Time-Message Trade-Offs in Distributed Algorithms

Robert Gmyr
Department of Computer Science, University of Houston, USA
rgmyr@uh.edu

Gopal Pandurangan
Department of Computer Science, University of Houston, USA
gopalpandurangan@gmail.com

Abstract

This paper focuses on showing time-message trade-offs in distributed algorithms for fundamental problems such as leader election, broadcast, spanning tree (ST), minimum spanning tree (MST), minimum cut, and many graph verification problems. We consider the synchronous CONGEST distributed computing model and assume that each node has initial knowledge of itself and the identifiers of its neighbors – the so-called *KT_1 model* – a well-studied model that also naturally arises in many applications. Recently, it has been established that one can obtain (almost) *singularly optimal* algorithms, i.e., algorithms that have *simultaneously* optimal time and message complexity (up to polylogarithmic factors), for many fundamental problems in the standard KT_0 model (where nodes have only local knowledge of themselves and not their neighbors). The situation is less clear in the KT_1 model. In this paper, we present several new distributed algorithms in the KT_1 model that trade off between time and message complexity.

Our distributed algorithms are based on a uniform and general approach which involves constructing a *sparsified spanning subgraph* of the original graph – called a *danner* – that trades off the *number of edges* with the *diameter* of the sparsifier. In particular, a key ingredient of our approach is a distributed randomized algorithm that, given a graph G and any $\delta \in [0, 1]$, with high probability[1] constructs a danner that has diameter $\tilde{O}(D + n^{1-\delta})$ and $\tilde{O}(\min\{m, n^{1+\delta}\})$ edges in $\tilde{O}(n^{1-\delta})$ rounds while using $\tilde{O}(\min\{m, n^{1+\delta}\})$ messages, where n, m, and D are the number of nodes, edges, and the diameter of G, respectively.[2] Using our danner construction, we present a family of distributed randomized algorithms for various fundamental problems that exhibit a trade-off between message and time complexity and that improve over previous results. Specifically, we show the following results (all hold with high probability) in the KT_1 model, which subsume and improve over prior bounds in the KT_1 model (King et al., PODC 2014 and Awerbuch et al., JACM 1990) and the KT_0 model (Kutten et al., JACM 2015, Pandurangan et al., STOC 2017 and Elkin, PODC 2017):

1. **Leader Election, Broadcast, and ST.** These problems can be solved in $\tilde{O}(D + n^{1-\delta})$ rounds using $\tilde{O}(\min\{m, n^{1+\delta}\})$ messages for any $\delta \in [0, 1]$.
2. **MST and Connectivity.** These problems can be solved in $\tilde{O}(D + n^{1-\delta})$ rounds using $\tilde{O}(\min\{m, n^{1+\delta}\})$ messages for any $\delta \in [0, 0.5]$. In particular, for $\delta = 0.5$ we obtain a distributed MST algorithm that runs in optimal $\tilde{O}(D + \sqrt{n})$ rounds and uses $\tilde{O}(\min\{m, n^{3/2}\})$ messages. We note that this improves over the singularly optimal algorithm in the KT_0 model that uses $\tilde{O}(D + \sqrt{n})$ rounds and $\tilde{O}(m)$ messages.
3. **Minimum Cut.** $O(\log n)$-approximate minimum cut can be solved in $\tilde{O}(D + n^{1-\delta})$ rounds using $\tilde{O}(\min\{m, n^{1+\delta}\})$ messages for any $\delta \in [0, 0.5]$.
4. **Graph Verification Problems such as Bipartiteness, Spanning Subgraph etc.** These can be solved in $\tilde{O}(D + n^{1-\delta})$ rounds using $\tilde{O}(\min\{m, n^{1+\delta}\})$ messages for any $\delta \in [0, 0.5]$.

[1] Throughout, by "with high probability (w.h.p.)" we mean with probability at least $1 - 1/n^c$ where n is the network size and c is some constant.
[2] The notation $\tilde{O}$ hides a polylog(n) factor.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Randomized Algorithm, KT_1 Model, Sparsifier, MST, Singular Optimality

Digital Object Identifier 10.4230/LIPIcs.DISC.2018.32

Funding Supported, in part, by NSF awards CCF-1527867, CCF-1540512, IIS-1633720, CCF-1717075, and BSF award 2016419.

1 Introduction

This paper focuses on a fundamental aspect of distributed algorithms: *trade-offs* for the two basic complexity measures of *time* and *messages*. The efficiency of distributed algorithms is traditionally measured by their time and message (or communication) complexities. Both complexity measures crucially influence the performance of a distributed algorithm. Time complexity measures the number of distributed "rounds" taken by the algorithm and determines the running time of the algorithm. Obviously, it is of interest to keep the running time as small as possible. Message complexity, on the other hand, measures the total amount of messages sent and received by all the processors during the course of the algorithm. In many applications, this is the dominant cost that also plays a major role in determining the running time and additional costs (e.g., energy) expended by the algorithm. For example, communication cost is one of the dominant costs in the distributed computation of large-scale data [17]. In another example, in certain types of communication networks, such as ad-hoc wireless networks, energy is a critical factor for measuring the efficiency of a distributed algorithm [15, 4]. Transmitting a message between two nodes in such a network has an associated energy cost and hence the total message complexity plays an important role in determining the energy cost of an algorithm. At the same time, keeping the number of rounds small also helps in reducing the energy cost. Thus, in various modern and emerging applications such as resource-constrained communication networks and distributed computation on large-scale data, it is crucial to design distributed algorithms that optimize both measures *simultaneously* [13, 17, 24, 26].

Unfortunately, designing algorithms that are simultaneously time- and message-efficient has proven to be a challenging task, which (for some problems) stubbornly defied all prior attempts of attack. Consequently, research in the last three decades in the area of distributed algorithms has focused mainly on optimizing either one of the two measures separately, typically at the cost of neglecting the other. In this paper, we focus on studying the two cost measures of time and messages *jointly*, and exploring ways to design distributed algorithms that work well under both measures (to the extent possible). Towards this goal, it is important to understand the relationship between these two measures. In particular, as defined in [25], we should be able to determine, for specific problems, whether it is possible to devise a distributed algorithm that is either *singularly optimal* or exhibits a *time-message trade-off*:

- **Singularly optimal:** A distributed algorithm that is optimal with respect to both measures simultaneously – in which case we say that the problem enjoys *singular optimality*.
- **Time-message trade-off:** Whether the problem inherently fails to admit a singularly optimal solution, namely, algorithms of better time complexity for it will necessarily incur higher message complexity and vice versa – in which case we say that the problem exhibits a *time-message trade-off*.

We note that, more generally, finding a simultaneously optimal (or almost optimal) solution may sometimes be difficult even for "singular optimality" problems and hence it might be useful to first design algorithms that have trade-offs.

This paper focuses on showing time-message trade-offs in distributed algorithms for fundamental problems such as leader election, broadcast, spanning tree (ST), minimum spanning tree (MST), minimum cut, and many graph verification problems. Throughout, we consider the synchronous CONGEST model (see Section 2.2 for details), a standard model in distributed computing where computation proceeds in discrete (synchronous) *rounds* and in each round only $O(\log n)$ bits are allowed to be exchanged per edge (CONGEST) where n is the number of nodes in the network. It turns out that message complexity of a distributed algorithm depends crucially (as explained below) on the initial knowledge of the nodes; in this respect, there are two well-studied models – the KT_0 and the KT_1 model.[3] In the $\mathbf{KT}_0$ model (i.e., **K**nowledge **T**ill radius **0**), also called the *clean network model* [27], where nodes have initial local knowledge of only themselves (and not their neighbors), it has (only) been recently established that one can obtain (almost) *singularly optimal* algorithms, i.e., algorithms that have *simultaneously* optimal time and message complexity (up to polylogarithmic factors), for many fundamental problems such as leader election, broadcast, ST, MST, minimum cut, and approximate shortest paths (under some conditions) [18, 25, 7, 12]. More precisely, for problems such as leader election, broadcast, and ST, it has been shown [18] that one can design a singularly optimal algorithm in the KT_0 model, that takes $\tilde{O}(m)$ messages and $O(D)$ rounds – both are tight (up to a polylog(n) factor); this is because $\Omega(m)$ and $\Omega(D)$ are, respectively, lower bounds on the message and time complexity for these problems in the KT_0 model [18] – note that these lower bounds hold even for randomized Monte Carlo algorithms. The work of [25] (also see [7]) showed that MST is also (almost) singularly optimal, by giving a (randomized Las Vegas) algorithm that takes $\tilde{O}(m)$ messages and $\tilde{O}(D + \sqrt{n})$ rounds (both bounds are tight up to polylogarithmic factors). It can be shown that the singular optimality of MST in the KT_0 model also implies the singular optimality of many other problems such as approximate minimum cut and graph verification problems (see Section 1.2). Recently, it was shown that approximate shortest paths and several other problems also admit singular optimality in the KT_0 model [12].

On the other hand, in the $\mathbf{KT}_1$ model (i.e., **K**nowledge **T**ill radius **1**), in which each node has initial knowledge of itself and the *identifiers*[4] of its neighbors, the situation is less clear. The KT_1 model arises naturally in many settings, e.g., in networks where nodes know the identifiers of their neighbors (as well as other nodes), e.g., in the Internet where a node knows the IP addresses of other nodes [23]. Similarly in models such as the k-machine model (as well as the congested clique), it is natural to assume that each processor knows the identifiers of all other processors [17, 13, 24, 26]. For the KT_1 model, King et al. [16] showed a surprising and elegant result: There is a randomized Monte Carlo algorithm to construct an MST in $\tilde{O}(n)$ messages ($\Omega(n)$ is a message lower bound) and in $\tilde{O}(n)$ time (see Section 2.3). Thus it is also possible to construct an ST, do leader election, and broadcast within these bounds.

[3] On the other hand, for time complexity it does not really matter whether nodes have initial knowledge of just themselves (KT_0) or also of their neighbors (KT_1); this is because this information (i.e., the identifiers of the neighbors) can be exchanged in one round in the CONGEST model. Hence, when focusing solely on optimizing time complexity, which is the typically the case in the literature, the distinction between KT_0 and KT_1 is not important and the actual model is not even explicitly specified.

[4] Note that only knowledge of the identifiers of neighbors is assumed, not other information such as the degree of the neighbors.

This algorithm is randomized and *not comparison-based.*[5] While this algorithm shows that one can achieve $o(m)$ message complexity (when $m = \omega(n \operatorname{polylog} n)$), it is *not* time-optimal – it can take significantly more than $\tilde{\Theta}(D + \sqrt{n})$ rounds, which is a time lower bound even for Monte-Carlo randomized algorithms [5]. In subsequent work, Mashreghi and King [19] presented a trade-off between messages and time for MST: a Monte-Carlo algorithm that takes $\tilde{O}(n^{1+\epsilon}\epsilon)$ messages and runs in $O(n/\epsilon)$ time for any $1 > \epsilon \geq \log\log n/\log n$. We note that this algorithm takes at least $O(n)$ time.

A central motivating theme underlying this work is understanding the status of various fundamental problems in the KT_1 model – whether they are singularly optimal or exhibit trade-offs (and, if so, to quantify the trade-offs). In particular, it is an open question whether one can design a randomized (non-comparison based) MST algorithm that takes $\tilde{O}(D + \sqrt{n})$ time and $\tilde{O}(n)$ messages in the KT_1 model – this would show that MST is (almost) singularly optimal in the KT_1 model as well. In fact, King et al [16] ask whether it is possible to construct (even) an ST in $o(n)$ rounds with $o(m)$ messages. Moreover, can we take advantage of the KT_1 model to get improved message bounds (while keeping time as small as possible) in comparison to the KT_0 model?

1.1 Our Contributions and Comparison with Related Work

In this paper, we present several results that show trade-offs between time and messages in the KT_1 model with respect to various problems. As a byproduct, we improve and subsume the results of [16] (as well as of Awerbuch et al. [2]) and answer the question raised by King et al. on ST/MST construction at the end of the previous paragraph in the affirmative. We also show that our results give improved bounds compared to the results in the KT_0 model, including for the fundamental distributed MST problem.

Our time-message trade-off results are based on a uniform and general approach which involves constructing a *sparsified spanning subgraph* of the original graph – called a *danner* (i.e., "diameter-preserving spanner") – that trades off the *number of edges* with the *diameter* of the sparsifier (we refer to Section 2.1 for a precise definition). *In particular, a key ingredient of our approach is a distributed randomized algorithm that, given a graph G and any $\delta \in [0,1]$, with high probability*[6] *constructs a danner that has diameter $\tilde{O}(D + n^{1-\delta})$ and $\tilde{O}(\min\{m, n^{1+\delta}\})$ edges in $\tilde{O}(n^{1-\delta})$ rounds while using $\tilde{O}(\min\{m, n^{1+\delta}\})$ messages, where n, m, and D are the number of nodes, edges, and the diameter of G, respectively.*[7] Using our danner construction, we present a family of distributed randomized algorithms for various fundamental problems that exhibit a trade-off between message and time complexity and that improve over previous results. Specifically, we show the following results (all hold with high probability) in the KT_1 model (cf. Section 4):

1. **Leader Election, Broadcast, and ST.** These problems can be solved in $\tilde{O}(D + n^{1-\delta})$ rounds using $\tilde{O}(\min\{m, n^{1+\delta}\})$ messages for any $\delta \in [0,1]$. These results improve over prior bounds in the KT_1 model [16, 19, 2] as well the KT_0 model [18] – discussed earlier

[5] Awerbuch et al. [2] show that $\Omega(m)$ is a message lower bound for MST even in the KT_1 model, if one allows only (possibly randomized Monte Carlo) *comparison-based* algorithms, i.e., algorithms that can operate on identifiers only by comparing them. The result of King et al. [16] breaches the $\Omega(m)$ lower bound by using non-comparison-based technique by using the identifiers as input to hash functions. Our results also breach the $\Omega(m)$ lower bound, since we use the results of King et al. as subroutines in our algorithms.

[6] Throughout, by "with high probability (w.h.p.)" we mean with probability at least $1 - 1/n^c$ where n is the network size and c is some constant.

[7] The notation $\tilde{O}$ hides a polylog(n) factor.

in Section 1. In particular, while the time bounds in [16, 19] are always at least linear, our bounds can be sublinear and the desired time-message trade-off can be obtained by choosing an appropriate δ. It is worth noting that the early work of Awerbuch et al. [2] showed that broadcast can be solved by a deterministic algorithm in the KT_ρ model using $O(\min\{m, n^{1+c/\rho}\})$ messages for some fixed constant $c > 0$ in a model where each node has knowledge of the *topology* (not just identifiers) up to radius ρ. Clearly, our results improve over this (for the KT_1 model), since δ can be made arbitrarily small.

2. **MST and Connectivity.** These problems can be solved in $\tilde{O}(D + n^{1-\delta})$ rounds using $\tilde{O}(\min\{m, n^{1+\delta}\})$ messages for any $\delta \in [0, 0.5]$. In addition to getting any desired trade-off (by plugging in an appropriate δ), we can get a time optimal algorithm by choosing $\delta = 0.5$, which results in a distributed MST algorithm that runs in $\tilde{O}(D + \sqrt{n})$ rounds and uses $\tilde{O}(\min\{m, n^{3/2}\})$ messages. We note that when $m = \tilde{\omega}(n^{3/2})$, this improves over the recently proposed singularly optimal algorithms of [25, 7] in the KT_0 model that use $\tilde{O}(m)$ messages and $\tilde{O}(D + \sqrt{n})$ rounds. It also subsumes and improves over the prior results of [19, 16] in the KT_1 model that take (essentially) $\tilde{O}(n)$ messages and $\tilde{O}(n)$ time.[8]
3. **Minimum Cut.** An $O(\log n)$-approximation to the minimum cut value (i.e., edge connectivity of the graph) can be obtained in $\tilde{O}(D+n^{1-\delta})$ rounds using $\tilde{O}(\min\{m, n^{1+\delta}\})$ messages for any $\delta \in [0, 0.5]$. Our result improves over the works of [11, 21] that are (almost) time-optimal (i.e., take $\tilde{O}(D + \sqrt{n})$ rounds), but not message optimal. In addition to getting any desired trade-off (by plugging in an appropriate δ), in particular, if $\delta = 0.5$, we obtain a $\tilde{O}(\min\{m, n^{3/2}\})$ messages approximate minimum cut algorithm that runs in (near) optimal $\tilde{O}(D + \sqrt{n})$ rounds. This improves the best possible bounds (for $m = \tilde{\omega}(n^{3/2})$) that can be obtained in the KT_0 model (cf. Section 1.2).
4. **Graph Verification Problems such as Bipartiteness, $s - t$ Cut, Spanning Subgraph.** These can be solved in $\tilde{O}(D + n^{1-\delta})$ rounds using $\tilde{O}(\min\{m, n^{1+\delta}\})$ messages for any $\delta \in [0, 0.5]$.

1.2 High-Level Overview of Approach

Danner. A main technical contribution of this work is the notion of a *danner* and its efficient distributed construction that jointly focuses on both time and messages. As defined in Section 2.1, a danner of a graph G is a spanning subgraph H of G whose diameter, i.e., $\mathrm{diam}(H)$, is at most $\alpha(\mathrm{diam}(G)) + \beta$, where $\alpha \geq 1$ and $\beta \geq 0$ are some parameters. The goal is to construct a danner H with as few edges as possible and with α and β as small as possible. It is clear that very sparse danners exist: the BFS (breadth-first spanning) tree has only $n - 1$ edges and its diameter is at most twice the diameter of the graph. However, it is not clear how to construct such a danner in a way that is efficient with respect to both messages and time, in particular in $\tilde{O}(n)$ messages and $\tilde{O}(D)$ time, or even $o(m)$ messages and $O(D)$ time, where $D = \mathrm{diam}(G)$. Note that in the KT_0 model, there is a tight lower bound (with almost matching upper bound) for constructing a danner: any distributed danner construction algorithm needs $\Omega(m)$ messages and $\Omega(D)$ time (this follows by reduction from leader election which has these lower bounds [18] – see Section 1). However, in the KT_1 model, the status for danner construction is not known. We give (a family of) distributed algorithms for constructing a danner that trade off messages for time (Section 3).

[8] Mashreghi and King [19] also give an algorithm with round complexity $\tilde{O}(\mathrm{diam}(\mathrm{MST}))$ and with message complexity $\tilde{O}(n)$, where $\mathrm{diam}(\mathrm{MST})$ is the diameter of the output MST which can be as large as $\Theta(n)$.

Danner Construction. We present an algorithm (see Algorithm 1) that, given a graph G and any $\delta \in [0,1]$, constructs a danner H of G that has $\tilde{O}(\min\{m, n^{1+\delta}\})$ edges and diameter $\tilde{O}(D + n^{1-\delta})$ (i.e., an $\tilde{O}(n^{1-\delta})$ additive danner) using $\tilde{O}(\min\{m, n^{1+\delta}\})$ messages and in $\tilde{O}(n^{1-\delta})$ time (note that the time does not depend on D). The main idea behind the algorithm is as follows. While vertices with low degree (i.e., less than n^{δ}) and their incident edges can be included in the danner H, to handle high-degree vertices we construct a dominating set that dominates the high-degree nodes by sampling roughly $n^{1-\delta}$ nodes (among all nodes); these are called "center" nodes.[9] Each node v adds the edges leading to its $\min\{\deg(v), n^{\delta}\}$ neighbors with the lowest identifiers (required for maintaining independence from random sampling) to H. It is not difficult to argue that each high-degree node is connected to a center in H and we use a relationship between the number of nodes in any dominating set and the diameter (cf. Lemma 5) to argue that the diameter of each connected component (or fragment) of H is at most $\tilde{O}(n^{1-\delta})$. We then use the FindAny algorithm of King et al. [16] to efficiently implement a distributed Boruvka-style merging (which is essentially the GHS algorithm [9]) of fragments in the subgraph induced by the high-degree nodes and the centers. The FindAny algorithm does not rely on identifier comparisons but instead uses random hash-functions to find an edge leaving a set of nodes very efficiently, which is crucial for our algorithm. In each merging phase, each fragment uses FindAny to efficiently find an outgoing edge; discovered outgoing edges are added to H. Only $O(\log n)$ iterations are needed to merge all fragments into a connected graph and only $\tilde{O}(\min\{m, n^{1+\delta}\})$ messages are needed overall for merging. At any time the set of centers in a fragment forms a dominating set of that fragment. Thereby, the above-mentioned relationship between dominating sets and diameters guarantees that the diameters of the fragments stay within $\tilde{O}(n^{1-\delta})$. We argue (Lemma 12) that the constructed subgraph H is an additive $\tilde{O}(n^{1-\delta})$-danner of G.

Danner Applications. What is the motivation for defining a danner and why is it useful? The answer to both of these questions is that a danner gives a uniform way to design distributed algorithms that are both time and message efficient for various applications as demonstrated in Section 4. Results for leader election, broadcast, and ST construction follow quite directly (cf. Section 4.1): Simply construct a danner and run the singularly optimal algorithm of [18] for the KT_0 model on the danner subgraph. Since the danner has $\tilde{O}(n^{1+\delta})$ edges and has diameter $\tilde{O}(D + n^{1-\delta})$, this gives the required bounds.

A danner can be used to construct a MST of a graph G (which also gives checking connectivity of a subgraph H of G) using $\tilde{O}(\min\{m, n^{1+\delta}\})$ messages in $\tilde{O}(D + n^{1-\delta})$ time, for any $\delta \in [0, 0.5]$. Note that this subsumes the bounds of the singularly optimal algorithms in the KT_0 model [25, 7]. The distributed MST construction (cf. Section 4.2) proceeds in three steps; two of these crucially use the danner. In the first step, we construct a danner and use it as a communication backbone to aggregate the degrees of all nodes and thus determine m, the number of edges. If $m \leq n^{1+\delta}$, then we simply proceed to use the singularly optimal algorithm of [25]. Otherwise, we proceed to Step 2, where we do *Controlled-GHS* which is a well-known ingredient in prior MST algorithms [22, 10, 25]. Controlled-GHS is simply Boruvka-style MST algorithm, where the diameter of all the fragments grow at a controlled rate. We use the graph sketches technique of King et al. (in particular the FindMin algorithm – cf. Section 2.3) for finding outgoing edges to keep the message complexity to $\tilde{O}(n)$. Crucially we run Controlled-GHS to only $\lceil (1-\delta)\log n \rceil$ iterations so that the number of fragments

[9] The idea of establishing a set of nodes that dominates all high-degree nodes has also been used by Aingworth et al. [1] and Dor et al. [6].

remaining at the end of Controlled-GHS is $O(n^\delta)$ with each having diameter $\tilde{O}(n^{1-\delta})$; all these take only $\tilde{O}(n^{1-\delta})$ time, since the running time of Controlled-GHS is asymptotically bounded (up to a $\log n$ factor) by the (largest) diameter of any fragment. In Step 3, we merge the remaining $O(n^\delta)$ fragments; this is done in a way that is somewhat different to prior MST algorithms, especially those of [25, 10]. We simply continue the Boruvka-merging (not necessarily in a controlled way), but instead of using each fragment as the communication backbone, we do the merging "globally" using a BFS tree of the danner subgraph. The BFS tree of the danner has $O(n)$ edges and has diameter $\tilde{O}(D + n^{1-\delta})$. In each merging phase, each node forwards at most $O(n^\delta)$ messages (sketches corresponding to so many distinct fragments) to the root of the BFS tree which finds the outgoing edge corresponding to each fragment (and broadcasts it to all the nodes). The total message cost is $\tilde{O}(n^{1+\delta})$ and, since the messages are pipelined, the total time is $\tilde{O}(D + n^{1-\delta} + n^\delta) = \tilde{O}(D + n^{1-\delta})$ (for $\delta = [0, 0.5]$). Building upon this algorithm, we give time and message efficient algorithms for $O(\log n)$-approximate minimum cut and graph connectivity problems (cf. Section 4.3).

1.3 Other Related Work

There has been extensive research on the distributed MST problem in the KT_0 model, culminating in the singularly optimal (randomized) algorithm of [25] (see also [7]); see [25] for a survey of results in this area. The work of [25] also defined the notions of *singular optimality* versus *time-message trade-offs*. Kutten et al. [18] show the singular optimality of leader election (which implies the same for broadcast and ST) by giving tight lower bounds for both messages and time as well as giving algorithms that simultaneously achieve the tight bounds (see Section 1).

Compared to the KT_0 model, results in the KT_1 are somewhat less studied. The early work of Awerbuch et al. [2] studied time-message trade-offs for broadcast in the KT_1 model. In 2015, King et al. [16] showed surprisingly that the basic $\Omega(m)$ message lower bound that holds in the KT_0 model for various problems such as leader election, broadcast, MST, etc. [18] can be breached in the KT_1 model by giving a randomized Monte Carlo algorithm to construct an MST in $\tilde{O}(n)$ messages and in $\tilde{O}(n)$ time. The algorithm of King et al. uses a powerful randomized technique of *graph sketches* which helps in identifying edges going out of a cut efficiently without probing all the edges in the cut; this crucially helps in reducing the message complexity. We heavily use this technique (as a black box) in our algorithms as well. However, note that we could have also used other types of graph sketches (see e.g., [24]) which will yield similar results.

The KT_1 model has been assumed in other distributed computing models such as the k-machine model [17, 24, 26] and the congested clique [13]. In [13] it was shown that the MST problem has a message lower bound of $\Omega(n^2)$ which can be breached in the KT_1 model by using graph sketches.

Distributed minimum cut has been studied by [11, 21], and the graph verification problems considered in this paper have been studied in [5]. However, the focus of these results has been on the time complexity (where KT_0 or KT_1 does not matter). We study these problems in the KT_1 model focusing on both time and messages and present trade-off algorithms that also improve over the KT_0 algorithms (in terms of messages) – cf. Section 1.1. We note that $\tilde{\Omega}(D + \sqrt{n})$ is a time lower bound for minimum cut (for any non-trivial approximation) and for the considered graph verification problems [5]. It can be also shown (by using techniques from [18]) that $\Omega(m)$ is a message lower bound in the KT_0 model for minimum cut.

2 Preliminaries

Before we come to the main technical part of the paper, we introduce some notation and basic definitions, present our network model, and give an overview of some of the algorithms from the literature that we use for our results.

2.1 Notation and Definitions

For a graph G we denote its *node set* as $V(G)$ and its *edge set* as $E(G)$. For a node $u \in V(G)$ the set $N_G(u) = \{v \in V(G) \mid \{u,v\} \in E(G)\}$ is the *open neighborhood* of u in G and $\Gamma_G(u) = N_G(u) \cup \{u\}$ is its *closed neighborhood.* For a set of nodes $S \subseteq V(G)$ we define $\Gamma_G(S) = \bigcup_{u \in S} \Gamma_G(u)$. The *degree* of a node u in G is $\deg_G(u) = |N_G(u)|$. For a path $P = (u_0, \ldots, u_\ell)$ we define $V(P)$ to be the set of nodes in P and we define $|P| = \ell$ to be the *length* of P. A set $S \subseteq V(G)$ is a *dominating set* of G if $\Gamma_G(S) = V(G)$. The *domination number* $\gamma(G)$ of a graph G is the size of a smallest dominating set of G. The *distance* $d_G(u,v)$ between two nodes $u, v \in V(G)$ is the length of a shortest path between u and v in G. We define the *diameter* (or the *hop diameter*) of G as $\mathrm{diam}(G) = \max_{u,v \in V(G)} d(u,v)$, where the distances are taken in the graph by ignoring edge weights (i.e., each edge has weight 1). For all of this notation, we omit G when it is apparent from context. For $S \subseteq V(G)$ the *induced subgraph* $G[S]$ is defined by $V(G[S]) = S$ and $E(G[S]) = \{\{u,v\} \in E(G) \mid u, v \in S\}$. A subgraph $H \subseteq G$ is an (α, β)-*spanner* of G if $V(H) = V(G)$ and $d_H(u,v) \leq \alpha \cdot d_G(u,v) + \beta$ for all $u, v \in V(G)$. In this work we make use of the weaker concept of a *diameter-preserving spanner*, or in short, *danner*: A subgraph $H \subseteq G$ is a (α, β)-*danner* of G if $V(H) = V(G)$ and $\mathrm{diam}(H) \leq \alpha \cdot \mathrm{diam}(G) + \beta$. We say H is an additive β-danner if it is a $(1, \beta)$-danner. An (α, β)-spanner is also an (α, β)-danner but the reverse is not generally true. Hence, the notion of a danner is weaker than that of a spanner.

2.2 Model

We briefly describe the distributed computing model used. This is the synchronous CONGEST model (see, e.g., [22, 27]), which is now standard in the distributed computing literature.

A point-to-point communication network is modeled as an undirected weighted graph $G = (V, E, w)$, where the vertices of V represent the processors, the edges of E represent the communication links between them, and $w(e)$ is the weight of edge $e \in E$. Let $n = |V(G)|$ and $m = |E(G)|$. Without loss of generality, we assume that G is connected. D denotes the hop-diameter (that is, the unweighted diameter) of G, and, in this paper, by diameter we always mean hop-diameter. Each node hosts a processor with limited initial knowledge. Specifically, we make the common assumption that each node has a unique identifier (from $\{1, \ldots, \mathrm{poly}(n)\}$), and at the beginning of computation each vertex v accepts as input its own identifier and the weights (if any) of the edges incident to it as well as the *identifiers of all its neighbors*. Thus, a node has *local* knowledge of only itself and its neighbor's identifiers; this is called the KT_1 *model.* Since each node knows the identifier of the node on the other side of an incident edge, both endpoints can define a *common* edge identifier as the concatenation of identifiers of its endpoints, lowest identifier first.

The vertices are allowed to communicate through the edges of the graph G. It is assumed that communication is synchronous and occurs in discrete rounds (time steps). In each time step, each node v can send an arbitrary message of $O(\log n)$ bits through each edge $e = \{v, u\}$ incident to v, and each message arrives at u by the end of this time step. The weights of the edges are at most polynomial in the number of vertices n, and therefore the weight of a single edge can be communicated in one time step. This model of distributed computation is called the CONGEST($\log n$) model or simply the CONGEST model [22, 27]. We also assume that each vertex has access to the outcome of unbiased private coin flips. We assume that all nodes know n.

2.3 Underlying Algorithms

We use an algorithm called *TestOut* that was introduced by King et al. [16] in the context of computing MSTs in the KT$_1$ model. Consider a tree T that is a subgraph of a graph G. The algorithm TestOut allows the nodes in T to determine whether there exists an *outgoing* edge, i.e., an edge that connects a node in $V(T)$ with a node in $V(G) \setminus V(T)$. We also refer to an outgoing edge as an edge *leaving* T. Let u be a node in T that initiates an execution of TestOut. On a high level, TestOut simply performs a single *broadcast-and-echo* operation: First, the node u broadcasts a random hash function along T. Each node in T computes a single bit of information based on the hash function and the identifiers of the incident edges. The parity of these bits is then aggregated using an echo (or convergecast) along T. The parity is 1 with constant probability if there is an edge in G leaving T, and it is 0 otherwise. The algorithm is always correct if the parity is 1. The running time of the algorithm is $O(\text{diam}(T))$ and it uses $O(|V(T)|)$ messages.

The correctness probability of TestOut can be amplified to high probability by executing the algorithm $O(\log n)$ times. Furthermore, TestOut can be combined with a binary search on the edge identifiers to find the identifier of an outgoing edge if it exists, which adds another multiplicative factor of $O(\log n)$ to the running time and the number of messages used by the algorithm. Finally, the procedure can also be used to find the identifier of an outgoing edge with minimum weight in a weighted graph by again using binary search on the edge weights at the cost of another multiplicative $O(\log n)$ factor. All of these algorithms can be generalized to work on a connected subgraph H that is not necessarily a tree: A node $u \in V(H)$ first constructs a breadth-first search tree T in H and then executes one of the algorithms described above on T. We have the following theorems.

▶ **Theorem 1.** *Consider a connected subgraph H of a graph G. There is an algorithm* FindAny *that outputs the identifier of an arbitrary edge in G leaving H if there is such an edge and that outputs $\emptyset$ otherwise, w.h.p. The running time of the algorithm is $\tilde{O}(diam(H))$ and it uses $\tilde{O}(|E(H)|)$ messages.*

▶ **Theorem 2.** *Consider a connected subgraph H of a weighted graph G with edge weights from $\{1, \ldots, \text{poly}(n)\}$. There is an algorithm* FindMin *that outputs the identifier of a lightest edge in G leaving H if there is such an edge and that outputs $\emptyset$ otherwise, w.h.p. The running time of the algorithm is $\tilde{O}(diam(H))$ and it uses $\tilde{O}(|E(H)|)$ messages.*

We also require an efficient leader election algorithm. The following theorem is a reformulation of Corollary 4.2 in [18].

▶ **Theorem 3.** *There is an algorithm that for any graph G elects a leader in $O(diam(G))$ rounds while using $\tilde{O}(|E(G)|)$ messages, w.h.p.*

3 Distributed Danner Construction

The distributed danner construction presented in Algorithm 1 uses a parameter δ that controls a trade-off between the time and message complexity of the algorithm. At the same time the parameter controls a trade-off between the diameter and the size (i.e., the number of

Algorithm 1 Distributed Danner Construction.

```
The algorithm constructs a danner $H$. Initially, we set $V(H) \leftarrow V(G)$ and $E(H) \leftarrow \emptyset$.
1. Each node becomes a *center* with probability $p = \min\{(c \log n)/n^\delta, 1\}$, where $c \geq 1$ is a
   constant determined in the analysis. Let $C$ be the set of centers.
2. Each node $v$ adds the edges leading to its $\min\{\deg(v), n^\delta\}$ neighbors with the lowest
   identifiers to $H$.
3. Each low-degree node sends a message to all its neighbors to inform them about its low
   degree and whether it is a center. The remaining steps operate on the induced subgraphs
   $\hat{G} \leftarrow G[V_{\text{high}} \cup C]$ and $\hat{H} \leftarrow H[V_{\text{high}} \cup C]$. Note that every node can deduce which of its
   neighbors lie in $V_{\text{high}} \cup C$ from the messages sent by the low-degree nodes.
4. For $i = 1$ to $\log n$ do the following in parallel in each connected component $K$ of $\hat{H}$.
   a. Elect a leader using the algorithm from Theorem 3.
   b. Use the algorithm FindAny from Theorem 1 to find an edge in $\hat{G}$ leaving $K$. If such
      an edge exists, add it to $H$ and $\hat{H}$.
   c. Wait until $T$ rounds have passed in this iteration before starting the next iteration in
      order to synchronize the execution between the connected components. The value of
      $T$ is determined in the analysis.
```

edges) of the resulting danner. For Algorithm 1 we assume that $\delta \in [0, 1)$. We explicitly treat the case $\delta = 1$ later on. We say a node u has *high degree* if $\deg(u) \geq n^\delta$. Otherwise, it has *low degree*. Let V_{high} and V_{low} be the set of high-degree and low-degree nodes, respectively.

We now turn to the analysis of Algorithm 1. We assume that the probability p defined in Step 1 is such that $p < 1$ since for $p = 1$ the analysis becomes trivial. Our first goal is to bound the diameter of any connected component K of $\hat{H}$ (defined in Step 3 of Algorithm 1) during any iteration of the loop in Step 4. To achieve this goal, we first show two fundamental lemmas that allow us to bound the diameter of K in terms of its domination number $\gamma(K)$ (see Section 2.1). The main observation behind Lemma 4 was also used by Feige et al. in [8].

▶ **Lemma 4.** *Let P be a shortest path in a graph G. For each node $v \in V(G)$ it holds $|\Gamma(v) \cap V(P)| \leq 3$.*

Proof. We show the lemma by contradiction. Let $P = (u_0, \ldots, u_\ell)$ be a shortest path in G. Suppose there is a node $v \in V(G)$ such that $|\Gamma(v) \cap V(P)| \geq 4$. Let u_i be the node in $\Gamma(v) \cap V(P)$ with the lowest index in P and let u_j be the node in $\Gamma(v) \cap V(P)$ with the highest index in P. Since $|\Gamma(v) \cap V(P)| \geq 4$ at least two nodes lie between u_i and u_j in P. We distinguish two cases. If $u_i = v$ or $u_j = v$ then $P' = (u_0, \ldots, u_i, u_j, \ldots, u_\ell)$ is a path in G such that $|P'| \leq |P| - 2$, which is a contradiction. Otherwise, the path $P' = (u_0, \ldots, u_i, v, u_j \ldots, u_\ell)$ is a path in G such that $|P'| \leq |P| - 1$, which is again a contradiction. ◀

▶ **Lemma 5.** *For a connected graph G it holds diam$(G) < 3\gamma(G)$.*

Proof. We show the lemma by contradiction. Suppose there is a shortest path P in G such that $|P| \geq 3\gamma(G)$. Let S be a dominating set in G with $|S| = \gamma(G)$. By definition, for each node $u \in V(P)$ there is a node $v \in S$ such that $u \in \Gamma(v)$. Since $|V(P)| = |P| + 1 > 3|S|$, the pigeonhole principle implies that there must be a node $v \in S$ such that $|\Gamma(v) \cap V(P)| > 3$. By Lemma 4, this implies that P is not a shortest path, which is a contradiction. ◀

With these lemmas in place, we can now turn to the problem of bounding the diameter of a connected component K of $\hat{H}$. We first bound the number of centers established in Step 1.

▶ **Lemma 6.** *It holds $|C| = \tilde{O}(n^{1-\delta})$, w.h.p.*

Proof. Let X_u be a binary random variable such that $X_u = 1$ if and only if $u \in C$. We have $\mathrm{E}[X_u] = (c \log n)/n^\delta$. By definition it holds $|C| = \sum_{u \in V} X_u$. The linearity of expectation implies $\mathrm{E}[\,|C|\,] = \sum_{u \in V} E[X_u] = cn^{1-\delta} \log n$. Since $|C|$ is a sum of independent binary random variables we can apply Chernoff bounds (see, e.g., [20]) to get $\Pr[\,|C| \geq 2cn^{1-\delta} \log n\,] \leq \exp(-cn^{1-\delta} \log n/3)$. The lemma follows by choosing c sufficiently large. ◀

The next two lemmas show that the set of centers in K forms a dominating set of K.

▶ **Lemma 7.** *After Step 2 each high-degree node is adjacent to a center in $\hat{H}$, w.h.p.*

Proof. Consider a node $u \in V_{\text{high}}$. Let S be the set of the n^δ neighbors of u with lowest identifier. Each node in S is a center with probability p. Hence, the probability that no node in S is a center is $(1-p)^{|S|} = \left(1 - (c \log n)/n^\delta\right)^{n^\delta} \leq \exp(-c \log n)$. The lemma follows by applying the union bound over all nodes and choosing the constant c sufficiently large. ◀

▶ **Lemma 8.** *Let K be a connected component of $\hat{H}$ before any iteration of the loop in Step 4 or after the final iteration. The set of centers in K is a dominating set of K, w.h.p.*

Proof. Recall that $V(K) \subseteq V_{\text{high}} \cup C$ by definition. Hence, each node $u \in V(K)$ is a center or has high degree. If u is a center, there is nothing to show. If u is not a center, it must be of high degree. According to Lemma 7, u is connected to a center v in $\hat{H}$. This implies that $v \in V(K)$ and $\{u, v\} \in E(K)$. ◀

By combining the statements of Lemmas 5, 6 and 8, we get the following lemma.

▶ **Lemma 9.** *Let $K_1, \ldots, K_r$ be the connected components of $\hat{H}$ before any iteration of the loop in Step 4 or after the final iteration. It holds $\sum_{i=1}^{r} diam(K_i) = \tilde{O}(n^{1-\delta})$, w.h.p.*

Proof. Let $C(K_i)$ be the set of centers in K_i. According to Lemma 8, $C(K_i)$ is a dominating set of K_i. Therefore, Lemma 5 implies $\operatorname{diam}(K_i) < 3|C(K_i)|$. This implies $\sum_{i=1}^{r} \operatorname{diam}(K_i) < 3 \sum_{i=1}^{r} |C(K_i)| = 3|C| = \tilde{O}(n^{1-\delta})$, where the last equality holds according to Lemma 6. ◀

The following simple corollary gives us the desired bound on the diameter of a connected component K of $\hat{H}$.

▶ **Corollary 10.** *Let K be a connected component of $\hat{H}$ before any iteration of the loop in Step 4 or after the final iteration. It holds $diam(K) = \tilde{O}(n^{1-\delta})$, w.h.p.*

On the basis of Corollary 10, we can bound the value of T, the waiting time used in Step 4c: Consider an iteration of the loop in Step 4. For each connected component K the leader election in Step 4a can be achieved in $\tilde{O}(n^{1-\delta})$ rounds according to Theorem 3. The algorithm FindAny in Step 4b requires $\tilde{O}(n^{1-\delta})$ rounds according to Theorem 1. Therefore, we can choose T such that $T = \tilde{O}(n^{1-\delta})$.

Our next objective is to show that the computed subgraph H is an additive $\tilde{O}(n^{1-\delta})$-danner. To this end, we first take a closer look at the connected components of $\hat{H}$ after the algorithm terminates.

▶ **Lemma 11.** *After Algorithm 1 terminates, the set of connected components of $\hat{H}$ equals the set of connected components of $\hat{G}$.*

Proof. Consider a connected component $K_{\hat{G}}$ of $\hat{G}$. We show by induction that after iteration i of the loop in Step 4, each connected component of $\hat{H}[V(K_{\hat{G}})]$ has size at least $\min\{2^i, |V(K_{\hat{G}})|\}$. Since $|V(K_{\hat{G}})| \leq n$ and the loop runs for $\log n$ iterations, this implies that after the algorithm terminates, only one connected components remains in $\hat{H}[V(K_{\hat{G}})]$.

The statement clearly holds before the first iteration of the loop, i.e., for $i = 0$. Suppose that the statement holds for iteration $i \geq 0$. We show that it also holds for iteration $i+1$. If there is only one connected component at the beginning of iteration $i+1$ then that connected component must equal $K_{\hat{G}}$ so the statement holds. If there is more than one connected component at the beginning of iteration $i+1$ then by the induction hypothesis each connected component has size at least 2^i. Each connected component finds an edge leading to another connected component in Step 4b and thereby merges with at least one other connected component. The size of the newly formed component is at least $\min\{2^{i+1}, |K_{\hat{G}}|\}$. ◀

We are now ready to show that H is an additive $\tilde{O}(n^{1-\delta})$-danner.

▶ **Lemma 12.** *Algorithm 1 computes an additive $\tilde{O}(n^{1-\delta})$-danner H of G, w.h.p.*

Proof. Let $P_G = (u_0, \ldots, u_\ell)$ be a shortest path in G. We construct a path P_H from u_0 to u_ℓ in H such that $|P_H| \leq |P_G| + \tilde{O}(n^{1-\delta})$. Some of the edges in P_G might be missing in H. Let $\{u_i, u_{i+1}\}$ be such an edge. Observe that if u_i or u_{i+1} has low degree then the edge $\{u_i, u_{i+1}\}$ is contained in H since a low degree node adds all of its incident edges to H in Step 2. Hence, u_i and u_{i+1} must have high degree. Since the nodes share an edge in G, they lie in the same connected component of $\hat{G}$. According to Lemma 11 this means that the nodes also lie in the same connected component of $\hat{H}$. Therefore, there is a path in $\hat{H}$ between u_i and u_{i+1}. We construct P_H from P_G by replacing each edge $\{u_i, u_{i+1}\}$ that is missing in H by a shortest path from u_i to u_{i+1} in $\hat{H}$.

While P_H is a valid path from u_0 to u_ℓ in H, its length does not necessarily adhere to the required bound. To decrease the length of P_H, we do the following for each connected component K of $\hat{H}$: If P_H contains at most one node from K, we proceed to the next connected component. Otherwise, let v be the first node in P_H that lies in K and let w be the last node in P_H that lies in K. We replace the subpath from v to w in P_H by a shortest path from v to w in $\hat{H}$. After iteratively applying this modification for each connected component, the path P_H enters and leaves each connected component of $\hat{H}$ at most once and within each connected component P_H only follows shortest paths. Hence, according to Lemma 9, the number of edges in P_H passing through $\hat{H}$ is bounded by $\tilde{O}(n^{1-\delta})$. The remaining edges in P_H stem from P_G, so their number is bounded by $|P_G|$. In summary, we have $|P_H| \leq |P_G| + \tilde{O}(n^{1-\delta})$. ◀

To complete our investigation we analyze the time and message complexity of Algorithm 1 and bound the number of edges in the resulting danner H.

▶ **Lemma 13.** *The running time of Algorithm 1 is $\tilde{O}(n^{1-\delta})$ and the number of messages sent by the algorithm is $\tilde{O}(\min\{m, n^{1+\delta}\})$. After the algorithm terminates it holds $|E(H)| = \tilde{O}(\min\{m, n^{1+\delta}\})$.*

Proof. We begin with the running time. The first three steps of the algorithm can be executed in a single round. The loop in Step 4 runs for $\log n$ iterations, each of which takes $T = \tilde{O}(n^{1-\delta})$ rounds.

Next we bound the number of edges in the danner. Steps 1 and 3 do not add any edges to H. Step 2 adds $\tilde{O}(\min\{m, n^{1+\delta}\})$ edges to H. The loop in Step 4 runs for $\log n$ iterations, and in every iteration each connected component of $\hat{H}$ adds at most one edge to H. Since the number of connected components is at most n at all times, the total number of edges added in Step 4 is $\tilde{O}(n)$.

Finally, we turn to the message complexity of the algorithm. In Step 1 the nodes send no messages. The number of messages sent in Step 2 is $\tilde{O}(\min\{m, n^{1+\delta}\})$. In Step 3 each low-degree node sends a message to each of its neighbors. By definition a low-degree node has at most n^δ neighbors and there are at most n low-degree nodes. Therefore, at most $\tilde{O}(\min\{m, n^{1+\delta}\})$ messages are sent in this step. Each iteration of the loop in Step 4 operates on a subgraph $\hat{H}$ of the final danner H. Consider a connected component K of $\hat{H}$. Both the leader election in Step 4a and the algorithm FindAny in Step 4b use $\tilde{O}(|E(K)|)$ messages according to Theorems 3 and 1, respectively. Hence, the overall number of messages used in any iteration is $\tilde{O}(|E(\hat{H})|)$ which is bounded by $\tilde{O}(|E(H)|) = \tilde{O}(\min\{m, n^{1+\delta}\})$. ◀

Finally, we treat the special case $\delta = 1$. In this case we do not use Algorithm 1 but instead let each node add all its incident edges to H such that $H = G$. Combining the statements of the previous two lemmas together with the special case of $\delta = 1$ yields the following theorem.

▶ **Theorem 14.** *There is an algorithm that for a connected graph G and any $\delta \in [0, 1]$ computes an additive $\tilde{O}(n^{1-\delta})$-danner H consisting of $\tilde{O}(\min\{m, n^{1+\delta}\})$ edges, w.h.p. The algorithm takes $\tilde{O}(n^{1-\delta})$ rounds and requires $\tilde{O}(\min\{m, n^{1+\delta}\})$ messages.*

4 Applications

In this section we demonstrate that the danner construction presented in Section 3 can be used to establish trade-off results for many fundamental problems in distributed computing.

4.1 Broadcast, Leader Election, and Spanning Tree

On the basis of the danner construction presented in Section 3 it is easy to obtain a set of trade-off results for broadcast, leader election, and spanning tree construction. The number of messages required for a broadcast can be limited by first computing a danner and then broadcasting along the danner. For leader election we can run the algorithm of Kutten et al. [18] mentioned in Theorem 3 on the computed danner. Finally, for spanning tree construction we can elect a leader which then performs a distributed breadth-first search on the danner to construct the spanning tree. We have the following theorem.

▶ **Theorem 15.** *There are algorithms that for any connected graph G and any $\delta \in [0, 1]$ solve the following problems in $\tilde{O}(D + n^{1-\delta})$ rounds while using $\tilde{O}(\min\{m, n^{1+\delta}\})$ messages, w.h.p.: broadcast, leader election, and spanning tree.*

4.2 Minimum Spanning Tree and Connectivity

In this section we assume that we are given a *weighted* connected graph G with edge weights from $\{1, \dots, \mathrm{poly}(n)\}$. Without loss of generality we assume that the edge weights are distinct such that the MST is unique. We present a three step algorithm for computing the MST.

Step 1. We compute a spanning tree of G using the algorithm described in Section 4.1 while ignoring the edge weights. Recall that the algorithm computes the spanning tree by having a leader node initiate a distributed breadth-first search along a danner of diameter $\tilde{O}(D + n^{1-\delta})$. Therefore, the algorithm supplies us with a *rooted* spanning tree T of depth $\tilde{O}(D + n^{1-\delta})$. We aggregate the number of edges m in G using a convergecast along T. If $m \leq n^{1+\delta}$, we execute the singularly optimal algorithm of Pandurangan et al. [25] on the original graph G to compute the MST, which takes $\tilde{O}(D + \sqrt{n})$ rounds and requires $\tilde{O}(m)$ messages. Otherwise, we proceed with the following steps.

Step 2. We execute the so-called Controlled-GHS procedure on G as described in [25]. This procedure is a modified version of the classical Gallager-Humblet-Spira (GHS) algorithm for distributed MST [9]. It constructs a set of *MST fragments* (i.e., connected subgraphs of the MST). However, in contrast to the original GHS, Controlled-GHS limits the diameter of the fragments by controlling the way in which fragments are merged. By running Controlled-GHS for $\lceil (1-\delta)\log n \rceil$ iterations we get a spanning forest consisting of at most n^δ MST-fragments, each having diameter $O(n^{1-\delta})$ in $\tilde{O}(n^{1-\delta})$ rounds. The Controlled-GHS described in [25] requires $\tilde{O}(m)$ messages. However, we can reduce the number of messages to $\tilde{O}(n)$ without increasing the running time by modifying the Controlled-GHS procedure to use the algorithm FindMin described in Theorem 2 to find the lightest outgoing edge of a fragment.

Step 3. Our goal in the final step of the algorithm is to merge the remaining n^δ MST fragments quickly. This step executes the same procedure for $\log n$ iterations. Each iteration reduces the number of fragments by at least a factor of two so that in the end only a single fragment remains, which is the MST.

We use a modified version of the algorithm TestOut that only communicates along the spanning tree T computed in Step 1 (i.e., it ignores the structure of the fragments) and that operates on all remaining fragments in parallel. Recall that the original TestOut algorithm for a single connected component consists of a broadcast-and-echo in which a leader broadcasts a random hash function and the nodes use an echo (or convergecast) to aggregate the parity of a set of bits. We can parallelize this behavior over all fragments as follows: Let v_T be the root of T. First, v_T broadcasts a random hash function through T. The same hash function is used for all fragments. Each node u uses the hash function to compute its individual bit as before and prepares a message consisting of the identifier of the fragment containing u and the bit of u. These messages are then aggregated up the tree in a pipelined fashion: In each round, a node sends the message with the lowest fragment identifier to its parent. Whenever a node holds multiple messages with the same fragment identifier, it combines them into a single message consisting of the same fragment identifier and the combined parity of the bits of the respective messages. Since T has depth $\tilde{O}(D+n^{1-\delta})$ and there are at most n^δ different fragment identifiers, v_T learns the aggregated parity of the bits in each individual fragment after $\tilde{O}(D+n^{1-\delta}+n^\delta)$ rounds, which completes the parallelized execution of TestOut.

As explained in Section 2.3, a polylogarithmic number of executions of TestOut in combination with binary search can be used to identify the lightest outgoing edge of a fragment. The ranges for the binary search for each fragment can be broadcast by v_T in a pipelined fashion and the TestOut procedure can be executed in parallel for all fragments as described above. Thereby, v_T can learn the identifier of a lightest outgoing edge for each fragment in parallel. To merge the fragments, v_T does the following: First, it learns the fragment identifiers of the nodes at the end of each outgoing edge. It then locally computes the changes in the fragment identifiers that follow from the merges. Finally, it broadcasts these changes along with the identifiers of the leaving edges to merge the fragments. This completes one iteration of the procedure.

Overall, the operations of the final step can be achieved a using polylogarithmic number of pipelined broadcast-and-echo operations. Therefore, the running time of this step is $\tilde{O}(D+n^{1-\delta}+n^\delta)$ rounds. In each pipelined broadcast-and-echo each node sends at most n^δ messages, so the overall number of messages is $\tilde{O}(n^{1+\delta})$. This gives us the following theorem.

▶ **Theorem 16.** *There is an algorithm that for any connected graph G with edge weights from $\{1,\dots,\mathrm{poly}(n)\}$ and any $\delta \in [0, 0.5]$ computes an MST of G in $\tilde{O}(D+n^{1-\delta})$ rounds while using $\tilde{O}(\min\{m, n^{1+\delta}\})$ messages, w.h.p.*

For $\delta = 0.5$ we get an algorithm with optimal running time up to polylogarithmic factors.

▶ **Corollary 17.** *There is an algorithm that for any connected graph G with edge weights from $\{1, \ldots, \text{poly}(n)\}$ computes an MST of G in $\tilde{O}(D + \sqrt{n})$ rounds while using $\tilde{O}(\min\{m, n^{3/2}\})$ messages, w.h.p.*

Using this result on MST, it is not hard to devise an algorithm that computes the connected components of a subgraph H of G (and thus also test connectivity): We assign the weight 0 to each edge in $E(H)$ and the weight 1 to each edge in $E(G) \setminus E(H)$. We then run a modified version of the above MST algorithm in which a fragment stops participating as soon as it discovers that its lightest outgoing edge has weight 1. Thereby, fragments only merge along edges in H. Once the algorithm terminates, any two nodes in the same connected component of H have the same fragment identifier while any two nodes in distinct connected components have distinct fragment identifiers.

▶ **Corollary 18.** *There is an algorithm that for any graph G, any subgraph H of G, and any $\delta \in [0, 0.5]$ identifies the connected components of H in $\tilde{O}(D + n^{1-\delta})$ rounds while using $\tilde{O}(\min\{m, n^{1+\delta}\})$ messages, w.h.p.*

4.3 $O(\log n)$-Approximate Minimum Cut

We describe an algorithm that finds an $O(\log n)$-approximation to the edge connectivity of the graph (i.e., the minimum cut value).

▶ **Theorem 19.** *There is a distributed algorithm for finding an $O(\log n)$-approximation to the edge connectivity of the graph (i.e., the minimum cut value) that uses $\tilde{O}(\min\{m, n^{1+\delta}\})$ messages and runs in $\tilde{O}(D + n^{1-\delta})$ rounds for any $\delta \in [0, 0.5]$, w.h.p.*

The main idea behind the algorithm is based on the following sampling theorem.

▶ **Theorem 20** ([14, 11]). *Consider an arbitrary unweighted multigraph[10] $G = (V, E)$ with edge connectivity λ and choose subset $S \subseteq E$ by including each edge $e \in E$ in set S independently with probability p. If $p \geq c \log n/\lambda$, for a sufficiently large (but fixed) constant c, then the sampled subgraph $G' = (V, S)$ is connected, w.h.p.*

We next sketch the distributed algorithm that is claimed in Theorem 19. The distributed algorithm implements the above sampling theorem which provides a simple approach for finding an $O(\log n)$-approximation of the edge connectivity of G by sampling subgraphs with exponentially growing sampling probabilities (e.g., start with an estimate of $\lambda = m$, the total number of (multi-)edges, and keep decreasing the estimate by a factor of 2) and checking the connectivity of each sampled subgraph. We take the first value of the estimate where the sampled graph is connected as the $O(\log n)$-approximate value. This algorithm can be easily implemented in the KT_0 distributed model using $\tilde{O}(m)$ messages and $\tilde{O}(D + \sqrt{n})$ rounds by using the singularly optimal MST algorithm of [25] or [7] (which can be directly used to test connectivity – cf. Section 4.2). However, implementing the algorithm in the KT_1 model in the prescribed time and message bounds of Theorem 19 requires some care, because of implementing the sampling step; each endpoint has to agree on the sampled edge without actually communicating through that edge.

[10] Note that a weighted graph with polynomially large edge weights can be represented as an unweighted graph with polynomial number of multiedges.

The sampling step can be accomplished by sampling with a *strongly $O(\log n)$-universal hash function.* Such a hash function can be created by using only $O(\log n)$ independent shared random bits (see, e.g., [3]). The main insight is that a danner allows sharing of random bits efficiently. This can be done by constructing a danner[11] and letting the leader (which can be elected using the danner – cf. Section 4.1) generate $O(\log n)$ independent random bits and broadcast them to all the nodes via the danner. The nodes can then construct and use the hash function to sample the edges. However, since the hash function is only $O(\log n)$-universal it is only $O(\log n)$-wise independent. But still one can show that the guarantees of Theorem 20 hold if the edges are sampled by a $O(\log n)$-wise independent hash function. This can be seen by using Karger's proof [14] and checking that Chernoff bounds for $O(\log n)$-wise independent random variables (as used in [28]) are (almost) as good as that as (fully) independent random variables. Hence edge sampling can be accomplished using time $\tilde{O}(D + n^{1-\delta})$ (the diameter of the danner) and messages $\tilde{O}(\min\{m, n^{1+\delta})$ (number of edges of the danner). Once the sampling step is done, checking connectivity can be done by using the algorithm of Section 4.2.

4.4 Algorithms for Graph Verification Problems

It is well known that graph connectivity is an important building block for several graph verification problems (see, e.g., [5]). Thus, using the connectivity algorithm of Section 4.2 as a subroutine, we can show that the problems stated in the theorem below (these are formally defined, e.g., in Section 2.4 of [5]) can be solved in the KT$_1$ model (see, e.g., [5, 24]).

▶ **Theorem 21.** *There exist distributed algorithms in the KT_1 model that solve the following verification problems in $\tilde{O}(\min\{m, n^{1+\delta}\})$ messages and $\tilde{O}(D + n^{1-\delta})$ rounds, w.h.p, for any $\delta \in [0, 0.5]$: spanning connected subgraph, cycle containment, e-cycle containment, cut, s-t connectivity, edge on all paths, s-t cut, bipartiteness.*

5 Conclusion

This work is a step towards understanding time-message trade-offs for distributed algorithms in the KT$_1$ model. Using our danner construction, we obtained algorithms that exhibit time-message trade-offs across the spectrum by choosing the parameter δ as desired. There are many key open questions raised by our work. First, it is not clear whether one can do better than the algorithms we obtained here for various fundamental problems in the KT$_1$ model. In particular, it would be interesting to know whether there are singularly optimal algorithms in the KT$_1$ model – e.g., for leader election, can we show an $\tilde{O}(n)$ messages algorithm that runs in $\tilde{O}(D)$ (these are respective lower bounds for messages and time in KT$_1$); and, for MST, can we show an $\tilde{O}(n)$ messages algorithm that runs in $\tilde{O}(D + \sqrt{n})$. A related question is whether one can construct an (α, β)-danner with $\tilde{O}(n)$ edges and $\alpha, \beta = \tilde{O}(1)$ in a distributed manner using $\tilde{O}(n)$ messages and in $\tilde{O}(D)$ time. Such a construction could be used to obtain singularly optimal algorithms for other problems. Finally, our danner construction is randomized; a deterministic construction with similar guarantees will yield deterministic algorithms.

[11] Note that the danner of a multigraph is constructed by treating multi-edges as a single edge.

References

1 Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM J. Comput.*, 28(4):1167–1181, 1999. doi:10.1137/S0097539796303421.

2 Baruch Awerbuch, Oded Goldreich, David Peleg, and Ronen Vainish. A trade-off between information and communication in broadcast protocols. *J. ACM*, 37(2):238–256, 1990.

3 Larry Carter and Mark N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.

4 Yongwook Choi, Gopal Pandurangan, Maleq Khan, and V. S. Anil Kumar. Energy-optimal distributed algorithms for minimum spanning trees. *IEEE Journal on Selected Areas in Communications*, 27(7):1297–1304, 2009.

5 Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM J. Comput.*, 41(5):1235–1265, 2012.

6 Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM J. Comput.*, 29(5):1740–1759, 2000. doi:10.1137/S0097539797327908.

7 Michael Elkin. A simple deterministic distributed MST algorithm, with near-optimal time and message complexities. In *Proceedings of the 2017 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 157–163, 2017.

8 Uriel Feige, David Peleg, Prabhakar Raghavan, and Eli Upfal. Randomized broadcast in networks. *Random Struct. Algorithms*, 1(4):447–460, 1990. doi:10.1002/rsa.3240010406.

9 Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983. doi:10.1145/357195.357200.

10 J. Garay, S. Kutten, and D. Peleg. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM Journal on Computing*, 27(1):302–316, February 1998.

11 Mohsen Ghaffari and Fabian Kuhn. Distributed minimum cut approximation. In *Distributed Computing - 27th International Symposium, DISC 2013, Jerusalem, Israel, October 14-18, 2013. Proceedings*, pages 1–15, 2013.

12 Bernhard Haeupler, D. Ellis Hershkowitz, and David Wajc. Round- and message-optimal distributed graph algorithms. In *PODC*, 2018.

13 James W. Hegeman, Gopal Pandurangan, Sriram V. Pemmaraju, Vivek B. Sardeshmukh, and Michele Scquizzato. Toward optimal bounds in the congested clique: graph connectivity and MST. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 91–100, 2015.

14 David R. Karger. Random sampling in cut, flow, and network design problems. *Math. Oper. Res.*, 24(2):383–413, 1999.

15 Maleq Khan, Gopal Pandurangan, and V. S. Anil Kumar. Distributed algorithms for constructing approximate minimum spanning trees in wireless sensor networks. *IEEE Trans. Parallel Distrib. Syst.*, 20(1):124–139, 2009.

16 Valerie King, Shay Kutten, and Mikkel Thorup. Construction and impromptu repair of an MST in a distributed network with o(m) communication. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 71–80, 2015. doi:10.1145/2767386.2767405.

17 Hartmut Klauck, Danupon Nanongkai, Gopal Pandurangan, and Peter Robinson. Distributed computation of large-scale graph problems. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 391–410, 2015.

18 Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. On the complexity of universal leader election. *J. ACM*, 62(1), 2015.

19 Ali Mashreghi and Valerie King. Time-communication trade-offs for minimum spanning tree construction. In *Proceedings of the 18th International Conference on Distributed Computing and Networking, Hyderabad, India, January 5-7, 2017*, page 8, 2017. URL: `http://dl.acm.org/citation.cfm?id=3007775`.

20 Michael Mitzenmacher and Eli Upfal. *Probability and computing - randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

21 Danupon Nanongkai and Hsin-Hao Su. Almost-tight distributed minimum cut algorithms. In *Distributed Computing - 28th International Symposium, DISC 2014, Austin, TX, USA, October 12-15, 2014. Proceedings*, pages 439–453, 2014.

22 Gopal Pandurangan. Distributed network algorithms. Draft, 2018. URL: `https://sites.google.com/site/gopalpandurangan/dna`.

23 Gopal Pandurangan and Maleq Khan. Theory of communication networks. In *Algorithms and Theory of Computation Handbook*. CRC Press, 2009.

24 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. Fast distributed algorithms for connectivity and MST in large graphs. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*, pages 429–438, 2016.

25 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. A time- and message-optimal distributed algorithm for minimum spanning trees. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 743–756, 2017. `doi:10.1145/3055399.3055449`.

26 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. On the distributed complexity of large-scale graph computations. In *Proceedings of the 30th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA*, 2018.

27 D. Peleg. *Distributed Computing: A Locality Sensitive Approach*. SIAM, 2000.

28 Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-hoeffding bounds for applications with limited independence. *SIAM J. Discrete Math.*, 8(2):223–250, 1995.