edge in the previous phase will be the ones to activate in the current phase. But all these nodes are bound to behave identically given an order-equivalence of received ids in their local history and there is no obvious way to exploit the pre-elected leader at u_0 for symmetry breaking, as its initial distance from many of them is asymptotically equal to the original diameter of the network, that is, $\Theta(n)$. What this example reveals, is an inherent trade-off between time and edge activations stemming from the inability of the distributed entities to break symmetry locally and, thus, fast. Intuitively, breaking symmetry takes time and, if left unbroken, costs many edge activations every time one of the nodes decides to activate.

The difficulties that we just highlighted are formally captured by our lower bounds presented in Section 6. 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 distributed 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 begin our algorithmic constructions with some basic algorithms for special types of initial and target networks, which will then be used as core components in our general algorithms. These are discussed in Section 2.3. One of these algorithms transforms any rooted tree into a star and the other transforms an oriented spanning line into a complete binary tree. Both operate in $O(\log n)$ time, have a linear number of active edges per round and an optimal total of $O(n \log n)$ edge activations. The latter algorithm additionally maintains a maximum degree of at most 3 throughout its course, while the degree of the former is linear.

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 id dominates. If properly performed, this ensures that eventually only one committee remains, namely, the committee of the node u_{max} with maximum id 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 winning 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 winning times of different committees may be different, but we can still show that their amortized growth is exponential.

Our first algorithm, called GraphToStar and presented in Section 3, 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 4, 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 improvement 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 spanning complete binary 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 5, 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 1.4 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.

1.4 Related Work

Temporal Graphs. The algorithmic study of temporal graphs was initiated by Berman [9] and Kempe *et al.* [16], 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 [19]. The design task was carried out by an offline centralized algorithm starting from an empty edge set. Subsequent work [12], 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 [20, 23]. 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 [24], 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 [17, 21, 25] 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.

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 id 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.* [27] 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 id. 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.

To the best of our knowledge, the only previous deterministic algorithm for the problem is the one by Gmyr *et al.* [14]. 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.

A very recent work by Götte *et al.* [15] 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.

Scheideler and Setzer [26] 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, 11, 13, 22]. A prominent such property is changing their shape. Typical examples of systems in this area are reconfigurable robotics, swarm robotics, and self-assembly systems [10, 18]. 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.

2 PRELIMINARIES

2.1 The 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 who 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 id 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 operating 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.

2.2 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 $deg(G)$ denote the degree of a graph G. Then, formally, the maximum activated degree is equal to $\max_{i \in [T]} deg(D(i) \setminus D(1))$, 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.

2.3 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. 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 2.1. *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, with $O(n)$ 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 2.2. *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.

2.4 General Strategy for Depth-d Tree

All algorithms developed in this paper solve the Depth-d Tree problem starting from any 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 id 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-id 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 id 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.

3 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 6.

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.
- **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 $E(G_s)\backslash E(C(u))$. In particular, each follower x in $C(u)$ deactivates all active edges incident to it but xu.

Correctness

Lemma 3.1. *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 $\mathcal{C}_p$. Observe that, in any given phase, the committees in pulling mode form a forest F, where each $C(u) \in \mathcal{C}_p$ belongs to a tree T of F. Any such tree executes the TreeToStar algorithm (from Section 2.3) on committees 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 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 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 an id 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 $S(C(u_s))$ to be the *size* of committee $C(u)$ in phase s.

Lemma 3.2. *Consider committee $C(v)$ 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 $C(v)$ once it enters the selection mode in phase $s+j+1$ is at least 2^{k+j}.*

Proof. Any committee $C(u)$ in waiting mode is a root of (i) either a 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 $C(u)$'s children are in merging mode. In every phase, $C(u)$'s children merge with $C(u)$ and their children become the new children of $C(u)$ and enter the merging mode. It follows that all non-root committees in the tree will eventually merge with $C(u)$ in some phase j. Note that due to the nature of the pulling mode, in each phase the children of $C(u)$ are doubled. This is true because the pulling mode is simulating the TreeToStar algorithm on committees. Recall that we assumed that the size of every committee is $S(C(v_s)) \geq 2^k$ in phase s. Then in each phase $s+i$, where $0 < i \leq j$, the size of the root committee is $S(C(u_{s+\log i})) = S(C(u_s)) + 2 \cdot S(C(v_s)) + 4 \cdot S(C(v_s)) + \ldots + 2^{\log(i-1)} \cdot S(C(v_s)) = 2^{k+i}$.

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

Lemma 3.3. *If committee $C(u)$ stays in the selection mode for $p \geq 4$ consecutive phases, then $C(u)$ has a neighboring committee $C(v) \in \mathcal{C}_p$ that belongs to a 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 tree T. If $C(v)$ does not belong to a 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 $C(v)$ and $C(w)$ enters the selection

mode. Therefore $C(v)$ would select $C(w)$ in phase $k+2$ and exit the selection mode. Thus, a contradiction. If $C(v)$ is in waiting mode in phase k, it cannot be the root of a 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 3.4. *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 $p \geq 4$ consecutive phases, then in phase $s+p+1$ it will select or get selected by a committee $C(v)$ of size 2^{k+p-4}.*

Proof. From Lemma 3.3 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 tree T with. Since $C(u)$ exits the selection mode in phase $s+p$, it either selects committee $C(w)$ that the root of tree T or $C(w)$ selects $C(v)$. Since $C(u)$ was in the selection phase for p phases, committee $C(w)$ was on a tree of depth at least $p-3$. From Lemma 3.2 it follows that the size of $C(w)$ is 2^{k+p-3}. □

Lemma 3.5. *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 in phase $p+1$ is at least 2^{k+p-4}.*

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 3.4 it follows that if $p \geq 4$ the winning committee between $C(w)$ and $C(u)$ will have size at least 2^{k+p-3} in phase $s+p+1$. □

Lemma 3.6. *After $O(\log n)$ phases, there is only a single committee left in the graph.*

Lemma 3.7. *Each phase consists of at most 2 rounds.*

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. 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.

Theorem 3.8. *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 and $O(n)$ active edges per round.*

4 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 edges 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 4.1 (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 $D_r = (V, E_r)$ belongs to the class of ring graphs, $D_b = (V, E_b)$ belongs to the class of complete binary tree graphs, and $E = E_r \cup E_b$.

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 w is formed, we cannot use the pulling mode since this would increase the degree significantly. Instead the committees on each tree merge in a single ring that includes all committees in $O(1)$ time (ring merging mode). After this, w deactivates one of its incident edges in order to create a line subgraph. Once this happens, each node on the ring executes an asynchronous version of the LineToCompleteBinaryTree subroutine in $O(\log n)$ time using the orientation of the new ring, where root w 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.

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. 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 subgraph.

Note here that we omit the communication steps for clarity and we claim that any communication performed between neighboring committees can be completed in $O(\log n)$ rounds since the diameter of each committee is at most $O(\log n)$.

Theorem 4.2. *For any initial connected graph with constant degree, the GraphToWreath algorithm solves Depth-*$\log n$ *Tree 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.*

5 TRADING THE DEGREE FOR TIME

For our new algorithm, we are going to try to have $O(\frac{\log n}{\log \log n})$ 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 our 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 $O(\frac{\log n}{\log \log n})$. We also have to make the assumption that all nodes know the size of the initial graph.

Our 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 $O(\frac{\log n}{\log \log n})$ and polylogarithmic degree. The $O(\frac{\log n}{\log \log n})$ diameter that the ThinWreath graph possesses, will allow the leaders of committees $C(u)$ to communicate with neighboring committees $C(v)$ in $O(\frac{\log n}{\log \log n})$ time.

Algorithm GraphToThinWreath

The structure of each committee/node is the same as the GraphToStar algorithm, apart from the fact that each committee $C(u)$ is a ThinWreath graph. We also have to 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, $C(u)$ 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 committees $C(k)$ had selected $C(u)$ in the previous phase, committee $C(u)$ matches committees $C(k)$ in pairs. If the number of committees $C(k)$ 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 $C(u)$ learns with which committee it has been matched. 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 $UID_u > UID_v$ and vice versa. Committee $C(u)$ enters the Leader Merging mode.
- **Leader Merging:** Given that in the previous phase committee $C(u)$ lost to committee $C(k)$, the leader of $C(u)$ activates an edge with the leader of $C(k)$. If committee $C(k)$ has lost to some other committee $C(l)$ 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 LineToCompletePolylogarithmicTree 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 have now merged with committee $C'(u)$ whose leader 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 spanning complete polylogarithmic tree subgraph.

Theorem 5.1. *For any initial connected graph with polylogarithmic degree, the GraphToThinWreath algorithm solves Depth-*$\log n$ *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.*

6 LOWER BOUNDS FOR THE DEPTH-$\log n$ TREE PROBLEM

Lemma 6.1. *If the initial graph* G_s *is a spanning line, any centralized transformation strategy requires* $\Omega(\log n)$ *rounds to solve Depth-*$\log n$ *Tree.*

Lemma 6.2. *Any centralized transformation strategy that solves Depth-*$\log n$ *Tree in* $O(\log n)$ *rounds, requires* $\Omega(n)$ *edge activations and* $\Omega(n/\log n)$ *edge activations per round.*

On the positive side:

Theorem 6.3. *There is a centralized transformation strategy that, for any initial graph* $D = (V, E)$*, solves Depth-*$\log n$ *Tree in* $O(\log n)$ *rounds, with* $\Theta(n)$ *total edge activations.*

We are now going to show that there is a difference in the minimum total edge activations required for solving the Depth-log n Tree problem between the centralized and the distributed case.

Theorem 6.4. *Any distributed algorithm that solves the Depth-$\log n$ Tree problem in $O(\log n)$ time, requires $\Omega(n \log n)$ total edge activations.*

7 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 potential neighborhoods, e.g., activating edges at larger distances.

REFERENCES

[1] Hugo A Akitaya, Esther M Arkin, Mirela Damian, Erik D Demaine, Vida Dujmovic, Robin Flatland, Matias Korman, Belen Palop, Irene Parada, André van Renssen, et al. 2019. Universal Reconfiguration of Facet-Connected Modular Robots by Pivots: The $O(1)$ Musketeers. In *27th Annual European Symposium on Algorithms (ESA)*.

[2] Abdullah Almethen, Othon Michail, and Igor Potapov. 2020. Pushing Lines Helps: Efficient Universal Centralised Transformations for Programmable Matter. *Theoretical Computer Science* 830-831 (2020), 43 – 59.

[3] Dana Angluin, James Aspnes, Jiang Chen, Yinghua Wu, and Yitong Yin. 2005. Fast Construction of Overlay Networks. In *17th ACM symposium on Parallelism in Algorithms and Architectures (SPAA)*. 145–154.

[4] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. 2006. Computation in Networks of Passively Mobile Finite-state Sensors. *Distrib. Comput.* 18, 4 (2006), 235–253.

[5] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. 2007. The Computational Power of Population Protocols. *Distrib. Comput.* 20, 4 (2007), 279–304.

[6] James Aspnes and Eric Ruppert. 2009. An Introduction to Population Protocols. In *Middleware for Network Eccentric and Mobile Applications*. Springer, 97–120.

[7] James Aspnes and Gauri Shah. 2007. Skip Graphs. *ACM Transactions on Algorithms (TALG)* 3, 4 (2007), 37.

[8] James Aspnes and Yinghua Wu. 2007. $O(\log n)$-Time Overlay Network Construction from Graphs with Out-Degree 1. In *11th International Conference On Principles Of Distributed Systems (OPODIS)*. 286–300.

[9] Kenneth A. Berman. 1996. Vulnerability of scheduled networks and a generalization of Menger's Theorem. *Networks* 28, 3 (1996), 125–134.

[10] Julien Bourgeois and Seth Copen Goldstein. 2011. Distributed Intelligent MEMS: Progresses and Perspectives. In *International Conference on ICT Innovations*. 15–25.

[11] Zahra Derakhshandeh, Robert Gmyr, Andréa W Richa, Christian Scheideler, and Thim Strothmann. 2016. Universal Shape Formation for Programmable Matter. In *28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 289–299.

[12] Jessica Enright, Kitty Meeks, George B. Mertzios, and Viktor Zamaraev. 2019. Deleting Edges to Restrict the Size of an Epidemic in Temporal Networks. In *44th International Symposium on Mathematical Foundations of Computer Science (MFCS)*. 57:1–57:15.

[13] Sándor Fekete, Andréa W Richa, Kay Römer, and Christian Scheideler. 2016. Algorithmic Foundations of Programmable Matter (Dagstuhl Seminar 16271). In *Dagstuhl Reports*, Vol. 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. Also in *ACM SIGACT News*, 48.2:87-94, 2017.

[14] Robert Gmyr, Kristian Hinnenthal, Christian Scheideler, and Christian Sohler. 2017. Distributed Monitoring of Network Properties: The Power of Hybrid Networks. In *44th International Colloquium on Automata, Languages, and Programming (ICALP)*. 137:1–137:15.

[15] Thorsten Götte, Kristian Hinnenthal, and Christian Scheideler. 2019. Faster Construction of Overlay Networks. In *26th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*. 262–276.

[16] David Kempe, Jon Kleinberg, and Amit Kumar. 2000. Connectivity and Inference Problems for Temporal Networks. In *32nd ACM Symposium on Theory of Computing (STOC)*. 504–513.

[17] Fabian Kuhn, Nancy Lynch, and Rotem Oshman. 2010. Distributed Computation in Dynamic Networks. In *42nd ACM Symposium on Theory of Computing (STOC)*. 513–522.

[18] Michael Andrew McEvoy and Nikolaus Correll. 2015. Materials that Couple Sensing, Actuation, Computation, and Communication. *Science* 347, 6228 (2015), 1261689.

[19] George B Mertzios, Othon Michail, and Paul G Spirakis. 2019. Temporal Network Optimization Subject to Connectivity Constraints. *Algorithmica* 81, 4 (2019), 1416–1449.

[20] Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. 2011. Mediated Population Protocols. *Theoretical Computer Science* 412, 22 (2011), 2434–2450.

[21] Othon Michail, Ioannis Chatzigiannakis, and Paul G Spirakis. 2013. Naming and Counting in Anonymous Unknown Dynamic Networks. In *15th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. 281–295.

[22] Othon Michail, George Skretas, and Paul G Spirakis. 2018. On the Transformation Capability of Feasible Mechanisms for Programmable Matter. *J. Comput. System Sci.* 102 (2018), 18–39.

[23] Othon Michail and Paul G. Spirakis. 2016. Simple and Efficient Local Codes for Distributed Stable Network Construction. *Distrib. Comput.* 29, 3 (2016), 207–237.

[24] Othon Michail and Paul G Spirakis. 2017. Connectivity Preserving Network Transformers. *Theoretical Computer Science* 671 (2017), 36–55.

[25] Regina O'Dell and Roger Wattenhofer. 2005. Information Dissemination in Highly Dynamic Graphs. In *Joint Workshop on Foundations of Mobile Computing (DIALM-POMC)*. 104–110.

[26] Christian Scheideler and Alexander Setzer. 2019. On the Complexity of Local Graph Transformations. In *46th International Colloquium on Automata, Languages, and Programming (ICALP)*. 150:1–150:14.

[27] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. 2001. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. *ACM SIGCOMM Computer Communication Review* 31, 4 (2001), 149–160.