Optimizing Round Complexity of Energy-Efficient Distributed MST Algorithms

Student Name: F.T. Cunningham
Supervisor Name: W.K. Moses Jr.
Submitted as part of the degree of BSc Computer Science to the
Board of Examiners in the Department of Computer Sciences, Durham University

Abstract—In this work, we focus on constant-diameter networks and present optimizations to the awake-optimal MST algorithm of Augustine, Moses Jr., and Pandurangan [1]. While preserving its asymptotic awake complexity, our optimizations reduce round complexity in practice. We introduce two key modifications: adaptive transmission scheduling, which shortens transmission schedules based on the maximum fragment depth in each phase, and procedure-specific scheduling, which eliminates unnecessary awake rounds depending on the fragment procedure being performed. Although these optimizations do not improve the worst-case bounds of the original algorithm, they offer significant practical benefits, achieving lower empirical round complexity with only a modest constant-factor increase in awake complexity. To evaluate our approach, we implement both the original and optimized algorithms in a custom simulation environment and apply empirical scaling analysis. We fit parametric models to observed data on both random and adversarial graph instances and assess these models through extrapolation beyond the instance sizes used for fitting. On random diameter 3 networks, the optimized algorithm exhibits a substantial improvement in empirical scaling of round complexity compared to the original, while incurring only a small constant-factor increase in awake complexity. Even on adversarial inputs designed to limit the effectiveness of our optimizations, improvements in round complexity persist. These results demonstrate that practical reductions in round complexity are achievable in small constant-diameter networks without significantly compromising awake complexity.

Index Terms—Distributed networks, Network problems, Performance evaluation of algorithms and systems

1 Introduction

The distributed minimum spanning tree (MST) problem is a fundamental problem in distributed computing that has been studied extensively for over four decades. We're given a distributed network modelled as a weighted graph $G = (V, E, w)$, and the goal is to identify a subset of the edges that connects all nodes without forming any cycles, while minimising the total weight of the selected edges. Computation proceeds in a distributed fashion, nodes can exchange messages with their immediate neighbours, perform local computations and upon termination, each node knows exactly which of its incident edges belong to the MST.

An MST is an important primitive in many network applications such as leader election and efficient broadcast [2], [3]. In the case of efficient broadcast, any node can send a message to all others nodes by forwarding it along the edges of the MST. When edge weights represent costs such as latency or energy, the MST minimizes the total communication cost by minimizing the sum of edge weights used in the broadcast.

1.1 Awake Complexity

Traditionally, the distributed MST problem has been studied through the lenses of round (time) and message complexity. A long line of research has established a tight optimal bound up to logarithmic factors of $\tilde{\Theta}(D + \sqrt{n})$[1] for the round complexity and $\tilde{\Theta}(m)$ for the message complexity [4], [5], [6]. Recently, several works have explored the awake complexity of several important distributed algorithms [1], [7], [8] including MST and maximal independent set (MIS). This metric is used to capture the energy efficiency of a distributed algorithm in a model of distributed computing called the sleeping model, where nodes can be in one of two states, awake or sleeping (and saving energy). The energy usage of a distributed algorithm is an important consideration particularly in energy-constrained networks such as battery-powered wireless sensor networks. In these settings, a significant portion of a node's energy is consumed by sending and receiving messages, and experimental studies have shown that the energy cost of even passively listening for messages is only marginally lower than the cost of active communication [7]. While devices remain in this awake state, they consume substantial energy, largely due to the energy cost of keeping their communication devices continuously powered on. Placing a node in a sleeping state, in which its communication devices are completely turned off, allows it to consume negligible amounts of energy. In this state, however, a node cannot send or receive messages nor can it perform local computations. This poses an interesting problem, whether we're able to design distributed algorithms in such a way that nodes are only awake for a small number of rounds, thereby significantly reducing their overall energy consumption. Communication between neighbouring nodes is only possible if both are awake in the same round, so a key challenge in designing algorithms that leverage the sleeping state is determining when each node must be awake to ensure that necessary communication can occur.

1. The notation $\tilde{\Theta}(\cdot)$ hides polylogarithmic factors in n.

The work of Augustine, Moses Jr., and Pandurangan showed that constructing an MST in the sleeping model can be accomplished in $O(\log n)$ awake rounds and $O(n \log n)$ round complexity. They also show a lower bound of $\Omega(\log n)$ on the awake complexity of constructing an MST, so their algorithm is awake optimal. By their trade-off lower bound of $\tilde{\Omega}(n)$ on the product of round complexity and awake complexity their randomized algorithm also achieves the best possible round complexity for an awake-optimal algorithm (up to logarithmic factors). However, this trade-off lower bound is only shown for graphs having diameter $\tilde{\Omega}(\sqrt{n})$ posing the question whether a round complexity $o(n \log n)$ can be achieved by an awake optimal MST algorithm for graphs of small, constant diameter. Existing upper bounds on the round complexity of constructing an MST have settled this for graphs of diameter 1 and 2 [9], [10], however, it remains open whether similar results can be achieved for graphs of diameter 3 and 4. Consequently, in this work, we study the round complexity of awake-optimal distributed MST algorithms for graphs of diameter 3.

1.2 Model and Complexity Measures

1.2.1 Distributed Computing Model

We are given a distributed network modelled as an arbitrary undirected, connected, weighted graph $G(V, E, w)$, where the set of nodes V represents the processors, and the set of edges E correspond to the communication channels through which these processors can exchange messages. Let $n = |V|$ denote the number of processors in the network. We define a weight function $w : E \to \mathbb{R}^+$, which assigns a positive real-valued weight to each edge which can model various parameters for example, bandwidth, delay and distance.

We define the diameter D of the network G to be the hop-diameter of G, namely, the maximum distance between any two vertices measured in hops.

We assume that communication is synchronous and occurs in lock-step in a series of discrete rounds. Nodes communicate directly (only) with their neighbours through the edges of the graph G. Within each round, a node can execute instantaneous local computations and exchange messages with its neighbours. Communication in the network is constrained by message size limitations. Specifically, we adopt the $CONGEST$ model, where each message exchanged between nodes is of size $O(\log n)$. For comparison, an alternative model known as the $LOCAL$ model places no restrictions on message size. We assume that nodes initially only have limited knowledge and only know about themselves and their incident edges.

All nodes run the same instance of the algorithm which is the distributed algorithm but depending on local information each node can exhibit its own behaviour.

Traditionally in distributed algorithm we seek to minimize round complexity. In the synchronous model that is the maximum number of rounds needed over all nodes before termination to a solution.

1.2.2 Sleeping Model

The sleeping model augments the traditional model outlined above by allowing nodes to be in one of two states: *awake* or *sleeping*. Nodes can transition into the sleeping state and specify in advance the round in which they will *wake up*, transitioning back into the awake state. While sleeping, a node cannot send or receive messages (messages sent to a sleeping node are dropped) nor can it perform any local computations. The awake state corresponds to the default state in the traditional model.

For a distributed algorithm in the sleeping model we introduce a second complexity metric, awake complexity. Let A_v be the number of rounds node v spends in the awake state before termination. Then we define the worst-case awake complexity to be $\max_{v \in V} A_v$, the worst-case number of awake rounds needed by a node to execute the algorithm.

While the goal in the sleeping model is typically to minimize the awake complexity, we would also like to minimize the traditional worst-case round complexity where both awake and sleeping rounds are counted.

1.3 Empirical Scaling Analysis

The time complexity of an algorithm has traditionally been studied by means of theoretical analysis. However, worst-case behaviour is often rarely observed in practise for many $\mathcal{NP}$-hard problems such as the Boolean satisfiability problem (SAT) and the travelling salesman problem (TSP). Therefore, the empirical analysis of the running time of these algorithms has received increasing interest [11], [12], [13], especially for heuristic algorithms. While this type of analysis has not previously been applied in the context of distributed algorithms, it provides a means of identifying the constants and lower-order terms that are omitted from asymptotic results. This, in turn, allows us to distinguish algorithms that perform better in practice, supporting more informed decisions in real-world applications.

Our empirical investigations follow the approach to empirical scaling analysis outlined by Hoos [14] and extended by Mu and Hoos [12]. This approach uses a statistical method known as bootstrapping to assess whether a given scaling model is statistically consistent with the observed behaviour of an algorithm. Scaling models are first fitted to observed running times using standard numerical optimization techniques, and then challenged by extrapolation to larger instance sizes beyond those used for fitting. Bootstrap CIs are derived both from the model predictions and from the observed data, allowing the quality of the model fit to be assessed in a statistically meaningful way.

1.4 Our Contributions

In light of the lower bound of $\Omega(\log n)$ on awake complexity for distributed MST algorithms, and the trade-off lower bound on the product of round and awake complexity that applies only to networks with diameter $\tilde{\Omega}(\sqrt{n})$, a natural question arises in the context of small, constant diameter networks. Motivated by both the theoretical bounds and the energy considerations discussed above, we ask:

Can we design a distributed MST algorithm for networks of diameter 3 with optimal awake complexity $O(\log n)$ and a round complexity $o(n \log n)$?

While we do not answer this question in the affirmative, we present promising results. Our main contribution is an

optimization to the awake-optimal randomized MST algorithm presented by Augustine, Moses Jr., and Pandurangan [1]. Based on the result of our empirical scaling analysis on randomly generated benchmark instances, the optimized algorithm demonstrates significantly better empirical scaling behaviour compared to the baseline. Specifically, the optimized algorithm exhibits scaling consistent with a polynomial model of the form $a \cdot n^b + c$, with $b \in [0.41433, 0.46214]$, whereas the original algorithm exhibits scaling consistent with a quasilinear model of the form $a \cdot n \cdot \log^b(n) + c$, with $b \in [0.87287, 0.99728]$. However, on a set of carefully constructed adversarial benchmark instances, our results indicate less significant improvements to the round complexity. Our empirical analysis suggests this is due to specific structures present in the input graphs that influence the performance of our optimizations. We also observe a small constant-factor increase in the awake complexity of our optimized variant by a factor of approximately 1.5 relative to the original algorithm. This constant-factor increase is marginal compared to the observed differences in the scaling models consistent with the round complexity of the optimized algorithm.

We were also able to derive an upper bound of $O(n \log n)$ on the worst-case round complexity of the optimized algorithm which shows our proposed optimizations do not lead to an asymptotic improvement over the original algorithm.

The essence of our proposed optimizations lies in leveraging the structural properties of constant-diameter graphs to reduce the round complexity without compromising the asymptotic awake complexity. In particular, we adapt the transmission schedules used in the original algorithm by shortening them based on the maximum fragment depth in each phase. This modification reduces the number of rounds required by fragment procedures.

As part of this work, we developed a Python environment for simulating distributed algorithms in the sleeping model, which includes, to the best of our knowledge, the first implementation of the awake-optimal randomized MST algorithm of [1] along with our optimized variant. Beyond supporting the experiments in this paper, the simulation environment was designed to be extensible, providing abstractions that make it easy to adapt and extend to other distributed algorithms, providing a foundation for future empirical investigations.

2 Related Work

2.1 Distributed Algorithms

Due to its fundamental importance, the distributed MST problem has been extensively studied for over four decades, beginning with the seminal work of Gallager, Humblet, and Spira (GHS) [15] in 1983, which presented a distributed MST algorithm with round (time) complexity $O(n \log n)$. Traditionally, the distributed MST problem has been studied through the lens of round complexity. A long line of research has focused on improving the round complexity of distributed MST algorithms, culminating in algorithms that are time-optimal up to polylogarithmic factors, running in $\tilde{O}(D + \sqrt{n})$ rounds [16], [17]. Subsequent work, focused on achieving simultaneous optimality with respect to both round and message complexity [4], [5], [6]. Pandurangan et al. [4] presented an algorithm that is singularly optimal, achieving $\tilde{O}(D + \sqrt{n})$ rounds and $\tilde{O}(m)$ messages, where m is the number of edges. Elkin [5] later extended this result presenting a deterministic algorithm which is also simultaneously optimal.

While there exist algorithms that are simultaneously optimal for the traditional metrics, round and message complexity, these metrics may not reflect the true cost of computation particularly in energy-constrained settings, where the quality of an algorithm is often better measured by the amount of energy it consumes. This has motivated recent work on the awake complexity of distributed algorithms, which was first introduced in the CONGEST model by Chatterjee, Gmyr, and Pandurangan in [7] where they studied the awake complexity of a different problem, Maximal Independent Set (MIS). They presented a randomized distributed algorithm for MIS that achieves expected $O(1)$ node-averaged awake complexity, while maintaining $O(\log n)$ worst-case awake complexity and $O(log^{3.41}(n))$ round complexity. In this paper our upper bounds focus on the worst-case awake complexity which is a stronger requirement than node-averaged awake complexity.

Subsequently, Barenboim and Maimon [8] studied the awake complexity of a broad class of global distributed problems in the sleeping model, including broadcast, leader election, and arbitrary spanning tree construction, but notably not MST. They presented a deterministic algorithm that achieves $O(\log n)$ awake complexity for these problems by constructing a structure called a Distributed Layered Tree (DLT). Using this structure broadcast and convergecast can be performed in $O(1)$ awake rounds. In the case of broadcast, for example, each node is awake only twice, once to receive the message from its parent and once to forward it to its children.

Moses Jr., and Pandurangan presented a randomized distributed algorithm for the MST problem in the CONGEST model that achieves a worst-case awake complexity of $O(\log n)$ and a round complexity of $O(n \log n)$ [1]. Their algorithm is awake-optimal, as they established a matching lower bound of $\Omega(\log n)$ on the awake complexity for any MST algorithm, even for randomized algorithms. Additionally, they prove a trade-off lower bound on the product of round and awake complexity of $\tilde{\Omega}(n)$ which applies to graphs with diameter $\tilde{\Omega}(\sqrt{n})$. This implies that for algorithms with optimal awake complexity, a round complexity of $O(n \log n)$ is essentially optimal up to logarithmic factors. They also show that the $O(\log n)$ awake complexity can be achieved deterministically, presenting a deterministic MST algorithm with $O(\log n)$ awake complexity but a increase in the round complexity to $O(n \log^5 n)$. A key technical contribution of their work was the construction of a spanning tree structure called the Labeled Distance Tree (LDT). An LDT is a rooted, oriented spanning tree where each node is labeled by its distance from the root, and each node knows the labels of its parent, its children, and the root. While this structure resembles the DLT of Barenboim and Maimon there are fundamental differences. Both structures, however, are designed to facilitate awake efficient procedures for broadcasting and upcasting information within their respective tree structures.

While the randomized MST algorithm of Augustine, Moses Jr., and Pandurangan [1] is awake-optimal and achieves the best possible round complexity by their trade-off lower bound on the product of round and awake complexity, this lower bound applies only to networks with diameter $\tilde{\Omega}(\sqrt{n})$. This leaves open what trade-off might be achievable in networks of small, constant diameter. For networks with diameter 1 (i.e., complete graphs) and diameter 2, the round complexity of constructing an MST is already well understood. Lotker et al. [18] gave an $O(\log n)$ round algorithm for diameter 2 networks, and later showed that an $O(\log \log n)$ round algorithm for networks of diameter 2 [10] confirming the trade-off does not apply to these networks. For diameter 3 and diameter 4 networks, however, existing results give lower bounds on round complexity of constructing an MST in the $CONGEST$ model of $\Omega(\sqrt[4]{n}/\sqrt{\log n})$ and $\Omega(\sqrt[3]{n}/\log n)$, respectively [10], that also hold for randomized algorithms. This leaves open whether it is possible to simultaneously achieve optimal awake complexity of $O(\log n)$ and a round complexity of $o(n \log n)$ in graphs with with diameter 3 and 4.

The term energy complexity is often used interchangeably with awake complexity, particularly in earlier work that studied energy complexity in radio network models. In these models, nodes operate under stricter communication constraints, during each round, a node may either transmit or listen, but not both. Moreover, nodes only successful receive a message when exactly one neighbour transmits otherwise, a collision occurs. Chang et al. [19] presented foundational results on the energy complexity of the Broadcast problem in such radio models. They established tight lower bounds for several variants of the radio model, including the "Local" model, in which collisions are ignored and nodes can transmit simultaneously. Importantly, their lower bound of $\Omega(\log n)$ on the energy complexity of Broadcast in the Local radio model applies directly to the SLEEPING-LOCAL and SLEEPING-CONGEST models. These models generalize the Local radio model by allowing nodes to send and receive in the same round.

2.2 Empirical Scaling Analysis

There is significant interest in the empirical scaling analysis of high-performance heuristic algorithms where theoretical analysis remains intractable [12], [20]. Early work by Parkes and Walser [21] studied the empirical scaling of several local search algorithms for the Boolean satisfiability problem (SAT) on a class of randomized benchmark problems, Random 3SAT. Their analysis involved fitting parametric scaling models to observed average running time data using the Marquardt-Levenberg algorithm and subsequent work adopted similar approaches [22], [23]. Gent et al. [23] extend this approach by challenging their scaling models by interpolation and extrapolation to instances different to those used for fitting the models. However, they did not formalize any methods for assessing the fitted models based on these results.

Goldsmith et al. [24] presented a tool, Trend Profile, for constructing models of empirical computational complexity. Their tool fits linear or powerlaw models to the execution frequency of user specified basic blocks of a program to describe the asymptotic behaviour of a program. This tool however, does not provide ways of assessing the quality of a given model beyond examining the residuals, that is the difference between the observed and predicted counts of basic blocks. This method of counting basic blocks is analogous to our approach of counting synchronous rounds in distributed algorithms, as both provide measures of performance that are independent of the hardware on which the benchmarks were run.

The work of Hoos [14] presented a novel approach for assessing the fit of a scaling model. Similar to the work of Gent et al. [23], they used extrapolation of the predictions of a scaling model beyond the instance sizes used to fit the model to assess its consistency with observed data. However, they propose using bootstrap analysis to derive CIs for the predictions generated by a given scaling model and use these to determine the degree to which observed data are statistically supported by the model. They demonstrate their method on the scaling of several high-performance algorithms for the Travelling salesman problem (TSP) which are shown to exhibit scaling consistent with polynomial and exponential models. Using this approach they were able to identify models that performed well subject to extreme extrapolation challenges. In their work they indicate that similar results could potentially be achieved for algorithms whose scaling is characterized by slow-growing functions as well as to other performance measures beyond the running time of an algorithm, two key aspects that also underpin our approach.

Subsequently, Mu and Hoos [25] introduced an automated tool, Empirical Scaling Analyser (ESA), for performing the empirical scaling analysis approach proposed in [14]. They later applied this tool to further study the empirical time complexity of SAT algorithms on the Random 3-SAT [12] benchmark problems. They extended the original approach in two key ways: first, by deriving bootstrap CIs for both observed performance data and model predictions, and second, by introducing a method to test whether the observed performance of one algorithm is consistent with the scaling model of another. These extensions were implemented as part of ESA v1.1, which is the version of the tool we used in this work.

3 Methodology

In this section, we present our optimizations to the awake-optimal randomized MST algorithm proposed in [1]. Since our work builds upon this algorithm, we begin with a brief overview of it. We then introduce the modifications that constitute our optimized variant. For clarity, we refer to the original algorithm as the baseline algorithm and to our proposed version as the optimized algorithm throughout the remainder of this paper.

We also describe the methodology used to empirically evaluate and compare both algorithms. In particular, we adopt a statistical approach proposed by Hoos [14] to derive empirical scaling models from observed performance data and to assess the accuracy of these models in characterizing how the round and awake complexity of each algorithm scales with the size of the input graph.

3.1 Baseline Algorithm Overview

The awake-optimal randomized MST algorithm proposed in [1] forms the foundation of our work. The high level structure of the algorithm is similar to that of the classical GHS algorithm, it operates in phases, each consisting of two main steps: (i) fragments identify their minimum-weight outgoing edge (MOE), and (ii) fragments merge across these MOE's to form larger fragments. Here, a fragment refers to a subtree of the MST and an outgoing edge of a fragment is an edge with one of its adjacent nodes in the fragment and the other in a different fragment. Initially, each node starts as a singleton fragment, and over the course of $O(\log n)$ phases, fragments merge until only one remains, which corresponds to the MST of the original graph.

To find the MOE of a fragment, a node designated as the root node of the fragment initiates a broadcast along the fragment's edges. Each node identifies its local minimum weight outgoing edge incident on it. Upcasting is then used to propagate the MOE back to the root: starting from the leaves, each node forwards the minimum of its own and its children's local MOE's up the tree.

Once the root node knows the fragment MOE, it broadcasts this information to the nodes in the fragment. In the next step the fragment attempts to merge with the neighbouring fragment adjacent to the MOE. Merging will only proceed subject to a merging criterion. To determine if the MOE is valid, each fragment flips a coin to label itself as either "head" or "tail". A merge is only permitted if the MOE connects a tail fragment to a head fragment. In this case, the head fragment retains its structure, while the tail fragment is absorbed into it and internally reorients itself to form a subtree, connected to the head fragment via the MOE.

The algorithm is able to perform the steps of a single phase in a constant number of awake rounds by leveraging the Labeled Distance Tree (LDT) structure. Throughout the execution of the algorithm, each fragment is maintained as an LDT, and from this point onward, we assume that any reference to a fragment refers to its corresponding LDT structure. In an LDT, every node knows its hop distance from the root (i.e., its depth in the tree). This structure enables the design of common procedures within fragments, such as broadcast and upcast, where nodes only need to be awake for a constant number of rounds.

To achieve this, a transmission schedule is constructed for each node, specifying, within a fixed block of rounds, the specific rounds during which the node must be awake to communicate with its parent and its children. A node's schedule depends on its depth in the tree as to propagate information from the root to all nodes in the fragment, each node needs to be awake only in the same round as its parent to receive the message, and in one subsequent round to forward it to its children. In all other rounds, the node can sleep, as it is neither sending nor receiving messages.

The transmission schedule is formalized with the function $Transmission - Schedule(root, u, n)$ which takes a node u in an LDT rooted at $root$ and returns a subset of rounds within a block of rounds of length $2n + 1$ that u must be awake in. Let i be the depth of u in the LDT, $Transmission - Schedule$ maps a non-root node u to the following rounds with corresponding round names in the block of $2n+1$ rounds: i (Down-Receive), $i+1$ (Down-Send), $n+1$ (Side-Send-Receive), $2n-i+1$ (Up-Receive), $2n-i+2$ (Up-Send). Notice that it doesn't make sense for the root node to be mapped to a Down-Receive or Up-Send round as the root has no parent node. $Transmission - Schedule$ therefore maps the node $root$ to the following rounds with corresponding round names in the block of $2n + 1$ rounds: 1 (Down-Send), $n + 1$ (Side-Send-Receive), $2n + 1$ (Up-Receive).

Utilizing the transmission schedule function we can define procedures for broadcasting a message in a tree, upcasting the minimum value in a tree and transmitting a message between nodes of adjacent trees in $O(1)$ awake rounds. Clearly, these procedures require $O(n)$ total rounds (i.e., awake and sleeping rounds) as the length of the fixed block of rounds within which awake rounds are allocated is $O(n)$. By maintaining each fragment as an LDT, each node can determine its transmission schedule by invoking $Transmission - Schedule(root, u, n)$. Depending on the procedure, nodes execute different logic associated with the awake rounds in their transmission schedule. For a message to be broadcast from the root to all nodes in the tree, in the Down-Receive round, nodes listen for a message from its parent, then in the Down-Send round, nodes transmit the message it received to its children in the tree. This procedure is referred to as FRAGMENT-BROADCAST. Similarly, we define a procedure, UPCAST-MIN, which is executed by the nodes of a fragment to propagate the minimum value held among them up the tree to the root. First, all nodes invoke the $Transmission-Schedule$ function to determine their awake rounds. Then in the Up-Receive round, nodes listen for messages from their children (if any), and in the Up-Send round, nodes find the minimum of the values it received and its own value and send it to its parent in the tree. Lastly, we define a procedure, $Transmit - Adjacent$, executed by the nodes of a fragment to transmit messages between neighbouring fragments. Whenever this procedure is invoked, all nodes in the network execute it simultaneously, using transmission schedules utilizing the same fixed block of rounds. This procedure allows information such as a fragment's head or tail label and its MOE to be shared between fragments. Sharing this information is essential for determining whether an MOE is valid and for facilitating the merging procedure.

We can now provide a high-level description of the procedure MERGING-FRAGMENTS. Before this procedure is invoked, head fragments are aware of any valid MOEs incident to their fragment, and tail fragments are aware of whether their own MOE is valid. In particular, the nodes adjacent to a valid MOE in the corresponding head and tail fragments are aware that they are adjacent to a valid MOE. We refer to the head fragment as H and the tail fragment as T, and the nodes adjacent to the valid MOE in H and T as u_H and u_T, respectively.

First, all nodes invoke the TRANSMIT-ADJACENT procedure to transmit their fragment identifier (i.e., the identifier of the root of the fragment) as well as their depth within their respective trees. Using this information, u_T in T can determine its new depth relative to H and its new fragment identifier. Additionally, u_H must update its child pointers to

include u_T, and u_T must change its parent pointer to point to u_H.

However, nodes must defer applying these updates until the end of the merging procedure, as the existing fragment structures must be maintained to allow the merging procedure to terminate correctly. Nodes therefore store their new fragment identifier, depth, parent pointer, and child pointers in temporary variables, which are used to update the actual values at the end of the procedure.

The fragment H maintains its structure aside from u_H adding a child pointer to u_T. Within fragment T, however, the new fragment identifier and depth information must be propagated to the nodes. To achieve this in a constant number of awake rounds, a procedure analogous to upcast is first used to propagate the updates from u_T up to the root of T, followed by a broadcast-like procedure to distribute the updates from the root down to the remaining nodes.

Detailed descriptions of this procedure are omitted, as they are not directly relevant to the optimizations we propose. Once this information has been propagated throughout T, it forms a subtree of H, with all nodes previously in T aware of their new depth relative to H, having adopted H's fragment identifier, and with parent and child pointers updated accordingly.

Using the procedures FRAGMENT-BROADCAST, UPCAST-MIN, TRANSMIT-ADJACENT and MERGING-FRAGMENTS, fragments can, within a given phase, find their MOE, confirm that it is valid and merge with other fragments in $O(1)$ awake rounds and $O(n)$ total rounds. It is shown in the analysis of [1] that after $O(\log n)$ phases, exactly one fragment remains with high probability, resulting in an overall running time of $O(n \log n)$ and awake time of $O(\log n)$.

3.2 Optimized Algorithm

In this subsection, we outline our modifications to the original algorithm. We separate our optimizations into two main concepts:

- Adaptive transmission scheduling
- Procedure-specific transmission scheduling

3.2.1 Adaptive transmission scheduling

In the baseline algorithm, the transmission schedule function assign a node awake rounds within a fixed block of $2n + 1$ rounds. To understand why a fixed block of length $2n+1$ rounds is always sufficient, we can consider the worst-case scenario. This would occur if a fragment contained all n nodes of a network arranged in a path with the root located at either end of this path. This fragment would have a maximum depth of $n - 1$ and therefore, propagating a message from the root to the leaf would require $n-1$ rounds. To propagate information from the leaf to the root would require another $n - 1$ rounds. Additionally, a single round (Side-Send-Receive) is reserved for communication between neighbouring fragments. Therefore, at least $2n - 1$ rounds would be required to construct a transmission schedule for this fragment. We note that the baseline algorithm uses a block length of $2n + 1$ rounds, these extra two rounds correspond to a Down-Send and Up-Receive round for leaf nodes which are not required since leaf nodes don't have any children. Since no fragment can have a depth exceeding $n - 1$, the block length of $2n + 1$ provides an upper bound that guarantees correctness, regardless of the maximum depth of the fragments in the network in any phases.

While the fixed block length of $2n + 1$ ensures the algorithm is always correct, it often provides an overly conservative bound of the required block length for a given phase. This is particularly relevant for early phases of the algorithm when the depth of the fragments is shallow. This leads to wasted idle rounds where nodes within a fragment are not sending or receiving messages. The key insight is that the required block length for communication within a fragment depends on the fragment's depth. Specifically, a transmission schedule for a given fragment can be constructed using a block of $2d+3$ rounds, where d is the depth of the fragment.

However, since the baseline algorithm is designed to proceed in synchronous phases, all fragments must operate on a common schedule. This is achieved in the baseline algorithm by using the same block length of $2n+1$ rounds for all transmission schedules across all fragments and phases. As a result, if we are to use different length schedules in each phase that depend on the depth of the fragments, it must accommodate the deepest fragment in the network in that phase. Therefore, we use the maximum depth among all of the fragments in the network in a given phase to determine the length of the block of rounds used for every nodes transmission schedule.

Let $\mathcal{F}_i$ denote the set of fragments present at the beginning of phase i. For each fragment $f \in \mathcal{F}_i$, let d_f denote its depth in phase i. Define the maximum fragment depth in phase i as $D_i = \max_{f \in \mathcal{F}_i} d_f$. Since all nodes must follow a common schedule in a given phase, we set the schedule length based on D_i. In phase i, every node uses a of block of $2D_i+3$ rounds for its transmission schedule. This adaptive approach ensures that all fragments, regardless of their individual depths, have a sufficient number of rounds within which transmission schedules can be constructed for all nodes. Doing this we reduce the number of rounds required to perform fragment procedures, particularly in phases when all fragments in the network are shallow.

We formalize this using the function $Adaptive - Transmission - Schedule(root, u, d)$ which takes a node u rooted at $root$ and returns a subset of rounds within a block of rounds of length $2d + 3$ that u should be awake in. Let i be the depth of u in the tree, $Adaptive - Transmission - Schedule$ maps a non-root node u to the following round names in the $2d + 3$ block of rounds: i (Down-Receive), $i + 1$ (Down-Send), $d + 2$ (Side-Send-Receive), $2d - i + 3$ (Up-Receive), $2d - i + 4$ (Up-Send). For the root node $Adaptive - Transmission - Schedule$ maps the node to the following rounds with corresponding round names in the $2d + 3$ block of rounds: 1 (Down-Send), $d + 2$ (Side-Send-Receive), $2d + 3$ (Up-Receive). If we let $d = D_i$, since $D_i \leq n - 1$, we have $2D_i + 3 \leq 2n + 1$; that is, our modification never exceeds the length of the transmission schedule used by the baseline algorithm.

Coordinating all fragments on a common schedule using the adaptive transmission schedule requires global knowledge of the maximum fragment depth D_i in each phase. This can be achieved using a modification to a well known

algorithm in distributed computing called flooding. It allows a message to be broadcast to all the nodes of a network. However, it is expensive in terms of round and awake complexity. Flooding has a round and awake complexity of $O(D)$ where D is the hop-diameter of the network. In a general graph D can be $O(n)$, therefore, in the worst-case flooding can take $O(n)$ awake and total rounds. To introduce a procedure requiring $O(n)$ awake rounds each phase would increase the awake complexity of the original algorithm to be $O(n \log n)$. However, the round and awake complexity of flooding is $O(D)$, therefore, for networks of diameter $O(1)$, the procedure requires $O(1)$ awake time. Therefore, introducing this in each phase would not increase the asymptotic awake complexity of the algorithm. It is therefore possible in graphs of constant diameter to flood the maximum of the depths of the fragments in the network to all nodes in $O(1)$ awake rounds and total rounds.

To formalize this we introduce a simple procedure, $Flood - Max$, which is executed by all nodes in a network to allow all nodes to learn the value of D_i. This procedure requires D consecutive rounds where D is the hop-diameter of the network. In the first round, each node sends its own depth within its fragment to all its neighbours (not just those within its own fragment). In each of the remaining $D - 1$ rounds, every node sends the maximum of its current value and any values received from its neighbours. Since at least one node has the maximum depth and any two nodes are at most D hops apart, the maximum value propagates across the network in at most D rounds. Thus, after D rounds, all nodes will have received the maximum depth.

The focus of this work are networks with diameter 3. In these networks, the $Flood - Max$ procedure requires 3 rounds to complete. The procedure is executed by all nodes at the end of each phase so that, in the next phase, every node can use the value of D_i to initialize the parameter d in $Adaptive - Transmission - Schedule(root, u, d)$.

3.2.2 Procedure-specific transmission scheduling

In the $Transmission - Schedule$ and $Adaptive - Transmission - Schedule$ functions both assign nodes awake rounds corresponding to Down-Receive, Down-Send, Side-Send-Receive, Up-Receive, Up-Send rounds. The key observation here is that for a procedure such as $Fragment - Broadcast$, we only specify specific behaviour for the rounds corresponding to Down-Receive and Down-Send. Therefore, the remaining awake rounds can be omitted. This reduces the number of rounds required for the transmission schedules block length. For the $Fragment - Broadcast$ procedure, we can design a procedure specific transmission schedule function that assigns nodes awake rounds in a block of rounds of length D_i+1 rounds reducing the total number of rounds required for this procedure by a factor of 2. Similarly, for the $Transmit - Adjacent$ procedure, we can reduce the required block length from $2D_i+3$ to a single round that corresponds to the Side-Send-Receive round which occurs in the same round for all nodes. This optimization does also yield a minor reduction in the number of awake rounds which depends on the number of awake rounds omitted for each procedure.

3.2.3 Correctness

To establish the correctness of the adaptive transmission schedule, we begin by formalising the constraints exploited by the transmission schedule function of the baseline algorithm in [1]. We define a valid transmission schedule as any assignment that satisfies these constraints.

Definition 1 (Valid Transmission Schedule)**.** *A valid transmission schedule function assigns non-root nodes awake rounds for the following named rounds: Down-Receive, Down-Send, Side-Send-Receive, Up-Receive, and Up-Send. For root nodes it assigns the following named rounds: Down-Send, Side-Send-Receive, and Up-Receive.*

Such an assignment is valid if,

- *Each node is assigned unique awake rounds for each named round, and they're assigned in the order specified above.*
- *All nodes are assigned the same Side-Send-Receive round.*
- *A nodes Down-Receive round occurs in the same round as its parents Down-Send round.*
- *A nodes Up-Send round occurs in the same round as its parent's Up-Receive round.*
- *All rounds are contained within the specified block of rounds.*

These conditions ensure that messages are able to propagate from the root to the leaf nodes, inter-fragment communication can occur in a globally synchronized round and messages can propagate from the leaf nodes back to the root all within the specified block of rounds.

We now show that our adaptive schedule function satisfies this definition when parametrised with the value D_i. This guarantees that a block of $2D_i + 3$ rounds suffices to construct a valid transmission schedule for all nodes in phase i.

Lemma 1. *The $Adaptive - Transmission - Schedule$ function produces valid transmission schedules in every phase i, when it is parametrized using D_i.*

Proof. For a non-root node at depth d ($1 \leq d \leq D_i$) the $Adaptive - Transmission - Schedule$ function parametrized with D_i assigns: d (Down-Receive), $d + 1$ (Down-Send), $D_i + 2$ (Side-Send-Receive), $2D_i - d + 3$ (Up-Receive), $2D_i - d + 4$ (Up-Send). The root node of the fragment is assigned: 1 (Down-Send), $D_i + 2$ (Side-Send-Receive), $2D_i + 3$ (Up-Receive).

First, we check the ordering of the assigned named rounds and that their unique. For every d, $d < d + 1 < D_i + 2 < 2D_i - d + 3 < 2D_i - d + 4$, so each non-root nodes named rounds are distinct and appear in the required order. The roots assigned named rounds are also strictly increasing $1 < D_i + 2 < 2D_i + 3$ given that $D_i \geq 0$.

Next we check the constraints between parent and children.

First we check the Down-Receive round of a non-root node coincides with the Down-Send of its parent. A non-root node at depth d is assigned Down-Receive in round d; if its parent, at depth $d - 1$, is also a non-root node, it is assigned Down-Send in round $(d-1)+1 = d$. Thus Down-Receive coincides with the parent's Down-Send. We must check this holds when a non-root nodes parent is the root.

This only occurs for non-root nodes at depth 1 which are assigned their Down-Receive round in round 1. The roots Down-Send occurs in round 1, so it holds.

Next we check the Up-Send round of a non-root node coincides with the Up-Receive round of its parent. A non-root node at depth d is assigned Up-Send in round $2D_i - d + 4$; if its parent, at depth $d-1$, is also a non-root node, it is assigned Up-Receive in round $2D_i - (d-1) + 3 = 2D_i - d + 4$. Thus Up-Send coincides with the parents Up-Receive. We must check this holds when a non-root nodes parent is the root. This only occurs for non-root nodes at depth 1 which are assigned their Up-Send round in round $2D_i - (1) + 4 = 2D_i + 3$. The root is assigned Up-Receive in round $2D_i + 3$, so it holds.

Lastly, we must check that all nodes are assigned the same Side-Send-Receive round. Both non-root and root nodes are assigned their Side-Send-Receive round in round $D_i + 2$, so it holds.

The earliest slot used by any node is round 1 (the root's Down-Send and Down-Receive for non-root nodes at depth 1), and the latest is round $2D_i+3$ (both the root's Up-Receive and the Up-Send for non-root nodes at depth 1). Hence the entire schedule fits inside a block of $2D_i + 3$ rounds.

Therefore, all conditions of Definition 1 hold, completing the proof. □

This confirms that our adaptive transmission schedule satisfies the definition of a valid transmission schedule. Because all fragment procedures analysed in [1] rely only on these conditions, substituting our adaptive schedule preserves the correctness of those procedures.

Finally, Our definition of a valid transmission schedule can also be adapted to procedure-specific transmission schedules to show they produce valid transmission schedules when not all of the named rounds are used. Intuitively, the correctness of the baseline algorithm is preserved when certain named rounds are omitted, provided that the corresponding procedure does not assign any logic to those rounds. In other words, if no logic depends on a particular named round, removing it from the transmission schedule does not affect the algorithm's correctness.

3.3 Empirical Scaling Analysis

To demonstrate the benefits and disadvantages of our optimizations, we conducted a series of empirical investigations comparing the baseline and optimized algorithms. Following the approach outlined by Hoos [14], we analysed the empirical scaling behaviour of the total and awake rounds required by each algorithm on randomly generated and adversarially constructed graphs of increasing size. Using standard numerical techniques, we fitted parametric models to a subset of the empirical data, referred to as the support set. These models were then challenged by extrapolation, comparing their predictions on instances larger than those used for fitting the model against the held out observed data for these instances. These larger instances, beyond the support set, are referred to as the challenge set. To assess the validity of a scaling model, a statistical method called bootstrap analysis is used to obtain bootstrap confidence intervals (CIs) for the model's predictions, which are then compared against the observed data. To construct a bootstrap CI (CI), multiple resamples are drawn with replacement from the original observations in the support set, and the effect size of interest is computed for each resample. In our case, we generate multiple resamples of the observed round and awake complexity data, fitting a separate scaling model to each resample to obtain a distribution of predictions. This distribution is then used to obtain the CI for the predictions of the scaling model. To obtain bootstrap CIs for the observed data, we resample with replacement from the original observations across all instances for a given instance size and compute the mean value, from which we can calculate the empirical distribution of the mean. We then compute the appropriate percentiles of this empirical distribution to construct the bootstrap CI at the desired confidence level. In our analysis, we consistently use a 95% confidence level for all reported intervals.

To evaluate how well a scaling model aligns with observed performance data, we compare the predicted and observed CIs. A scaling model is considered weakly consistent if the CIs of the observed data overlap with the predicted CIs, and strongly consistent if the observed CIs are fully contained within the predicted ones. Conversely, a model is considered inconsistent with the observed data if the observed CIs are disjoint from the predicted CIs. The proportion of model predictions that are weakly or strongly consistent determines the level of support for the scaling model. This approach using bootstrap resampling is particularly advantageous because it does not require any assumptions about the underlying distribution of the round and awake times for instances of a given size.

To collect the empirical data, we implemented both the baseline and optimized algorithms. A custom simulation environment was developed in Python to simulate distributed algorithms in the sleeping model, as no existing solution was available. The environment keeps track of the number of awake rounds and total (awake and sleeping) rounds used by each node in the network.

As performance was measured in terms of synchronous rounds (rather than wall-clock time), the results are independent of low-level aspects of the execution environment, such as the hardware used to run the simulations or the programming language in which they were implemented. We therefore omit the details of the specific hardware on which the benchmarking was performed as they have no bearing on the results.

Two separate classes of benchmark instances were constructed for our analysis. The first was a set of random benchmark instances generated using the Erdős–Rényi random graph model [26]. In this model, a random graph is constructed by adding edges between n labelled nodes, where each possible edge is included independently with probability p. To generate graphs with a specified diameter, we randomly sample values of p in the interval $[0, 1]$, generate a random n-vertex graph using the Erdős–Rényi model using the NetworkX Python package [27], and then verify that the resulting graph is connected and has the desired diameter. If a valid graph is obtained, we assign distinct weights to its edges uniformly at random with weights of size $O(\log n)$. The second class of benchmark instances consisted of carefully constructed adversarial inputs designed

to minimize the potential benefits of our proposed optimizations. Specifically, we constructed networks whose MST forms a path of n nodes. In such networks, the maximum fragment depth tends to grow to be large by the final phase. The intuition is that by forcing a large maximum fragment depth in the final phase, and since each phase's duration depends on this depth, these adversarial instances deliberately extend the length of each phase, thereby maximizing the overall round complexity of the optimized algorithm. Adversarial instances with this property are therefore expected to expose worst-case behaviour of the optimized algorithm. Importantly, these adversarial inputs should not impact the baseline algorithm in the same way. Both algorithms require the same number of phases in expectation, and the baseline uses a fixed-length schedule per phase that does not depend on the fragment depth only on the size of the network n.

The validity of the fitted models typically benefits from a large range of instance sizes. To support this, we used as many distinct graph sizes as computationally feasible, given the limited overall computation time available to run the simulations. The instances sizes used form a geometric series spanning instances of size $12 \leq n \leq 446$.

As outlined earlier, when describing the merging criteria, the algorithms are randomized. Fragments flip a coin each phase which determines whether their MOE is valid, introducing stochasticity into the algorithms. This source of variability needed to be accounted for to ensure a rigorous empirical evaluation. Following established practices [14], for each instance size, we generate 1000 random graph instances and execute the algorithm 5 times per instance, each with a different random seed. This allows us to characterize the distribution of round and awake metrics within instances.

For each instance, we compute the median number of rounds across its runs; we then take the mean of these per-instance medians across all instances of the same size to report an overall mean-of-medians.

To quantify uncertainty in the fitted models, we used bootstrap analysis [14]. For each algorithm, we constructed 1,000 bootstrap resamples of the support data. Specifically, for each graph size, we sampled (with replacement) from the set of per-instance medians to create a new synthetic sample. For each resample, we computed the mean round complexity at every input size, then fitted a parametric model using the Levenberg-Marquardt Algorithm. This yielded 1,000 fitted models per algorithm.

From the resulting family of models, we derived the bootstrap CI for both the fitted model parameters and the predicted number of rounds for each instance size in the challenge set. This enabled us to obtain 95% bootstrap CI around the fitted scaling curves which is used to determine whether differences in scaling behaviour are statistically significant. Specifically, to determine extent to which the scaling behaviour of the optimized algorithm differs from that of the baseline algorithm, we compared the observed statistics of the optimized algorithm to the predicted bootstrap CIs of the scaling models of the baseline algorithm. If the bootstrap CIs for the baselines scaling model do not contain the observed statistics of the optimized algorithm, we can reject the hypothesis that the scaling behaviour of the optimized algorithm is consistent with the scaling behaviour of the baseline algorithm.

Furthermore, to quantify practical performance differences between the baseline and optimized algorithms we used the bootstrap CIs for the observed performance data of both algorithms. Using these intervals, we derived bootstrap CIs for the speed-up factors achieved per instance size by the optimized algorithm. This approach allowed us to characterize the practical advantage of the optimized algorithm over the baseline across the observed range of problem instances, with 95% confidence.

For our scaling analysis of round complexity, we considered the following parametric models:

- $Linearithmic\,[a,b]\,(n) = a \cdot n \cdot \log(x) + b$ (2-parameter Linearithmic)
- $Quasilinear\,[a,b,c]\,(n) = a \cdot n \cdot \log^b(n) + c$ (3-parameter Quasilinear)
- $Polylogarithmic\,[a,b,c]\,(n) = a \cdot \log^b(n) + c$ (3-parameter Polylogarithmic)
- $Poly\,[a,b,c]\,(n) = a \cdot n^b + c$ (3-parameter Poly)

For our scaling analysis of awake complexity, we considered two parametric models:

- $Log\,[a,b]\,(n) = a \cdot \log(n) + b$ (2-parameter Log)
- $Polylogarithmic\,[a,b]\,(n) = a \cdot \log^b(n)$ (2-parameter Polylogarithmic)

This approach to empirical scaling analysis was implemented as a fully automated tool called Empirical Scaling Analyser (ESA), which we used to perform all our analyses [28].

4 Results

4.1 Round Complexity

We fitted parametric models to the mean number of total rounds required by the baseline and optimized algorithms, for both the random and adversarial benchmark instances spanning sizes $12 \leq n \leq 215$ (support set). This resulted in the fitted scaling models shown in Table 1. The models were evaluated based on the root mean squared error (RMSE) of their predictions compared to the observed data in the challenge set ($258 \leq n \leq 446$). For the random instances, a quasilinear model and a polynomial model (with an exponent < 1) provided the best fit for the baseline and optimized algorithms, respectively. For the adversarial instances a quasilinear model provided the best fit for both the baseline and optimized algorithm. We note that the small negative exponent in the quasilinear model fitted to the optimized algorithm on adversarial inputs suggests that the model may be overfitting the observed data. The fitted exponent of 0.98604 in the polynomial model for the adversarial instances indicates nearly linear scaling, suggesting that the observed data could be consistent with a linear scaling model. Consequently, we also fitted a linear model to the observed data, $Linear\,[a,b]\,(n) = a \cdot n + b$ (2-parameter Linear).

To assess the confidence in the fitted models, we applied the bootstrap analysis approach described in Section 3 to determine CIs for both the predicted and observed total number of rounds required. To keep the presentation concise, we report only the results for the models that performed best in

TABLE 1
Fitted models of the means of the per-instance median of the total number of rounds and RMSE values (in rounds) for the baseline and optimized algorithm on the random and adversarial benchmark instances. The models yielding the most accurate predictions (as per RMSEs on challenge data) are shown in boldface.

Algorithm	Instances	Model		RMSE	
		Name	Equation	Support	Challenge
Baseline	Random	Linearithmic	$77.48 \cdot n \cdot \log(n) + 479.53$	1.4826×10^5	1753.6
		Quasilinear	$\mathbf{85.674 \cdot n \cdot \log^{0.93953}(n) + 267.83}$	**62779**	**569.41**
		Polylogarithmic	$4.0057 \cdot \log^{5.9286}(n) + 2360$	1.7762×10^6	13604
		Poly	$159.06 \cdot n^{1.1808} - 390.03$	2.2687	2819.1
	Adversarial	Linearithmic	$77.692 \cdot n \cdot \log(x) + 295.85$	3.0555×10^6	2560
		Quasilinear	$\mathbf{79.781 \cdot n \cdot \log^{0.98403}(n) + 240.37}$	$\mathbf{3.0495 \times 10^6}$	**2347.7**
		Polylogarithmic	$3.7622 \cdot \log^{5.9684}(n) + 2278.1$	4.5114×10^6	12933
		Poly	$152.35 \cdot n^{1.1896} + -432.83$	5.2594	4295.6
Optimized	Random	Linearithmic	$2.2411 \cdot n \cdot \log(n) + 789.11$	3.6324×10^5	1720.7
		Quasilinear	$315.41 \cdot n \cdot \log^{-1.8425}(n) + 22.558$	80357	621.62
		Polylogarithmic	$27.064 \cdot \log^{2.7001}(n) + 298.7$	520.21	118.44
		Poly	$\mathbf{302.77 \cdot n^{0.43591} + -290.35}$	**0.42585**	**21.213**
	Adversarial	Linearithmic	$6.8481 \cdot n \cdot \log(x) + 642.64$	2.3363×10^5	1779.3
		Quasilinear	$\mathbf{42.449 \cdot n \cdot \log^{-0.087092}(n) + 203.2}$	**24574**	**69.802**
		Polylogarithmic	$1.5494 \cdot \log^{5.034}(n) + 577.41$	69752	1019.3
		Poly	$39.458 \cdot n^{0.98604} + 218.54$	1.5698	72.869
		Linear	$36.627 \cdot n + 247.84$	1.5851	180.43

the extrapolation tests for each algorithm and benchmark set, as these models are most consistent with the observed scaling behavior. The bootstrap CIs for the predictions of the selected models on the challenge instances, alongside the CIs for the observed number of total rounds, are shown in Table 2.

When challenged with extrapolation to the challenge set on random instances, the baseline algorithm's linearithmic and polynomial model (with an exponent > 1) tended to over-estimate the observed data, while the polylogarithmic model tended to under-estimate it. In contrast, the quasilinear model fit the data very well. For the optimized algorithm, the linearithmic and quasilinear models tended to overestimate, the polylogarithmic model tended to underestimate, and the polynomial model (with an exponent < 1) fit the data very well.

On the adversarial instances, none of the models fit the baseline algorithm's observed data well. The polylogarithmic model continued to underestimate the data, and although the remaining models exhibited poor fits, there was insufficient evidence to reject any of them at the 95% confidence level. For the optimized algorithm, however, the quasilinear and polynomial model (with exponent ≈ 1) fit the observed data very well, while the linearithmic model over-estimated and the polylogarithmic model under-estimated the data as before. The linear model we considered also tended to fit the data, providing a good fit despite having fewer parameters than the polynomial and quasilinear models.

Figures 1 and 2 show the most promising fitted models for the baseline and optimized algorithms, respectively, on both random and adversarial instances.

To assess the differences between the scaling models of the baseline and optimized algorithms, we use bootstrap CIs. On random instances, the observed number of rounds for the optimized algorithm is inconsistent with the baseline's quasilinear scaling model across all input sizes ($12 \leq n \leq 446$), confirming that the optimized algorithm scales better. Similarly, on adversarial instances, the observed number of rounds for the baseline algorithm is inconsistent with the predictions of the optimized algorithm's polynomial scaling model, further confirming that the optimized algorithm achieves better scaling across both benchmarks.

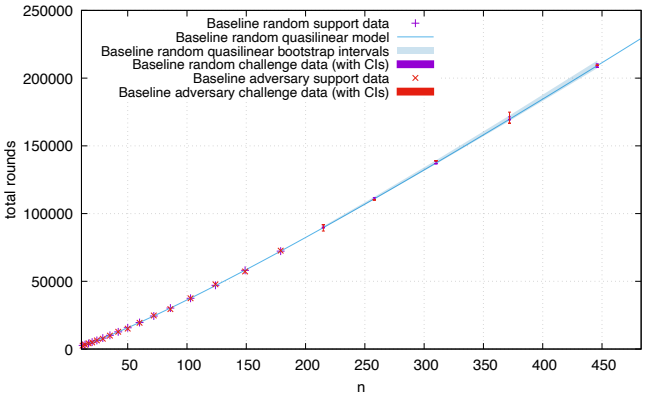

Fig. 1. Fitted model for the mean number of total rounds required by the baseline algorithm on both random and adversarial benchmark instances. The models are fitted using data from instances of size $12 \leq n \leq 215$ and validated against instances of size $258 \leq n \leq 446$. Fitted curves for the baseline algorithm on the adversarial instances were omitted as they all provide a poor fit, though both support and challenge data for adversarial instances are still shown.

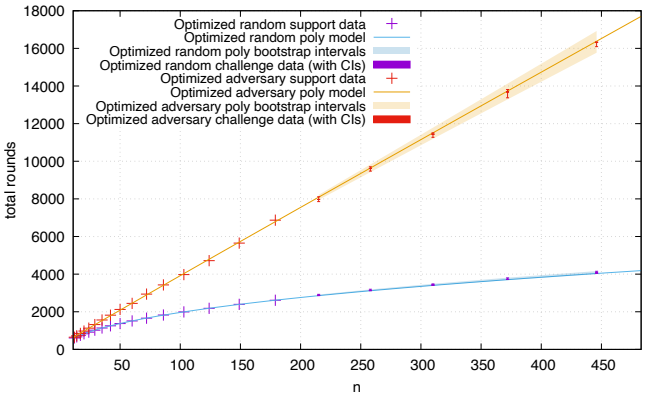

Fig. 2. Best fitted models for the mean number of total rounds required by the optimized algorithm on both random and adversarial instances. The models are fitted using data from instances of size $12 \leq n \leq 215$ and validated against instances of size $258 \leq n \leq 446$. Separate fitted curves are shown for the random and adversarial benchmarks.

To further compare the scaling behaviour of the baseline and optimized algorithms on the random benchmark instances, we examine the quasilinear and polynomial scaling models which are most consistent with their respective observed data. Since a quasilinear model of the form $n \log^b(n) \in \omega(n^{b'})$ for any $b' < 1$, the baseline algorithm's empirical scaling model grows strictly faster, indicating that the optimized algorithm scales more efficiently for large n.

For the adversarial benchmark instances, since none of the fitted models accurately fit the observed data for the baseline algorithm, we are unable to make a reliable comparison based on difference in observed scaling behaviour. Instead we can directly compare the observed number of total rounds required by both algorithms across the range of instance sizes we considered ($12 \leq n \leq 446$). As outlined in section 3, we derive bootstrap CIs for the speed-up factors achieved by the optimized algorithm by using the

TABLE 2
95% bootstrap CIs for the predictions of the mean number of total rounds and the observed total rounds on the problem instances. The instance sizes shown here are larger than those used for fitting the models. Bootstrap intervals on predictions that are weakly consistent with the observed data are shown in boldface and those that are strongly consistent are marked by asterisks (*).

Algorithm	Instances	Model	n	Predicted confidence intervals	Observed total rounds: Point estimate	Observed total rounds: Confidence interval
Baseline	Random	Quasilinear $[a, b, c]$ $a \in [78.454, 95.938]$ $b \in [0.87287, 0.99728]$ $c \in [101.54, 456.7]$	215	$[\mathbf{8.944 \times 10^4}, \mathbf{9.068 \times 10^4}]$	8.954×10^4	$[8.96 \times 10^4, 9.082 \times 10^4]$
			258	$[\mathbf{1.105 \times 10^5}, \mathbf{1.124 \times 10^5}]$	1.106×10^5	$[1.103 \times 10^5, 1.118 \times 10^5]$
			310	$[\mathbf{1.366 \times 10^5}, \mathbf{1.394 \times 10^5}]$*	1.369×10^5	$[1.366 \times 10^5, 1.385 \times 10^5]$
			372	$[\mathbf{1.684 \times 10^5}, \mathbf{1.724 \times 10^5}]$*	1.695×10^5	$[1.691 \times 10^5, 1.714 \times 10^5]$
			446	$[\mathbf{2.073 \times 10^5}, \mathbf{2.129 \times 10^5}]$*	2.081×10^5	$[2.079 \times 10^5, 2.106 \times 10^5]$
Optimized	Random	Poly $[a, b, c]$ $a \in [259.97, 348.85]$ $b \in [0.41433, 0.46214]$ $c \in [-366.76, -199.83]$	215	$[\mathbf{2858}, \mathbf{2914}]$	2874	$[2859, 2926]$
			258	$[\mathbf{3113}, \mathbf{3188}]$	3128	$[3123, 3197]$
			310	$[\mathbf{3388}, \mathbf{3488}]$*	3426	$[3405, 3480]$
			372	$[\mathbf{3684}, \mathbf{3812}]$*	3725	$[3716, 3801]$
			446	$[\mathbf{4000}, \mathbf{4163}]$*	4062	$[4052, 4151]$
	Adversarial	Poly $[a, b, c]$ $a \in [33.818, 46.542]$ $b \in [0.95392, 1.016]$ $c \in [163.08, 269.09]$	215	$[\mathbf{7944}, \mathbf{8214}]$	7967	$[7856, 8112]$
			258	$[\mathbf{9419}, \mathbf{9829}]$*	9620	$[9476, 9707]$
			310	$[\mathbf{1.119 \times 10^4}, \mathbf{1.179 \times 10^4}]$*	1.144×10^4	$[1.127 \times 10^4, 1.151 \times 10^4]$
			372	$[\mathbf{1.329 \times 10^4}, \mathbf{1.412 \times 10^4}]$*	1.368×10^4	$[1.337 \times 10^4, 1.381 \times 10^4]$
			446	$[\mathbf{1.577 \times 10^4}, \mathbf{1.692 \times 10^4}]$*	1.632×10^4	$[1.609 \times 10^4, 1.634 \times 10^4]$

bootstrap CIs of the observed data for both algorithms at the input sizes $n = 12, 20, 42, 86, 179, 446$. The resulting CIs are shown in Table 3 for both the random and adversarial benchmark instances.

TABLE 3
95% bootstrap confidence intervals for the speed-up factors for a given instance size achieved by the optimized algorithm relative to the baseline algorithm at selected instance sizes, shown separately for the random and adversarial benchmark instances.

Instance size n	Speed-up factor (95% CI): Random	Speed-up factor (95% CI): Adversarial
12	$[4.2786, 4.5109]$	$[3.7699, 4.3541]$
20	$[5.9655, 6.2574]$	$[4.9445, 5.0725]$
42	$[9.8753, 10.320]$	$[6.8973, 7.1405]$
86	$[16.195, 16.851]$	$[8.5268, 8.7108]$
179	$[26.988, 28.031]$	$[10.453, 10.773]$
446	$[50.084, 51.974]$	$[12.500, 13.151]$

4.2 Awake Complexity

We now present the results for the number of awake rounds required to complete the execution of each algorithm. Since our optimizations only introduce a constant number of awake rounds each phase which does not depend on the maxmium fragment depth in that phase, the adversarial instances were not included in the awake complexity analysis, as no significant differences in scaling behaviour is expected. All results presented here correspond to empirical data collected on the random benchmark instances.

Following the same process used to analyse round complexity, parametric scaling models were fitted to the mean number of awake rounds required by the baseline and optimized algorithms on instances in the support set, resulting in the fitted models shown in Table 4. Based on the RMSE values on the challenge set, the polylogarithmic model provides the best predictions for both the baseline and optimized algorithm on the challenge set.

TABLE 4
Fitted models of the means of the per-instance median number of awake rounds and RMSE values (in awake rounds).

Algorithm	Model		RMSE (support)	RMSE (challenge)
Baseline	Logarithmic	$99.374 \cdot log(n) + 3.5593$	17.344	3.5381
	Polylogarithmic	$\mathbf{101.67 \cdot \log^{0.9901}(n)}$	**0.25278**	**3.1408**
Optimized	Logarithmic	$150.53 \cdot log(n) + 8.1622$	75.918	3.4894
	Polylogarithmic	$\mathbf{155.87 \cdot \log^{0.9848}(n)}$	**0.36234**	**2.6937**

When challenged by extrapolation to the challenge set, we conclude that for both algorithms the log and polylogarithmic models tend to fit the data. The bootstrap confidence intervals for predictions of the polylogarithmic model and the observed awake rounds are shown in Figure 3.

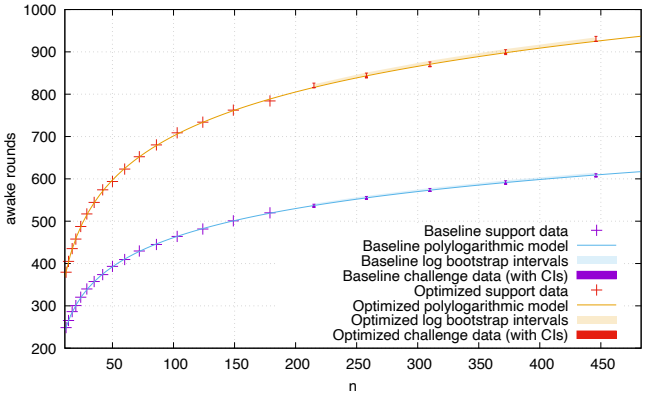

Fig. 3. Fitted models for the means of the per-instance median number of awake rounds for the baseline and optimized algorithms. The models are fitted using data from problem instances of size $12 \leq n \leq 179$ and are challenged by instances of size $215 \leq n \leq 446$.

TABLE 5
95% bootstrap confidence intervals for the means of the per-instance median predicted and observed awake rounds across problem instances. The instance sizes shown are larger than those used to fit the models. Bootstrap intervals on predictions that are weakly consistent with the observed awake rounds are shown in boldface.

Algorithm	Model	n	Predicted confidence intervals	Observed awake rounds	
				Point estimate	Confidence interval
Baseline	Polylogarithmic $[a, b]$ $a \in [101.73, 104.78]$ $b \in [0.97244, 0.99422]$	215	$[\mathbf{537.1}, \mathbf{541.3}]$	533.9	$[532.6, 540.1]$
		258	$[\mathbf{554.8}, \mathbf{559.5}]$	552.7	$[551.3, 558.4]$
		310	$[\mathbf{572.6}, \mathbf{577.9}]$	570.8	$[570.4, 577.3]$
		372	$[\mathbf{590.4}, \mathbf{596.2}]$	587.9	$[587.8, 595.5]$
		446	$[\mathbf{608}, \mathbf{614.3}]$	605.2	$[604.4, 612.1]$
Optimized	Polylogarithmic $[a, b]$ $a \in [156.03, 160.82]$ $b \in [0.96716, 0.9887]$	215	$[\mathbf{816.7}, \mathbf{822.8}]$	815.9	$[814.9, 826.2]$
		258	$[\mathbf{843.5}, \mathbf{850.5}]$	840.4	$[838.3, 849.8]$
		310	$[\mathbf{870.5}, \mathbf{878.2}]$	865.9	$[864.8, 876.2]$
		372	$[\mathbf{897.3}, \mathbf{905.7}]$	895.7	$[893.6, 905]$
		446	$[\mathbf{923.9}, \mathbf{933.1}]$	924.9	$[924.5, 936.6]$

As both algorithms are consistent with a log scaling model, we use the bootstrap CIs of the fitted model parameters, presented in Table 6, to compare how the number of awake rounds scales with instance size. Specifically, the scaling behaviour of both algorithms is consistent with the model $Log\,[a, b]\,(n) = a \cdot \log(n) + b$, where the coefficient a represents the dominant constant factor for large n. The ratio of these coefficients $\frac{a_{\text{optimized}}}{a_{\text{baseline}}}$ provides an estimate of the relative growth in awake rounds between the two algorithms. The 95% bootstrap CI for this constant factor slow down is $[1.4837, 1.5472]$, suggesting that the optimized algorithm requires approximately 1.5 times more awake rounds than the baseline algorithm for large n.

TABLE 6
95% bootstrap intervals of model parameters for the means of the per-instance median number of awake round for the baseline and optimized algorithms.

Algorithm	Model	Confidence interval of a	Confidence interval of b
Baseline	Log	$[98.281, 100.38]$	$[1.9587, 9.9591]$
Optimized	Log	$[148.93, 152.06]$	$[6.0525, 18.241]$

To investigate the significance of the differences observed between the scaling models of the baseline and optimized algorithms, we again use bootstrap CIs. Specifically, we want to determine whether there is sufficient evidence to suggest the scaling of the optimized algorithm is worse than that of the baseline algorithm. Using this approach, we conclude that the observed number of awake rounds for the optimized algorithm is inconsistent with the polylogarithmic scaling model for the baseline algorithm, that is the intervals are disjoint. This confirms that the awake complexity of the optimized algorithm scales worse than that of the baseline algorithm.

To assess the increase in the awake complexity of the optimized algorithm independently of the scaling models, we directly compare the observed awake times of both algorithms across the range of instance sizes ($12 \leq n \leq 446$). As outlined in section 3, we do this using the bootstrap CIs of the observed data for both algorithms to derive bootstrap CIs for the slow-down factor for the optimized algorithms for instances sizes $n = 12, 20, 42, 86, 179, 446$, these are shown in Table 7. The factors are consistent with the expected constant factor slow down of approximately 1.5 times, derived from the analysis of the fitted scaling models.

TABLE 7
95% bootstrap confidence intervals for the slow-down factors for the optimized algorithm compared to the baseline algorithm across selected instance sizes. The confidence intervals are derived from the observed number of awake rounds collected during the empirical study.

Instance size n	Slow-down factor (95% CI)
12	$[1.4841, 1.5595]$
20	$[1.4927, 1.5571]$
42	$[1.5049, 1.5612]$
86	$[1.5038, 1.5527]$
179	$[1.4868, 1.5295]$
446	$[1.5117, 1.5516]$

5 Evaluation

5.1 Round Complexity

The results of the empirical scaling analysis presented in section 4 clearly demonstrate that the optimized algorithm exhibits better scaling behaviour when simulated across the random and adversarial benchmark instances.

The empirical scaling analysis showed that the baseline algorithm's round complexity was most consistent with a quasilinear scaling model of the form $a \cdot n \cdot \log^b(n) + c$ on random instances. However, there was insufficient evidence to reject a simpler linearithmic model across both random and adversarial instances. This aligns with the theoretical worst-case bound of $O(n \log n)$ for the baseline algorithm, although the empirical data slightly favoured the more flexible quasilinear model. We believe that substantially increasing the number of instances per instance size would narrow the bootstrap CIs of the scaling models and likely reveal clearer differences between the linearithmic and quasilinear scaling behaviour.

From the analysis of Augustine et al. [1], we know that the baseline algorithm requires $O(\log n)$ phases. Since our optimizations do not modify the merging criteria used by the baseline algorithm, the number of phases required

by the optimized algorithm is also $O(\log n)$. However, the number of round required per phase differs between the two algorithms. In the baseline algorithm, a fixed transmission schedule of length $O(n)$ is used, resulting in $O(n)$ rounds per phase. In contrast, in the optimized algorithm, nodes use an adaptive transmission schedule parametrized by the maximum fragment depth D_i in phase i, resulting in transmission schedules of length $O(D_i)$. Consequently, each phase of the optimized algorithm requires $O(D_i)$ rounds, so the overall round complexity of the optimized algorithm can be expressed as $\sum_{i=1}^{O(\log n)} O(D_i)$.

In the worst case, $D_i \leq n-1$ for all i, and substituting $D_i = O(n)$ into the sum yields $\sum_{i=1}^{O(\log n)} O(n) = O(n \log n)$. Therefore, the worst-case asymptotic round complexity of the optimized algorithm remains $O(n \log n)$, consistent with that of the baseline algorithm.

Despite the asymptotic worst-case bounds, our empirical scaling analysis shows that on random instances, the optimized algorithm exhibits sublinear scaling of the round complexity, with the observed data most consistent with a polynomial model having an exponent of approximately 0.4. In contrast, on adversarial instances, the optimized algorithm exhibits scaling of the round complexity consistent with a polynomial model with an exponent close to 1.

To further investigate why the optimized algorithm exhibits better scaling in practise, and to understand the potential weaknesses of our optimizations, we perform two empirical investigations. First, we examine whether the number of phases required by the optimized algorithm on both random and adversarial instances is consistent with the theoretical bound $O(\log n)$. Second, we analyse how the maximum fragment depth scales with instance size.

To investigate the empirical scaling of the number of phases with instance size, we apply the same empirical scaling analysis approach used to assess round and awake complexity. We measured the number of phases required from executions of the optimized algorithm on both random and adversarial instances. The results, visualised in Figure 4, indicate that for both random and adversarial instances, the empirical scaling of the number of phases is consistent with a logarithmic model.

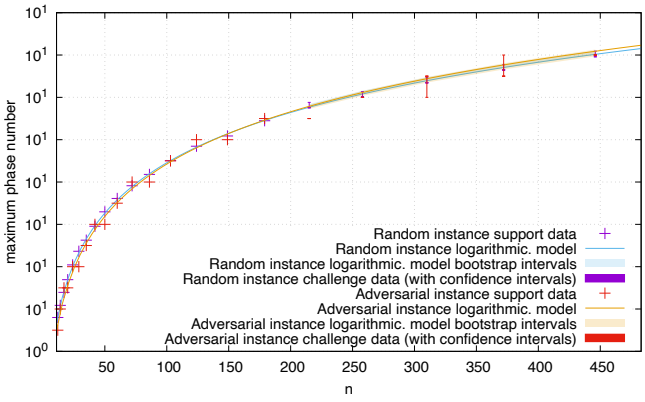

Fig. 4. Fitted models of the maximum phase. The models are fitted with the mean maximum phase required for the optimized algorithm for the random instances and adversarial instances of size $12 \leq n \leq 179$, and are challenged by the mean maximum phase for instances of size $215 \leq n \leq 446$.

Next, we apply a similar approach to analyse the impact of maximum fragment depth on round complexity. Directly comparing how D_i scales across all phases, either within a single instance or across instances of different sizes, presents challenges. Due to the randomization used in the algorithm, the number of phases can vary between instances, and as instance size increases, the number of phases required and the magnitude of the maximum fragment depths naturally increases. Consequently, comparisons of the scaling of D_i across phases and instances is difficult to interpret meaningfully.

Instead, we focus on the maximum fragment depth D_{max} reached during each execution. This is justified because the number of rounds required in each phase is upper bounded by D_{max}. Therefore, the scaling behaviour of D_{max} provides a good bound on the scaling of the overall round complexity, and analysing how D_{max} grows with instance size provides a meaningful way to explain the observed empirical scaling of the optimized algorithm.

The results of our empirical scaling analysis, visualised in Figure 5, show that the scaling of D_{max} with instance size for the random instances is sublinear, whereas the scaling of D_{max} for the adversarial instances is consistent with a linear model. This suggests the sublinear scaling of the round complexity observed for the optimized algorithm on random instances arises from the sublinear growth of D_{max} with instance size. Conversely, it suggests that the round complexity of the optimized algorithm on the adversarial instances should be consistent with the worst-case bound of $O(n \log n)$.

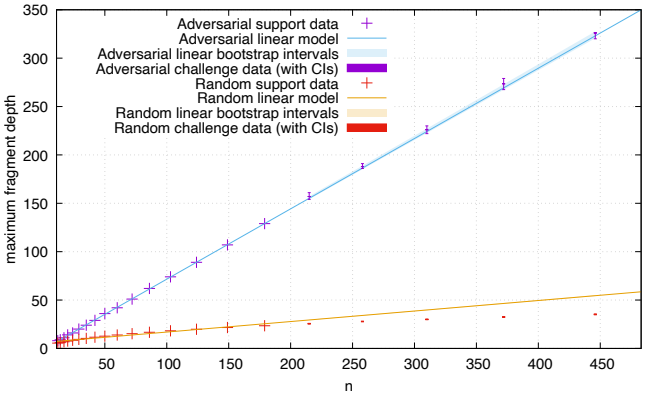

Fig. 5. Fitted models of the mean maximum fragment depth. The models are fitted with the mean maximum fragment depth for the optimized algorithm for the random instances and adversarial instances of size $12 \leq n \leq 179$, and are challenged by the mean maximum fragment depth for instances of size $215 \leq n \leq 446$.

Our empirical scaling analysis of the optimized algorithm's round complexity on adversarial instances was inconclusive. Given that we did not obtain sufficient evidence to reject an $O(n \log n)$ model, we believe further investigations using a larger set of adversarial instances could clarify whether the observed scaling behaviour matches the asymptotic worst-case bound. However, constructing a large set of adversarial instances proved challenging, as the construction method is highly specific and does not easily scale to more instances.

We were not able to answer the research question in the affirmative as our theoretical analysis establishes an $O(n \log n)$ upper bound for the round complexity of the optimized algorithm, confirming that our proposed optimizations do not yield an asymptotic improvement over the the current best algorithm.

5.2 Awake Complexity

The results of our empirical scaling analysis indicate that the optimized algorithm requires approximately 1.5 times more awake rounds than the baseline algorithm as input size increases. This can be attributed to the constant number of extra awake rounds required to perform the $Flood-Max$ procedure used to disseminate the maximum fragment depth throughout the network at the end of each phase. Although this represents an increase in awake rounds, the increase is only in a constant-factor meaning the increase remains constant across all n. The constant-factor increase is therefore small relative to the reduction in total rounds observed which exhibit scaling behaviour consistent with models that grow strictly slower.

5.3 Empirical Scaling Analysis

Empirical scaling analysis was chosen as the primary evaluation method, following the framework introduced by Hoos [14], due to its ability to characterise the practical performance of algorithms where theoretical analysis is insufficient or inconclusive. This method enabled us to quantify improvements in both round and awake complexity, and to explore the influence of structural properties of the input instances.

However, empirical scaling analysis has inherent limitations. It only allows reasoning about the algorithm's behaviour on a specific set of benchmark instances and cannot provide formal asymptotic guarantees. As discussed throughout this section, the results depend on the construction of the benchmarks used, which may not fully represent the whole space of possible inputs. In particular, the number of adversarial instances we were able to generate was limited by the nuances of its construction, limiting the conclusions we were able to make.

Nonetheless, empirical scaling analysis was instrumental in guiding the design of our optimizations. It provided rapid feedback and enabled meaningful comparisons between the baseline and optimized algorithms, particularly when we were unable to reason about the behaviour theoretically.

5.4 Approach

Our approach combined theoretical analysis with empirical simulations of both the baseline and optimized algorithm. This methodology differs from the traditional focus in distributed algorithms research, which is primarily theoretical. Although some material exists that discusses implementation-level aspects of the classical GHS algorithm [29], a combined theoretical and empirical evaluation approach has, to our knowledge, not been previously adopted in the literature of this field.

This deviation from standard practice also highlights the significance of our implementation of the baseline algorithm. As no publicly available implementations of the baseline algorithm existed, our implementation was crucial for our empirical evaluation and serves as a foundation for future work on applying these algorithms in practice.

5.5 Project Organisation

The project was planned around two deliverables, developing an asymptotically better algorithm and implementing constant-factor optimizations to an existing algorithm. An initial timeline with milestones structured the work, with an iterative approach for the primary goal and a sequential (waterfall) approach to the backup deliverable.

The application of planning to manage risks in the project was essential to its success. We had identified early on that being able to achieve our primary goal was uncertain and planned to work simultaneously on both deliverables. Despite the fact progress on the primary goal proved to be slow due to the time needed to build foundational knowledge about distributed algorithms as well as the unpredictable nature of theoretical research, progress towards the backup deliverable ensured that the project continued to advance.

This ensured that, despite being unable to achieve our primary goal, we had sufficient progress towards simulating the baseline algorithm to further support the implementation and empirical evaluation of the optimizations proposed in this project.

6 Conclusion

In this work, we presented optimizations to the awake-optimal distributed randomized MST algorithm proposed by Augustine, Moses Jr., and Pandurangan [1], for networks of small, constant diameter. Using the approach to empirical scaling analysis outlined by Hoos [14], we evaluated the empirical performance of our optimized variant on two classes of benchmark instances with diameter 3, including a carefully constructed set of adversarial instances.

Our results demonstrate that the optimized algorithm exhibits improved empirical scaling of round complexity compared to the original algorithm. A direct comparison of the total number of rounds observed indicates reductions by factors of approximately 50 for random instances and 12 for adversarial instances when $n = 446$. However, theoretical analysis confirms that the worst-case asymptotic round complexity of the optimized algorithm remains $O(n \log n)$, consistent with the original algorithm.

These improvements in round complexity come at the cost of a constant-factor increase in the empirical scaling of the algorithm's awake complexity of approximately 1.5 times.

Further analysis revealed that the improved round complexity is influenced by structural properties of the input graphs. In particular, the algorithm performs significantly better on random instances than on adversarial ones. A key factor is the growth of the maximum fragment depth across phases, as the length of each phase is bounded by this depth. Consequently, for instances where the maximum fragment depth grows as large as $O(n)$, the improvement in round complexity is diminished.

While the optimized algorithm was shown empirically to perform better for graphs of diameter 3, similar results

should be achievable for graphs of diameter 4 by extending the $Flood - Max$ procedure to four rounds.

Future work could investigate theoretically how structural properties of input graphs affect the growth of maximum fragment depth across phases. Theoretical analysis of this would allow for a more precise quantification of the optimizations benefits, as our current insights are based solely on empirical observations from a limited set of benchmark instances. Additionally, optimizing the design of our simulation environment to reduce runtime would enable us to scale our empirical studies to larger instance sizes, thereby providing stronger validation for the scaling models. Furthermore, our implementation of both the baseline and optimized algorithms as well as our simulation environment could serve as the foundation for further empirical studies and practical implementations of distributed algorithms.

REFERENCES

[1] J. Augustine, W. K. Moses Jr, and G. Pandurangan, "Awake complexity of distributed minimum spanning tree," in *International Colloquium on Structural Information and Communication Complexity*. Springer, 2024, pp. 45–63.

[2] C. Ambühl, "An optimal bound for the mst algorithm to compute energy efficient broadcast trees in wireless networks," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2005, pp. 1139–1150.

[3] G. Pandurangan, P. Robinson, M. Scquizzato *et al.*, "The distributed minimum spanning tree problem," *Bulletin of EATCS*, vol. 2, no. 125, 2018.

[4] G. Pandurangan, P. Robinson, and M. 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*, 2017, pp. 743–756.

[5] M. Elkin, "A simple deterministic distributed mst algorithm with near-optimal time and message complexities," *Journal of the ACM (JACM)*, vol. 67, no. 2, pp. 1–15, 2020.

[6] B. Haeupler, D. E. Hershkowitz, and D. Wajc, "Round-and message-optimal distributed graph algorithms," in *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, 2018, pp. 119–128.

[7] S. Chatterjee, R. Gmyr, and G. Pandurangan, "Sleeping is efficient: Mis in o (1)-rounds node-averaged awake complexity," in *Proceedings of the 39th Symposium on Principles of Distributed Computing*, 2020, pp. 99–108.

[8] L. Barenboim and T. Maimon, "Deterministic logarithmic completeness in the distributed sleeping model," *arXiv preprint arXiv:2108.01963*, 2021.

[9] Z. Lotker, E. Pavlov, B. Patt-Shamir, and D. Peleg, "Mst construction in o (log log n) communication rounds," in *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, 2003, pp. 94–100.

[10] Z. Lotker, B. Patt-Shamir, and D. Peleg, "Distributed mst for constant diameter graphs," in *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, 2001, pp. 63–71.

[11] H. H. Hoos and T. Stützle, "On the empirical scaling of run-time for finding optimal solutions to the travelling salesman problem," *European Journal of Operational Research*, vol. 238, no. 1, pp. 87–94, 2014.

[12] Z. Mu and H. H. Hoos, "On the empirical time complexity of random 3-sat at the phase transition." in *IJCAI*, 2015, pp. 367–373.

[13] Z. Mu, J. Dubois-Lacoste, H. H. Hoos, and T. Stützle, "On the empirical scaling of running time for finding optimal solutions to the tsp," *Journal of Heuristics*, vol. 24, pp. 879–898, 2018.

[14] H. H. Hoos, "A bootstrap approach to analysing the scaling of empirical run-time data with problem size," Technical report, TR-2009-16, Department of Computer Science, University of . . . , Tech. Rep., 2009.

[15] R. G. Gallager, P. A. Humblet, and P. M. Spira, "A distributed algorithm for minimum-weight spanning trees," *ACM Transactions on Programming Languages and systems (TOPLAS)*, vol. 5, no. 1, pp. 66–77, 1983.

[16] S. Kutten and D. Peleg, "Fast distributed construction of k-dominating sets and applications," in *Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing*, 1995, pp. 238–251.

[17] M. Elkin, "A faster distributed protocol for constructing a minimum spanning tree," *Journal of Computer and System Sciences*, vol. 72, no. 8, pp. 1282–1308, 2006.

[18] Z. Lotker, B. Patt-Shamir, E. Pavlov, and D. Peleg, "Minimum-weight spanning tree construction in o (log log n) communication rounds," *SIAM Journal on Computing*, vol. 35, no. 1, pp. 120–131, 2005.

[19] Y.-J. Chang, V. Dani, T. P. Hayes, Q. He, W. Li, and S. Pettie, "The energy complexity of broadcast," in *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, 2018, pp. 95–104.

[20] Y. Pushak and H. H. Hoos, "Advanced statistical analysis of empirical performance scaling," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 2020, pp. 236–244.

[21] A. J. Parkes, "Tuning local search for satis ity testing."

[22] H. H. Hoos and T. Stützle, "Local search algorithms for sat: An empirical evaluation," *Journal of Automated Reasoning*, vol. 24, no. 4, pp. 421–481, 2000.

[23] I. P. Gent, E. MacIntyre, P. Prosser, and T. Walsh, "The scaling of search cost," in *AAAI/IAAI*, 1997, pp. 315–320.

[24] S. F. Goldsmith, A. S. Aiken, and D. S. Wilkerson, "Measuring empirical computational complexity," in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2007, pp. 395–404.

[25] Z. Mu and H. H. Hoos, "Empirical scaling analyser: An automated system for empirical analysis of performance scaling," in *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, 2015, pp. 771–772.

[26] P. ERDdS and A. R&wi, "On random graphs i," *Publ. math. debrecen*, vol. 6, no. 290-297, p. 18, 1959.

[27] A. Hagberg, P. J. Swart, and D. A. Schult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Laboratory (LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[28] Y. Pushak, Z. Mu, and H. H. Hoos, "Empirical scaling analyzer (esa)," https://www.cs.ubc.ca/~w-esa/ESA/, accessed: 2025-05-01.

[29] Flysher and Rubinshtein, "A distributed algorithm for minimum weight spanning trees," [Online]. Available: https://raw.github.com/arjungmenon/Distributed-Graph-Algorithms/master/Minimum-Spanning-Tree/papers/GHS_enhanced.pdf (Accessed: April 29, 2025).