Message Optimality and Message-Time Trade-offs for APSP and Beyond

Fabien Dufoulon
Lancaster University
Lancaster, UK
f.dufoulon@lancaster.ac.uk

Shreyas Pai
IIT Madras
Chennai, India
shreyas@cse.iitm.ac.in

Gopal Pandurangan
University of Houston
Houston, USA
gopal@cs.uh.edu

Sriram V. Pemmaraju
University of Iowa
Iowa City, USA
sriram-pemmaraju@uiowa.edu

Peter Robinson
Augusta University
Augusta, USA
perobinson@augusta.edu

Abstract

Round complexity is an extensively studied metric of distributed algorithms. In contrast, our knowledge of the *message complexity* of distributed computing problems and its relationship (if any) with round complexity is still quite limited. To illustrate, for many fundamental distributed graph optimization problems such as (exact) diameter computation, All-Pairs Shortest Paths (APSP), Maximum Matching etc., while (near) round-optimal algorithms are known, message-optimal algorithms are hitherto unknown. More importantly, the existing round-optimal algorithms are not message-optimal. This raises two important questions: (1) Can we design message-optimal algorithms for these problems? (2) Can we give message-time tradeoffs for these problems in case the message-optimal algorithms are not round-optimal?

In this work, we focus on a fundamental graph optimization problem, *All Pairs Shortest Path (APSP)*, whose message complexity is still unresolved. We present two main results:

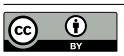
(1) An algorithm that solves weighted APSP, using $\tilde{O}(n^2)$ messages in $\tilde{O}(n^2)$ rounds. This algorithm is message-optimal (up to logarithmic factors) for algorithms that take poly(n) rounds.
(2) For any $0 \leq \varepsilon \leq 1$, we show how to solve unweighted APSP in $\tilde{O}(n^{2-\varepsilon})$ rounds and $\tilde{O}(n^{2+\varepsilon})$ messages. At one end of this smooth trade-off, we obtain a (nearly) message-optimal algorithm (for algorithms that take poly(n) rounds) using $\tilde{O}(n^2)$ messages (for $\varepsilon = 0$), whereas at the other end we get a (nearly) round-optimal algorithm using $\tilde{O}(n)$ rounds (for $\varepsilon = 1$).

CCS Concepts

• **Theory of computation → Distributed algorithms**.

Keywords

Distributed APSP, Message Complexity, Time-Message Tradeoffs, CONGEST, Network Decompositions



PODC '25, Huatulco, Mexico

ACM ISBN 979-8-4007-1885-4/2025/06
https://doi.org/10.1145/3732772.3733506

ACM Reference Format:
Fabien Dufoulon, Shreyas Pai, Gopal Pandurangan, Sriram V. Pemmaraju, and Peter Robinson. 2025. Message Optimality and Message-Time Trade-offs for APSP and Beyond. In *ACM Symposium on Principles of Distributed Computing (PODC '25), June 16–20, 2025, Huatulco, Mexico.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3732772.3733506

1 Introduction

Message and *time* complexities are two fundamental performance measures of distributed algorithms. Both complexity measures crucially influence the performance of a distributed algorithm. Time complexity measures the number of distributed "rounds" taken by the algorithm, and it directly determines the running time of the algorithm. Traditionally, keeping the time (round) complexity as small as possible has been an important goal. Message complexity, on the other hand, measures the *total number of messages* sent and received by all the processors during the algorithm. In many applications, message complexity is the dominant cost, playing a major role in determining the running time as well as additional resources (e.g., energy, bandwidth, memory, etc.) expended by the algorithm. Hence, designing distributed algorithms with low message complexity, sometimes even at the cost of an increased round complexity could be desirable.

Since both measures are essential, ideally, one would like to design distributed algorithms with *simultaneously* optimal message and time complexities — so-called *singularly optimal* algorithms. However, if this is too difficult or even impossible, then one would like to design distributed algorithms that are at least *separately* message or time optimal. One would like to go even further and design *a family of algorithms* that smoothly trades off between the two measures. However till now, while singularly-optimal algorithms are known for fundamental problems such as leader election [24] and minimum spanning trees [12, 29], there are other fundamental problems such as diameter computation or (even, unweighted) All-Pairs Shortest Paths (APSP) where not much is known with respect to message complexity vis-a-vis the round complexity.

This paper focuses on the message complexity of APSP, though some of our results apply to other problems such as Maximum Matching (MaxM) and computing Neighborhood Covers. APSP has been studied extensively for many decades in both sequential and distributed computing. Significant progress has been made towards designing distributed algorithms with near-optimal round

complexity for APSP in the classical CONGEST model of distributed computing (see Section 1.1 for model definitions). For example, after a series of improvements [13, 22], Bernstein and Nanongkai [5] presented an $\tilde{O}(n)$-round[1] randomized (Las Vegas) algorithm for weighted APSP. This is optimal within logarithmic factors because $\Omega(n)$ is a round complexity lower bound even for unweighted APSP [6]. However, the optimum attainable message complexity is unresolved for APSP. In particular, the Bernstein-Nanongkai [5] round-optimal distributed algorithm mentioned above, has $\tilde{\Theta}(mn)$ message complexity (throughout, m denotes the number of edges in the graph) which can be as large as $\tilde{\Theta}(n^3)$ for dense graphs. Our work is motivated by two key questions:

(1) Is $\tilde{\Theta}(n^3)$ the optimal (or near-optimal) message complexity for APSP or can we design distributed algorithms with significantly better (say, $\tilde{O}(n^2)$) message complexity? We note that [11] presents $\tilde{\Omega}(n^3)$ message lower bounds for poly(n)-round algorithms that produce exact solutions of graph optimization problems such as minimum vertex cover. We also note that using techniques from [11] applied to the lower bound graph in [1], one can show an $\tilde{\Omega}(n^2)$ message lower bound even for sparse graphs.
(2) Is it possible to trade off rounds for messages for APSP? Specifically, can we design APSP algorithms that are more message-frugal (relative to the $\tilde{\Theta}(n^3)$ message bound of [5]), but possibly use more rounds?

The thrust of the paper is to prove the two following theorems that answer the above questions. Our first result shows that indeed, it is possible to solve weighted APSP in the CONGEST model using $\tilde{O}(n^2)$ messages, showing that message optimality (within logarithmic factors) is achievable for weighted APSP. This result is a specific instance of a more general simulation result we prove (see Theorem 2.1) that shows that broadcast-based CONGEST algorithms that perform a total of B broadcast operations (across all nodes) can be simulated with a message complexity of $\tilde{O}(B)$. Our result is significant because there are many examples of distributed algorithms that are broadcast-based (e.g., Breadth First Search, Luby's algorithm for Maximal Independent Sets [26]) and the message complexity of these algorithms is typically quite a bit larger than number of broadcasts it performs. For example, the natural Breadth First Search (BFS) algorithm performs at most n broadcasts, but has $\Theta(n^2)$ message complexity in the worst case (for dense graphs). We also apply our simulation result to other problems, such as MaxM and Neighborhood Covers (see full version [10]).

Theorem 1.1. *There is a* CONGEST *algorithm that, with high probability, computes exact weighted APSP in $\tilde{O}(n^2)$ rounds and with $\tilde{O}(n^2)$ message complexity, even on directed graphs and even if the edge weights are negative.*

Our second result applies to unweighted APSP and shows that for this problem, there is a natural, smooth trade-off, parameterized by $\varepsilon \in [0, 1]$, between messages and rounds. At one end of this smooth trade-off, we obtain a (nearly) message-optimal algorithm using $\tilde{O}(n^2)$ messages (for $\varepsilon = 0$). For this point in the trade-off, the round complexity is $\tilde{O}(n^2)$ and this point in the trade-off can be viewed as a special case of Theorem 1.1. At the other end of the trade-off, we get a (nearly) round-optimal algorithm using $\tilde{O}(n)$ rounds (for $\varepsilon = 1$).

Theorem 1.2. *For any $\varepsilon \in [0, 1]$, unweighted APSP (on undirected graphs) can be solved (w.h.p.) in $\tilde{O}(n^{2-\varepsilon})$ rounds and $\tilde{O}(n^{2+\varepsilon})$ messages (w.h.p.) in* CONGEST.

We note that the results in this paper are for *exact* APSP. Allowing approximate solutions leads to very different bounds and message-time trade-offs. For example, by first building an $\Theta(\log n)$-stretch spanner and then solving APSP using just the spanner edges, one obtains an $O(\log n)$-approximation for APSP in $\tilde{O}(n)$ rounds and $\tilde{O}(n^2)$ messages. More detailed versions of our algorithms and all proofs of claims appear in the full version of our paper [10].

1.1 Model

1.1.1 CONGEST *model.* The CONGEST model [30] is a standard message-passing model in distributed computing where the input graph G defines the distributed network with the nodes denoting machines and the edges denoting communication links between two machines. Each node has a unique `ID` drawn from a space whose size is polynomial in n. We assume the *synchronous* model, where both computation and communication proceed in lockstep, i.e., in discrete time steps called *rounds*. We assume the standard KT_0 model, where each node has initial knowledge of only its ID. (Our results, in particular, the tightness of our message bounds, also apply to another well-studied model, namely the KT_1 model, where each node has initial knowledge of its ID and the IDs of its neighbors, under a mild restriction of algorithms running in only polynomial rounds.) In the CONGEST model, we allow only small message sizes (typically logarithmic in n, the number of nodes) to be sent per edge per round. Since each `ID` can be represented with $O(\log n)$ bits, each message in the CONGEST model is large enough to contain a constant number of `ID`s. The CONGEST model captures bandwidth limitations inherent in real-world networks. In each round of the synchronous CONGEST model, each node (i) receives all messages sent to it in the previous round, (ii) performs arbitrary local computation based on information it has[2], and (iii) sends a $O(\log n)$-bit message to each of its neighbors in the graph. Note that a node may send *different messages* to *different neighbors* in a round. For an algorithm $\mathcal{A}$ in the synchronous CONGEST model, its *round complexity* is the number of rounds it takes to finish and produce output and its *message complexity* is the total number of messages sent by all nodes over the course of the algorithm. For brevity we drop "synchronous" and just call this the CONGEST model.

1.1.2 BCONGEST *model.* The BCONGEST model differs from the CONGEST model only in that in each round, a node must send the *same* message to all of its neighbors. Just as in the CONGEST model, we define the *message complexity* of a BCONGEST algorithm as the number of messages sent by all nodes over the entire execution. However, we also introduce a related complexity measure, which we call *broadcast complexity*: it is the number of broadcasts by

[1] $\tilde{\Omega}$ and $\tilde{O}$ hide a 1/polylog n and polylog n factor respectively.

[2] As is standard, the cost of local computation is ignored and, in each round, each node can perform an arbitrary, even exponential time, computation, using the information it possesses. This assumption is justified by the fact that communication cost dominates local computation cost substantially in many settings.

all nodes over the entire execution. In this paper, we show that low broadcast complexity plays an important role in designing message efficient CONGEST algorithms. In particular, the broadcast complexity bounds the cost of simulating a BCONGEST algorithm message efficiently in the CONGEST model.

1.2 Our Technical Contributions

Our contributions are two-fold. We present the first message-optimal algorithm for weighted APSP (Theorem 1.1) and the first time-message trade-off for unweighted APSP (Theorem 1.2). We now describe our approach, at a high level, for obtaining these results.

Message-Optimal Algorithms. In a nutshell, our approach for message-optimal algorithms uses the following general idea. First, suppose that we are able to partition the graph into clusters such that (i) each cluster has low diameter and (ii) each vertex has neighbors in only a small number of clusters. Note that requirement (ii) permits vertices to have high degree. Now consider an algorithm $\mathcal{A}$ in the BCONGEST model with low broadcast complexity. Our key observation is this: *whenever a vertex v performs a broadcast, instead of sending a message to all neighbors of v, we can send a message to one representative neighbor for each of the neighboring clusters.* The fact that every vertex has only a small number of neighboring clusters implies that the number of messages sent is small. The fact that the diameter of each cluster is small implies that it is possible for the center of each cluster to efficiently (in terms of messages) serve as a proxy for all nodes in the cluster and thus it is enough for v's message to reach a single node in each neighboring cluster. A key ingredient of our approach is the notion of a *Low Diameter and Communication (LDC) graph decomposition.* A LDC decomposition structure allows us to pre-process the graph to obtain the clustering with properties described above. This decomposition is a simple variant of the low-diameter graph decomposition algorithm of Miller, Peng, and Xu [27]. Using this approach, we show how to obtain an equivalent CONGEST algorithm whose message complexity is within polylogarithmic factors of the simulated algorithm's broadcast complexity at the cost of a higher time complexity (roughly, by a linear in n factor) — cf. Theorem 2.1.

Message-Time Tradeoffs. While the above approach yields a message-optimal algorithm for weighted APSP, this algorithm has a high round complexity — $\tilde{O}(n^2)$ — for unweighted APSP. This is in fact, an $O(n)$ factor higher than the round optimal algorithm known for unweighted APSP, as discussed above. On the other hand, as mentioned earlier, known round-optimal algorithms are not message optimal; they have a $\Theta(n^3)$ message complexity. The LDC decomposition framework, discussed above, that led to message optimality does not help obtain a trade-off between these two extremes. We need a new framework and our main technical contribution in this paper is the use of several new and disparate ideas that lead to such a framework. Below, we overview these ideas and how they lead to our family of algorithms.

We start with a natural idea: solve unweighted APSP by simulating a collection of n Breadth First Search (BFS) algorithms $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$, initiated at each of the n different nodes.

Baswana-Sen cluster hierarchy. In order to control the message complexity of the collection of BFS algorithms, we simulate them over a *Baswana-Sen cluster hierarchy*. This hierarchy of clusters was defined by Baswana and Sen [3] as part of their classical randomized spanner algorithm. Baswana and Sen present an algorithm that, for any input graph $G = (V, E)$ and any integer $\kappa \geq 1$, constructs a $(2\kappa - 1)$-spanner $H = (V, E_H)$ of G with $O(n^{1+1/\kappa})$ edges. While the final output of this algorithm is a spanning subgraph H of G, a byproduct of this algorithm is a $(\kappa + 1)$-level hierarchy of clusters (defined precisely in Section 3.1). It is this Baswana-Sen cluster hierarchy that we utilize for our simulation. Given an $\varepsilon \in [0, 1]$, setting $\kappa = \lceil 1/\varepsilon \rceil$ and computing a $(\kappa + 1)$-level hierarchy of clusters, gives us, roughly speaking, the property that each node has neighbors in $\tilde{O}(n^{\varepsilon})$ clusters. We can then use the same idea as in our previous simulation, i.e., whenever a vertex v performs a broadcast, instead of sending a message to all neighbors of v, we can send a message to one representative neighbor for each of the neighboring clusters.

"Congestion + Dilation" framework. A classical result of Leighton, Maggs, and Rao [25] showed that if we are given ℓ packets, where packet p_j needs to be routed from a given source s_j to a given target t_j along a given path $\mathcal{P}_j$, then all of these packets can be scheduled, so as to take just $\tilde{O}(\text{congestion}+\text{dilation})$ rounds, where congestion is the maximum number of packets that are routed through an edge and dilation is the length of the longest path. An extremely elegant idea of using *random delays* to start the different packets, leads to their result. Ghaffari [15] extended this result to a collection of arbitrary CONGEST algorithms (see Theorem 1.3). In order to bound the round complexity of our collection of BFSs using this "Congestion + Dilation" result, we need an efficient simulation over a Baswana-Sen cluster hierarchy with good bounds on both the maximum congestion of an edge and the dilation (i.e., maximum running time) of the BFS algorithms. In order to obtain an efficient simulation, we need to schedule our collection of BFS algorithms to have an additional property, namely, in any round, each node receives messages from at most $O(\log n)$ distinct BFS algorithms. We show (in Theorem 1.4) that applying the random delay technique to a collection of BFS algorithms, not only gives the $\tilde{O}(\text{congestion} + \text{dilation})$ bound on the running time, but also this additional property. This additional property allows us to view our collection of algorithms as an *aggregation-based algorithm* and plays a crucial role in ensuring the simulation is both message and round efficient.

Simulating aggregation-based algorithms. A key reason for the efficiency of our simulation is the fact that we are simulating *aggregation-based algorithms* (see Definition 3.1). To understand the definition of an aggregation-based algorithm, imagine that messages to a node v at a particular round r, arrive in separate batches. In an aggregation-based algorithm, the next local state of v, can be computed by processing each batch separately, with all intermediate computation results being small in size. Algorithms that use functions such as min, max, sum, etc. fall in this category and for such functions it is clear that messages can be batched and intermediate results are small. Individual BFS algorithms also have the nice property of being aggregation-based. However, a collection of BFS algorithms may not have this property because each batch of messages may contain information about many different BFSs and

this information cannot be aggregated to have a small size. This is where the additional property of BFS scheduling mentioned in the previous item ("Congestion + Dilation" framework) turns out to be useful.

In the new simulation built upon the Baswana-Sen cluster hierarchy, a cluster can no longer maintain complete knowledge of all its cluster nodes' states efficiently, as a node may belong to multiple clusters across the levels of the cluster hierarchy. Therefore, instead, each cluster (center) routes messages on behalf of its (non-unique) cluster nodes. But this can lead to a significant increases in both runtime and message overheads, where the former comes about due to high congestion over certain edges of the cluster hierarchy. To remedy that, we restrict the simulation to aggregation-based algorithms, which now allows each cluster to compress all the messages it routes to a single destination into $\tilde{O}(1)$ bits, and logarithmically as many messages. This leads to better congestion bounds, and thus good runtime overhead, as well as better message complexity, through the following argument: informally, each node only needs to receive one message per cluster in its neighborhood, which is, roughly speaking, bounded to $O(n^{\varepsilon})$ within a cluster hierarchy.

Smoothing congestion using an ensemble of Baswana-Sen cluster hierarchies. Using just a single Baswana-Sen cluster hierarchy leads to excessive congestion on a few edges and very little congestion on many edges. A key idea to bypass this congestion bottleneck is to use not just one, but a small number $\zeta = n^{\varepsilon}$ of independently constructed Baswana-Sen cluster hierarchies. Then the set of n BFS algorithms are partitioned into ζ equal-sized batches and each batch can be assigned a different Baswana-Sen cluster hierarchy. This simple and natural idea plays a crucial role in reducing the maximum congestion of an edge (see Lemma 3.8). A key reason why this smoothing property holds is that the Baswana-Sen cluster hierarchy has the helpful property that any edge in the input graph has a small probability of being chosen as a cluster edge (see Lemma 3.7).

Using "landmark" nodes. Even with the ideas described above, we do not completely achieve the trade-off we seek. Specifically, the dilation of our simulated BFS algorithms is too high. One final idea we use is to terminate the n BFSs at a depth of $O(n^{1-\varepsilon})$. This allows the BFSs to discover short paths, i.e., shortest paths of length $O(n^{1-\varepsilon})$. But this leaves us with all pairs of nodes that are further apart from each other. However, for these, we can take a more brute force approach based on the idea of sampling "landmark" nodes, that will generate BFS trees containing all the remaining shortest paths (i.e., between any two far away nodes).

1.3 Additional Related Work

The BCONGEST model is used in this paper mainly as a vehicle for obtaining a message-efficient simulation for algorithms in the CONGEST model. However, there are a few papers that focus on designing algorithms and proving lower bounds in the BCONGEST model [4, 7–9, 20, 23, 28].

There has been a lot of research on APSP, but from a round complexity point of view. For example, for the problem of computing the diameter, an $\Omega(n/\log n)$ lower bound is shown in [14], even for graphs with constant diameter. A matching $O(n/\log n)$ upper bound is proved for this problem in [21]. For the more general, APSP problem the authors of [2] show an $\Omega(n)$ lower bound for weighted APSP, thus separating its round complexity from unweighted APSP by at least a logarithmic factor. In a breakthrough result, Bernstein and Nanongkai shows that this lower bound can be almost-exactly matched, by presenting a Las Vegas $\tilde{O}(n)$-round algorithm for weighted APSP [5].

The work of [18] studies time-message tradeoffs for distributed algorithms in the KT_1 model for various fundamental problems.

The work of [16] gives a low-energy algorithm for the Single-Source Shortest Paths (SSSP) problem by restricting the amount of congestion per node, which can be helpful for the design of message-efficient algorithms.

Aggregation operations that are helpful to obtain our message-time tradeoffs have been central to many efficient distributed algorithms, see, e.g., [17, 19, 31].

1.4 The Congestion Plus Dilation Framework

Consider a scenario in which we want to run ℓ independent distributed algorithms $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_\ell$ together in the CONGEST model. Following notation in [15], we use dilation to refer to the maximum running time (number of rounds) of any of the algorithms $\mathcal{A}_j$, when executed in isolation. For any edge e, let $c_j(e)$ be the number of rounds in which algorithm $\mathcal{A}_j$ sends a message over e and let $\text{congestion}(e) = \sum_{j=1}^{\ell} c_j(e)$. Thus $\text{congestion}(e)$ denotes the total number of messages sent over edge e over all ℓ algorithms $\mathcal{A}_j$. Finally, let $\text{congestion} = \max_e \text{congestion}(e)$.

We will make repeated use of the following result due to Ghaffari [15] that shows that the ℓ algorithms can be scheduled in such a way that it takes essentially, only congestion + dilation rounds for all the algorithms to complete. This result extends the classical result of Leighton, Maggs, and Rao [25] that applied to the special case in which each algorithm $\mathcal{A}_j$ performed a routing task, routing a packet from a given source s_j to a given target t_j along a pre-specified path $\mathcal{P}_j$.

Theorem 1.3 (Ghaffari [15]). *It is possible to schedule algorithms $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_\ell$ together in the* CONGEST *model, using only private randomness such that, with high probability, all the algorithms complete in $O(\text{congestion} + \text{dilation} \cdot \log n)$ rounds, after $O(\text{dilation} \cdot \log^2 n)$ rounds of pre-computation.*

We also make use of a version of Theorem 1.3 that applies to scheduling a collection of *partial BFS* algorithms (see Theorem 1.4 below). By a partial BFS algorithm, we just mean a BFS algorithm that, by design, might terminate early, even before it explores the entire graph. Our theorem assumes that we are dealing with the standard BFS algorithms, i.e., the algorithm in which each node broadcasts just once, on first receiving a "BFS exploration" message. Our theorem applies in the BCONGEST model, i.e., the final algorithm, which consists of an efficient scheduling of the ℓ partial BFS algorithms, is also in the BCONGEST. This scheduling also has a useful additional property that we prove, which is that in any round, at most $O(\log n)$ algorithms (out of ℓ) are "active" in a neighborhood.

Theorem 1.4. *Consider a collection of $\ell \leq n$ BFS algorithms $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_\ell$, each initiated by a different node. Then it is possible*

to schedule all the BFS algorithms together in the BCONGEST *model, using private randomness such that, with high probability, (i) all the algorithms complete in $\tilde{O}(\ell + \text{dilation})$ rounds and (ii) every node receives messages from at most $O(\log n)$ distinct BFS algorithms in any round.*

2 Message-Efficient Simulation of BCONGEST Algorithms

In this section, we show how we can message-efficiently simulate BCONGEST algorithms whose message complexity is significantly higher than their broadcast complexity. More formally, we show how to obtain an equivalent CONGEST algorithm whose message complexity is within polylogarithmic factors of the simulated algorithm's broadcast complexity, at the cost of a higher time complexity (roughly, by a linear in n factor) — see Theorem 2.1 below.

Theorem 2.1. *Let $\mathcal{A}$ be any* BCONGEST *algorithm with $T_{\mathcal{A}}$ round complexity, $B_{\mathcal{A}}$ broadcast complexity, and let In and Out denote respectively the size of the inputs (including communication graph and IDs) and outputs over all nodes. Then, there exists a randomized (Monte Carlo)* CONGEST *algorithm $\mathcal{A}'$ simulating $\mathcal{A}$ with round complexity $\tilde{O}(In + Out + T_{\mathcal{A}}\, n)$ and message complexity $\tilde{O}(In + Out + B_{\mathcal{A}})$, with high probability.*

Note that by definition $B_{\mathcal{A}} \leq T_{\mathcal{A}}\, n$ for any BCONGEST algorithm $\mathcal{A}$, and thus we can get the following simpler statement as a corollary of Theorem 2.1.

Corollary 2.2. *Let $\mathcal{A}$ be any* BCONGEST *algorithm with $T_{\mathcal{A}}$ round complexity and let In and Out denote respectively the size of the inputs (including communication graph and IDs) and outputs over all nodes. Then, there exists a randomized (Monte Carlo)* CONGEST *algorithm $\mathcal{A}'$ simulating $\mathcal{A}$ with round and message complexity $\tilde{O}(In + Out + T_{\mathcal{A}}\, n)$, with high probability.*

Low Diameter and Communication (LDC) Graph Decomposition. For the sake of improving message complexity, we introduce low diameter and communication (LDC) graph decompositions. Informally, such decompositions also bound the number of edges going out of any one node of a cluster into all neighboring clusters.

Definition 2.3 (Low Diameter and Communication (LDC) Graph Decomposition)**.** *Let $G = (V, E)$ be an unweighted graph. A (r, d)-low diameter and communication decomposition of G is a partition of the vertex set V into subsets $V_1, \ldots, V_k$, called clusters, combined with a sparse inter-cluster communication directed edge set $F \subseteq E$ such that:*

- *Each cluster V_i has strong diameter at most r, i.e., for any two nodes $u, v \in V_i$, $dist_{G[V_i]}(u, v) \leq r$,*
- *Each node $v \in V_i$ has at most d (outgoing) incident edges in F, such that for each cluster V_j containing a neighbor of v, there exists at least one incident edge $e = (v, u) \in F$ with $u \in V_j$.*

Next, we show that a low diameter and low communication graph decomposition can be obtained with some simple additions to the low diameter graph decomposition algorithm of Miller, Peng, and Xu [27] which is popularly referred to as the MPX algorithm.

Lemma 2.4. *There exists an $O(\log n)$ round algorithm that produces an $(O(\log n), O(\log n))$ low diameter and communication graph decomposition. Moreover, each cluster of that decomposition is spanned by a tree of depth $O(\log(n))$.*

Description of the Simulation. Let $\mathcal{A}$ denote an arbitrary BCONGEST algorithm with $T_{\mathcal{A}}$ round complexity and $B_{\mathcal{A}}$ broadcast complexity, and let *In* and *Out* denote respectively the size of the inputs (including communication graph and IDs) and outputs over all nodes. Then, we give a CONGEST algorithm $\mathcal{A}'$ that simulates $\mathcal{A}$ (i.e., produces the same output). The algorithm $\mathcal{A}'$ can be decomposed into two parts: a preprocessing part, and a simulation part. In the preprocessing part, we compute a (low diameter and communication) clustering of the communication graph which will be crucial in limiting the message complexity of the simulation part to roughly the broadcast complexity of the simulated algorithm $\mathcal{A}$. The simulation part of $\mathcal{A}'$ is broken up into $T_{\mathcal{A}}$ phases. Each phase $p \in [1, T_{\mathcal{A}}]$ of $\mathcal{A}'$ is used to simulate one round (round p) of $\mathcal{A}$. Throughout most of these phases, the cluster centers take care of the simulation whereas non-center cluster nodes simply transmit information between cluster centers. However in the final phase, cluster centers inform their cluster's nodes of their outputs. Somewhat more formally, each phase $p \in [1, T_{\mathcal{A}}]$ satisfies the following invariant: at the start of phase p, each cluster center c_i knows the state of each of its cluster's node $v \in V_i$ at the start of round p in the simulated algorithm $\mathcal{A}$. Moreover, at the end of the last phase $T_{\mathcal{A}}$, all nodes know their state at the end of round $T_{\mathcal{A}}$ in $\mathcal{A}$. We refer to the full version of the paper [10] for a detailed simulation.

Applications of this Simulation. As a key application, we consider the APSP problem on weighted graphs, where each edge is labeled with a weight chosen from a range that is polynomial in n. The goal is for each node to output its distance to all other nodes. The work of [5] presents a randomized Las Vegas algorithm that computes the (exact) APSP problem in $\tilde{O}(n)$ rounds of the BCONGEST model. An immediate application of Corollary 2.2 yields the following:

Theorem 1.1. *There is a* CONGEST *algorithm that, with high probability, computes exact weighted APSP in $\tilde{O}(n^2)$ rounds and with $\tilde{O}(n^2)$ message complexity, even on directed graphs and even if the edge weights are negative.*

In this paper, we mostly focus on the APSP problem, and in particular on its unweighted version. However, the simulation given in this section is especially general, and leads to message-efficient algorithms for other key problems such as (1) exact bipartite maximum matching, and (2) neighborhood covers (in particular, those with small $o(\log n)$ radius) — see the full version [10] for more details.

3 Message-Time Trade-offs for APSP

In Section 2, we show that message-optimality — more precisely, $\tilde{O}(n^2)$ message complexity — is achievable for weighted APSP, and several other problems. However, this comes at the cost of a high and clearly non-optimal round complexity of $\tilde{O}(n^2)$. On the other hand, round-optimal (i.e, with $\tilde{O}(n)$ runtime) algorithms for weighted APSP exist (e.g., [5]) but they are not message-optimal — they have $\tilde{O}(n^3)$ message complexity. In this section, we provide a

distributed CONGEST algorithm achieving a message-time trade-off for unweighted APSP that provides a natural, linear trade-off between the above mentioned two points, i.e., ($\tilde{O}(n^2)$ rounds, $\tilde{O}(n^2)$ messages) at one end and ($\tilde{O}(n)$ rounds, $\tilde{O}(n^3)$ messages) at the other:

Theorem 1.2. *For any $\varepsilon \in [0,1]$, unweighted APSP (on undirected graphs) can be solved (w.h.p.) in $\tilde{O}(n^{2-\varepsilon})$ rounds and $\tilde{O}(n^{2+\varepsilon})$ messages (w.h.p.) in* CONGEST.

Recall that we obtained the results in Section 2 by presenting a general simulation theorem for BCONGEST algorithms. Here, Theorem 1.2 is also obtained via a simulation, but this simulation applies to BCONGEST algorithms that satisfy an additional constraint: that is, the algorithm should be *aggregation-based*, in the sense that there is an aggregation function that, intuitively speaking, allows us to replace any collection of messages addressed to the same destination by an equivalent collection of messages that have a length of only $\tilde{O}(1)$ bits:

Definition 3.1 (Aggregation-based Algorithm)**.** *Consider a* BCONGEST *algorithm $\mathcal{A}$. For an arbitrary node v and an arbitrary round r, let $\mathbb{M}_{v,r}$ denote the set of possible messages that v may receive in round r, let $\mathcal{P}(\mathbb{M}_{v,r})$ denote its power set, and let $f_{v,r}$ denote the local function computed by node v at the end of round r. Algorithm $\mathcal{A}$ is an aggregation-based algorithm if, for all nodes v and rounds r, there exists a function $agg_{v,r} : \mathcal{P}(\mathbb{M}_{v,r}) \to \mathcal{P}(\mathbb{M}_{v,r})$ such that, for any subset $M \subseteq \mathbb{M}_{v,r}$:*

- *$agg_{v,r}(M)$ can be represented in $\tilde{O}(1)$ bits,*[3] *and*
- *for any partition $(M'_1, \ldots, M'_k)$ of M, it holds that*

$$f_{v,r}(state_v, M) = f_{v,r}(state_v, \bigcup_{i=1}^{k} agg_{v,r}(M_i)).$$

To take full advantage of the message improvements given by the new simulation, we consider BCONGEST algorithms that we call ℓ-decomposable:

Definition 3.2. *An ℓ-decomposable algorithm $\mathcal{A}$ consists of ℓ completely independent* BCONGEST *algorithms $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_\ell$ that we call components. Every node v locally computes its final output from its ℓ outputs, one from each of the ℓ components.*

We now give an overview of the underlying techniques of our simulation. The basis for our simulation are the cluster hierarchies defined by Baswana and Sen [3] as part of their classical randomized spanner algorithm. Baswana and Sen present an algorithm that, for any input graph $G = (V, E)$ and any integer $\kappa \geq 1$, constructs a $(2\kappa-1)$-spanner $H = (V, E_H)$ of G with $O(n^{1+1/\kappa})$ edges. While the final output of this algorithm is a spanning subgraph H of G, a byproduct of this algorithm is a $(\kappa+1)$-level hierarchy of clusters (to be defined precisely in the next subsection). It is this Baswana-Sen cluster hierarchy that we utilize for our simulation. However, using a single Baswana-Sen cluster hierarchy leads to excessive congestion on a few edges and very little congestion on many edges. A key idea to bypass this congestion bottleneck is to use not one, but a small number ζ of independently constructed Baswana-Sen cluster hierarchies. Then the set of ℓ components of an ℓ-decomposable algorithm $\mathcal{A}$ can be partitioned into ζ equal-sized batches and each batch can be assigned a different Baswana-Sen cluster hierarchy. This simple and natural idea plays a crucial role in reducing the maximum congestion of an edge and allows us to appeal to the congestion plus dilation framework (see Section 1.4) to bound the round complexity of our simulation. One additional requirement of our simulation is that each proper subtree of a cluster in the Baswana-Sen cluster hierarchy needs to have a relatively small number of nodes. To ensure this we need an additional pruning phase. The final product is an *ensemble of pruned Baswana-Sen cluster hierarchies* that we then use in our simulation.

[3]Note that $agg_{v,r}(M)$ returns a subset of the messages of M rather than just a single message.

3.1 Ensemble of Pruned Baswana-Sen Cluster Hierarchies

Let $\varepsilon \in [\frac{1}{\Theta(\log n)}, 1]$ be some input parameter. Let $\kappa := \lceil 1/\varepsilon \rceil$. First, we compute a sequence of $\kappa + 1$ subsets of vertices $S_0, S_1, \ldots, S_\kappa$ via random sampling as follows. We define $S_0 = V$, $S_\kappa = \emptyset$, and for any $i \in [1, \kappa - 1]$, S_i is obtained by sampling nodes independently from S_{i-1} with probability $n^{-\varepsilon}$. These subsets will form centers of clusters defined below.

Second, for each level $i = 0, 1, \ldots, \kappa$, we compute three objects: (1) *clustering* C_i; each clustering C_i being a collection of vertex-disjoint clusters, (2) *low-degree vertex subset* L_i, and (3) *inter-cluster communication edge subset* F_i. The sequence $(C_i, L_i, F_i)_{i=0}^{\kappa}$ consisting of $\kappa + 1$ levels defines a *Baswana-Sen cluster hierarchy*. We now describe its construction in detail.

- **Bottom level:** At the bottom level, i.e., level $i = 0$, the clustering $C_0 := \{\{v\} \mid v \in V\}$. Furthermore, the low-degree vertex subset L_0 and the inter-cluster communication edge subset F_0 are both empty. For any cluster $C = \{v\}$ in C_0, we designate v as the *center* of cluster C. Note that this means that the vertices in S_0 form the centers of clusters in C_0. Each cluster $C \in C_0$ can be viewed as (trivial) rooted tree. We use V_0 to denote the set of vertices that belong to clusters in C_0. Clearly, $V_0 = V$. More generally, we will use V_i to denote the vertices that belong to clusters in C_i.
- **Levels $i = 1, 2, \ldots, \kappa - 1$:** We obtain the clustering C_{i+1}, low-degree vertex subset L_{i+1}, and inter-cluster communication edge subset F_{i+1} for level $i + 1$, $0 \leq i \leq \kappa - 2$, from level i as follows. Let $\mathcal{R}_i \subseteq C_i$ denote the subset of clusters whose centers are in S_{i+1}; we call these *sampled i-clusters*. Consider each non-sampled i-cluster C, i.e., a cluster $C \in C_i \setminus \mathcal{R}_i$ and each node $v \in C$. There are two cases depending on the neighborhood of v.
  - (i) v **has a neighbor that belongs to a sampled i-cluster.** Then v joins an arbitrary sampled i-cluster in its neighborhood, via an edge to some arbitrary neighbor in that cluster. The edge along which v joins the neighboring sampled i-cluster is called a *cluster edge* and is denoted by $parent(v, i + 1)$. This process results in the growth of clusters in $\mathcal{R}_i$; this set of sampled i-clusters, along with their newly-joined nodes, forms the new clustering C_{i+1}. Note that each cluster in C_{i+1} can be viewed as a rooted tree, rooted at a center in S_{i+1}.
  - (ii) v **does not have a neighbor that belongs to a sampled i-cluster.** Then node v is added to L_{i+1}. Furthermore, for each neighboring cluster in C_i, one (undirected) edge from v to some (arbitrary) vertex in that cluster is added to F_{i+1}. Note that the vertex subsets L_{i+1} and V_{i+1} form a partition of V_i.

- **Top level:** At the top level, $C_\kappa = \emptyset$ and thus $V_\kappa = \emptyset$. We set $L_\kappa = V_{\kappa-1}$ and for each $v \in L_\kappa$ and each neighboring cluster $C \in C_{\kappa-1}$, we add one (undirected) edge from v to some vertex in C to F_κ.

The set of all *inter-cluster communication edges* F is simply $\cup_{i=0}^{\kappa} F_i$. This completes the construction of the Baswana-Sen cluster hierarchy. The following theorem states well-known properties of the Baswana-Sen cluster hierarchy [3].

Theorem 3.3. *The Baswana-Sen cluster hierarchy $(C_i, L_i, F_i)_{i=0}^{\kappa}$ has the following properties:*

(a) For any level $i \in [0, \kappa-1]$, the clustering C_i is a collection of disjoint clusters of V_i with (strong) radius i.
(b) The following statement holds with high probability: for any level $i \in [1, \kappa]$, for any node $v \in L_i$, there are at most $O(n^\epsilon \log n)$ edges in F_i incident on v, each connecting v to a distinct cluster in C_{i-1}.
(c) Consider an edge $(u, v) \in E$ and suppose that $u \in L_i$ and $v \in L_j$ for $i \leq j$. Then either (1) some cluster in C_{i-1} contains both u and v, or (2) there exists some inter-cluster communication edge $e = (u, w) \in F_i$ and some cluster in C_{i-1} containing both w and v.

The construction of the Baswana-Sen cluster hierarchy described above can be implemented in a straightforward manner in the CONGEST model, leading to the following well-known theorem.

Theorem 3.4. *Let $G = (V, E)$ be an unweighted graph. Then, there exists a* CONGEST *algorithm running in $O(\kappa)$ rounds and using $O(\kappa \cdot m)$ messages for computing a Baswana-Sen cluster hierarchy with $\kappa + 1$ levels.*

Pruning the clusters. In order to use the Baswana-Sen cluster hierarchy for efficient simulation, we need a pruning step to avoid clusters with large subtrees. In fact, clusters with large subtrees suffer from high maximum edge congestion in the simulation. More concretely, we want to avoid having any clusters in which any proper subtree has $\omega(n^{1-\varepsilon})$ nodes.

We will now describe how the pruned Baswana-Sen hierarchy $(C_i^*, L_i, F_i^*)_{i=0}^{\kappa}$ is obtained from the Baswana-Sen hierarchy $(C_i, L_i, F_i)_{i=0}^{\kappa}$. This pruning step involves (i) pruning each cluster at each level, if necessary and (ii) adding inter-cluster communication edges.

- **Pruning clusters:** Consider each level $i \in [1, \kappa-1]$ (for levels 0 and κ, pruning is not required) and each cluster $C \in C_i$. If every proper subtree in C contains fewer than $n^{1-\varepsilon}$ nodes, then there is no need to prune C. Otherwise, we repeatedly look for the deepest node, say u^*, such that the subtree rooted at u^* contains at least $n^{1-\varepsilon}$ nodes. We then split off the subtree rooted in u^* into its own cluster. Note that this can happen at most $O(n^\epsilon)$ times at a particular level i, by a simple counting argument over the n nodes, thereby adding at most $O(n^\epsilon)$ clusters to the existing clustering C_i. Once there exists no such node for all clusters in a clustering C_i, we have the new level-i clustering C_i^*. Note that while this pruning procedure is described as a seemingly sequential algorithm, all clusters within a clustering C_i at level i can be processed in parallel using upcast, leading to an algorithm that runs in $O(\kappa^2)$ rounds, using $O(\kappa \cdot n)$ messages.
- **Adding inter-cluster communication edges:** Consider some level $i \in [1, \kappa]$. (For level 0, F_0^* is empty, similarly to F_0.) Then, for each node $v \in L_i$ and for each neighboring cluster $C \in C_{i-1}^*$, we add to F_i^* one edge from v to some node C. Because v has only $O(n^\epsilon)$ neighboring clusters in C_{i-1}, and only $O(n^\epsilon)$ new clusters are created per clustering, there are only $O(n^\epsilon)$ edges incident to v in F_i^*.

We call the result *pruned Baswana-Sen cluster hierarchy*.

The following two corollaries are versions of Theorems 3.3 and 3.4 that apply to pruned Baswana-Sen cluster hierarchies.

Corollary 3.5. *Properties (a), (b), and (c) in Theorem 3.3 also hold for a pruned Baswana-Sen cluster hierarchy. In addition, each proper subtree of a cluster in a pruned Baswana-Sen cluster hierarchy contains at most $O(n^{1-\epsilon})$ nodes.*

Corollary 3.6. *Let $G = (V, E)$ be an unweighted graph. Then, there exists a* CONGEST *algorithm running in $O(\kappa^2)$ rounds and using $O(\kappa \cdot m)$ messages for computing a pruned Baswana-Sen cluster hierarchy with $\kappa + 1$ levels.*

Ensemble of pruned Baswana-Sen cluster hierarchies. Instead of using one pruned Baswana-Sen cluster hierarchy to simulate our algorithm, we use multiple pruned Baswana-Sen cluster hierarchies with each cluster hierarchy responsible for simulating some number of components of an ℓ-decomposable algorithm. The motivation for doing this is to "smooth" out congestion on cluster edges, i.e., instead of a few cluster edges (of a particular cluster hierarchy) having extremely high congestion, we end up with many more cluster edges, each having lower congestion. Recall that by "cluster edge" we mean an edge of a rooted tree that forms a cluster. This idea crucially relies on the following lemma that shows that the likelihood of an edge ending up as a cluster edge in a pruned Baswana-Sen cluster hierarchy is quite small.

Lemma 3.7. *For any edge e of the input graph G, e is a cluster edge of a pruned Baswana-Sen cluster hierarchy with probability $O(\kappa \cdot n^{-\varepsilon})$.*

Let $\mathcal{A}$ be an ℓ-decomposable BCONGEST algorithm and let $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_\ell$ be its ℓ components. Suppose that each algorithm $\mathcal{A}_j$ takes as input a pruned Baswana-Sen cluster hierarchy and uses it in its execution. In other words, messages sent by each algorithm $\mathcal{A}_j$ are sent only along the cluster edges and the inter-cluster edges of the given pruned Baswana-Sen cluster hierarchy $(C_i, L_i, F_i)_{i=0}^{\kappa}$. Let congestion$_j$ denote the maximum number of messages that pass over a cluster edge, in the worst case, during the execution of $\mathcal{A}_j$. Here, the "worst case" is over all possible inputs, including all possible pruned Baswana-Sen cluster hierarchies. Let congestion $= \max_j$ congestion$_j$. Now suppose that we execute all ℓ algorithms together, using a single pruned Baswana-Sen cluster hierarchy $\mathcal{H} = (C_i, L_i, F_i)_{i=0}^{\kappa}$ as input to all ℓ algorithms. Then it is possible that in the worst case there is a cluster edge of $\mathcal{H}$ over which $\ell \cdot$ congestion messages pass. This happens if the same cluster edge e of $\mathcal{H}$ sees the maximum number of messages passing over it, in all of the ℓ algorithms. We now show that we can spread out this congestion across many different edges by simply having the algorithms use an ensemble of pruned Baswana-Sen cluster hierarchies, instead of a single hierarchy.

Lemma 3.8 (Congestion Smoothing Lemma). *Let $\zeta = \lceil n^{\varepsilon} \rceil$ and consider an ensemble of ζ independently constructed, pruned Baswana-Sen cluster hierarchies $\{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_\zeta\}$. Arbitrarily partition the set of ℓ algorithms that are the components of $\mathcal{A}$ into ζ equal-sized batches $B_1, B_2, \ldots, B_\zeta$ and provide each pruned Baswana-Sen cluster hierarchy $\mathcal{H}_j$ as input to all the algorithms in batch B_j. If we execute the components $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_\ell$ with this input distribution, then with high probability, the maximum number of messages that pass over any cluster edge over the course of algorithm $\mathcal{A}$ is $\tilde{O}\left(\frac{\text{congestion}\cdot\ell}{\zeta}\right)$.*

Although the above lemma statement is given for any ℓ, ζ, note that when $\ell = \zeta$ the resulting statement remains interesting. Indeed, in that case, the maximum congestion that passes over any cluster edge over the course of algorithm $\mathcal{A}$ is $\tilde{O}(1)$.

3.2 Message-Time Trade-off Simulations

In this section, we provide a simulation that works for any BCONGEST aggregation-based algorithm $\mathcal{A}$ with a known upper bound $T_{\mathcal{A}}$ on the round complexity — see Definition 3.1 for a formal definition of aggregation-based algorithms, but these are, roughly speaking, algorithms in which all nodes locally update their state every round using some *aggregate* function such as min, max, average, etc. Algorithm $\mathcal{A}$ may use up to $O(T_{\mathcal{A}} \cdot n^2)$ messages (say, in dense graphs). We present a simulation that, informally, allows to obtain a CONGEST algorithm $\mathcal{A}'$ simulating $\mathcal{A}$ (i.e., producing the same output) sending less messages at the cost of using more rounds — see Theorem 3.9 below.

Theorem 3.9. *For any $\varepsilon \in [\frac{1}{\Theta(\log n)}, 1]$ and a pruned Baswana-Sen cluster hierarchy $\mathcal{H}$ of parameter ε, any* BCONGEST *aggregation-based algorithm $\mathcal{A}$ with (known upper bound on the) runtime $T_{\mathcal{A}}$ can be converted into a randomized (Monte Carlo)* CONGEST *algorithm that (a) simulates $\mathcal{A}$ correctly (w.h.p.), (b) runs in $\tilde{O}(T_{\mathcal{A}} \cdot n + n^{2-\varepsilon})$ rounds, (c) sends $\tilde{O}(T_{\mathcal{A}} \cdot n^{1+\varepsilon} + m)$ messages (w.h.p.), and (d) incurs $\tilde{O}(T_{\mathcal{A}})$ maximum edge congestion over non cluster edges of $\mathcal{H}$.*

Moreover, we adapt the above simulation to be faster when $\varepsilon \in [1/2, 1]$ (see Theorem 3.10 below).

Theorem 3.10. *For any $\varepsilon \in [1/2, 1]$ and a pruned Baswana-Sen cluster hierarchy $\mathcal{H}$ of parameter ε, any* BCONGEST *aggregation-based algorithm $\mathcal{A}$ with (known upper bound on the) runtime $T_{\mathcal{A}}$ can be converted into a randomized (Monte Carlo)* CONGEST *algorithm that (a) simulates $\mathcal{A}$ correctly with high probability, (b) takes $\tilde{O}(T_{\mathcal{A}} \cdot n^{1-\varepsilon} + n)$ runtime, (c) sends $\tilde{O}(T_{\mathcal{A}} \cdot n^{1+\varepsilon} + m)$ messages with high probability, and (d) incurs $\tilde{O}(T_{\mathcal{A}})$ maximum edge congestion over non cluster edges of $\mathcal{H}$.*

The analysis of the two subsequent simulations, and proofs of the above theorems, are deferred to the full version [10].

3.2.1 General Simulation for $\varepsilon \in [\frac{1}{\Theta(\log n)}, 1]$: Our first, general simulation — in that it applies to a wider range of ε values — takes as input a pruned Baswana-Sen cluster hierarchy (with parameter ε). Then, Algorithm $\mathcal{A}'$ can be decomposed into two parts: a preprocessing part and a simulation part. In the preprocessing part, nodes aggregate information over each of the clusters of the pruned Baswana-Sen cluster hierarchy. In the simulation part, we simulate the execution of $\mathcal{A}$ using the cluster hierarchy. Referring back to Theorem 3.9, the $\tilde{O}(n^{2-\varepsilon})$ term in the running time and the $\tilde{O}(m)$ term in the message complexity are due to the preprocessing, whereas the $\tilde{O}(T_{\mathcal{A}} \cdot n)$ term in the running time and the $\tilde{O}(T_{\mathcal{A}} \cdot n^{1+\varepsilon})$ term in the message complexity are due to the round-by-round simulation of $\mathcal{A}$.

Preprocessing. The preprocessing part of $\mathcal{A}'$ comprises two steps.

(1) Elect a leader, compute a BFS tree rooted in that leader, aggregate the number of nodes n and finally broadcast n to all nodes.
(2) Iteratively over the $\kappa + 1$ levels of the pruned Baswana-Sen cluster hierarchy, do the following: For each cluster C in a given level, its cluster center gathers the information of all edges incident to nodes in C. More precisely, each node in C upcasts $O(d(v) \log n)$ bits describing its 1-hop neighborhood in the communication graph G and in the pruned Baswana-Sen cluster hierarchy, broken up in $O(\log n)$ bit messages, to its parent in the cluster tree of C. Indeed, $O(d(v)(\log n + \kappa)) = O(d(v) \log n)$ bits can describe all ID pairs corresponding to the $d(v)$ incident edges to v, as well as the $\kappa = O(\log n)$ bits per such pair to indicate whether the corresponding edge is an inter-communication edge of a given level of the hierarchy.

Simulation. All nodes start the simulation (i.e., $\mathcal{A}'$) with the same initial state as in $\mathcal{A}$ (and some additional information obtained in the preprocessing). After which, any phase $p \in [1, T_{\mathcal{A}}]$ of the simulation uses $\tilde{O}(n)$ rounds to simulate round p of $\mathcal{A}$ (where the precise runtime of each phase depends on the analysis). More precisely, if we let B_p be the set of nodes broadcasting in round p of $\mathcal{A}$ (given the above mentioned initial state), and among these we let $B_p(u)$ denote the neighbors of any node $u \in V$ contained in B_p, then each phase of $\mathcal{A}'$ simulates the communication of the corresponding round of $\mathcal{A}$ in the aggregate sense: that is, for any phase $p \in [1, T_{\mathcal{A}}]$ and node $u \in V$, during phase p node u receives the aggregate of all messages sent by (nodes in) $B_p(u)$ (or in other words, function $agg_{u,r}$ applied to all messages sent by $B_p(u)$).

Each phase of $\mathcal{A}'$ simulates a round of $\mathcal{A}$ through the following three *send, receive and compute* steps, described for some arbitrary node v:

(1) **Send:** If v broadcasts a message m_v in round p of $\mathcal{A}$, then in this step (of phase p) node v executes the *indirect send and direct send* sub-steps that ensure that for each neighbor u of v and cluster C containing v, either:
   (i) **Indirect Send:** The message m_v is sent to some cluster containing u,
   (ii) **Direct (Aggregate) Send:** Or an aggregate (of at most $\tilde{O}(1)$ bits) of all messages from (nodes in) $B_p(u) \cap C$ is sent directly to u,
(2) **(Aggregate) Receive:** For any cluster C containing v in the hierarchy, v receives an aggregate of all messages sent to C in the indirect send sub-step by some node in $B_p(v)$.
(3) **Compute:** Finally, v locally computes the aggregate of all the messages received in the send — more precisely, in the direct send sub-step — and receive step of this phase (i.e., via its its incident inter-communication edges, and the up to κ

clusters v can belong to) and updates its state for phase $p+1$ accordingly — just as v would update its state for round $p+1$ after receiving this aggregate in round p of $\mathcal{A}$.

When the send and receive steps are executed correctly — we give their implementation below — and $\mathcal{A}$ is an aggregation-based distributed algorithm, it is straightforward to see the simulation is correct; in particular, the output of each node in $\mathcal{A}'$ is exactly the output of that same node in $\mathcal{A}$. Note that the send step is separated into two sub-steps due the asymmetric nature of property (c) of the cluster hierarchy (see Theorem 3.3). More precisely, for any two neighbors u, v, there is either (i) an inter-communication edge incident to v and going to a cluster containing u, or (ii) an inter-communication edge incident to u and going to some node in C (but not necessarily v). Then, messages are sent between the cluster of v and u in the first case within the indirect send sub-step, and in the second case within the direct send sub-step.

Next, we describe the implementation of the send step of some phase p (in $\mathcal{A}'$). Consider some node v that broadcasts some message m_v in round p of $\mathcal{A}$. Then, in the send step of phase p, v executes the two sub-steps sequentially:

(i) **Indirect Send:** Node v sends m_v, appended with its ID, over any incident inter-communication edges of the pruned Baswana-Sen cluster hierarchy.

(ii) **Direct (Aggregate) Send:** First, v upcasts m_v over all cluster trees it belongs to in the cluster hierarchy. Once all upcasts are done, let C be any cluster, $\mathcal{M}(C)$ be all messages received by the center of C (during the upcast) and $R(C)$ be the set of nodes outside C, but with a neighbor in C and an inter-communication edge to C (where the latter can be locally computed by the center of C from the knowledge obtained during the preprocessing). Then, for any node $u \in R(C)$, the center of C computes the aggregate of all messages in $\mathcal{M}(C)$ originating from nodes in $B_p(u)$. (Note that each such aggregate may consist of at most $\tilde{O}(1)$ bits, and sending this information is done via at most $\tilde{O}(1)$ bits, the collection of which we call an *aggregate packet*.) After which, the center of C downcasts, simultaneously for all nodes $u \in R(C)$, the corresponding computed aggregate packet to the endpoint in C of (one of) the inter-communication edge incident to u, after appending each aggregate packet with the ID of u. (Note that the same node in C may be the endpoint of many such inter-communication edges.) Finally, any aggregate packet received by some node w (after the downcast terminates) is sent by w over the corresponding inter-communication edge (i.e., where the other endpoint is the node with the ID appended to the received message).

Now, we describe the implementation of the receive step of some phase p (in $\mathcal{A}'$). Consider any node v that receives some message in the indirect send sub-step of that same phase — importantly, these cannot be messages from an aggregate packet. Then, v upcasts any such message over all cluster trees it belongs to. Moreover, all broadcasting nodes in B_p also upcast their message over all cluster trees they belong to. Once all upcasts are done, let C be any cluster and $\mathcal{M}(C)'$ be all messages received by the center of C (during the upcast). Then, for any node $u \in C$, the center of C computes the aggregate of all messages in $\mathcal{M}(C)'$ originating from nodes in $B_p(v)$. (Note that the cluster center can compute which messages in $\mathcal{M}(C)'$ originate from nodes in $B_p(v)$ because each message has its sender ID appended.) After which, the center of C downcasts, simultaneously for all nodes $u \in C$, the corresponding computed aggregate packet to u.

3.2.2 Improved Simulation for $\varepsilon \in [1/2, 1]$: This simulation takes as input a pruned Baswana-Sen cluster hierarchy with parameter $\varepsilon \geq 1/2$ (i.e., with at most 3 levels). Note that the cluster hierarchy comprises a partition C_0 of V into singleton clusters, and possibly a partition C_1 of some subset $V_1 = V \setminus L_1$ into depth 1 clusters — which we call *star clusters*. Moreover, the following properties (see Lemma 3.11) can be easily shown to hold with high probability.

Lemma 3.11. *Any pruned Baswana-Sen cluster hierarchy with parameter $\varepsilon \geq 1/2$ satisfies the following properties with high probability:*

- *For any node $v \in L_1$, v has at most $O(n^\varepsilon \log n)$ incident edges in G, and all are in F_1.*
- $|C_1| = \tilde{O}(n^{1-\varepsilon})$.

The improved simulation also consists of a preprocessing part followed by a simulation part. The preprocessing part is identical to the preprocessing for general ε (see Sec. 3.2.1), whereas the simulation part contains some key differences. Recall that the simulation part for general ε works in phases, each simulating one round of $\mathcal{A}$ through the three *send, receive and compute* steps. Here, the key differences are that each phase takes significantly less rounds — which is allowed since we limit the congestion over cluster edges to $\tilde{O}(n^{1-\varepsilon})$ per phase — and the send step is implemented differently (but the receive and compute steps remain identical).

More concretely, this send step is implemented as follows. Consider some node v broadcasting a message m_v in round p of $\mathcal{A}$.

- If $v \in L_1$, then v simply sends m_v over all incident inter-communication edges — this implements the direct send sub-step (and there is no indirect send sub-step for L_1 nodes).
- Otherwise, v is part of a star cluster, say C, in which case v sends m_v to its parent (i.e., the center of C). After receiving messages from all broadcasting nodes in C (i.e., from $B_p \cap C$), the center executes the following local computations for any neighboring cluster $C' \in C_1$:
  - The center computes the set $R_p(C, C')$ comprising the neighbors in C' of nodes in $B_p \cap C$ — i.e., nodes in $N(B_p \cap C) \cap C'$ — using its knowledge of all edges incident to nodes in C. (Note that $C' \cap C = \emptyset$.)
  - Next, the center computes a maximal matching $M(C, C') \subseteq E$ between nodes in $B_p \cap C$ and $R_p(C, C')$.
  - For each edge $e = (w, u) \in M(C, C')$, where w.l.o.g. $w \in C$, the center computes a message $m_1(e)$ containing the ID of w as well as the message m_w broadcasted by w, and a aggregate packet $m_2(e)$ which consists of $agg_{u,p}$ applied to all messages sent by nodes in $B_p(u) \cap C$ — the resulting aggregate packet contains $\tilde{O}(1)$ bits, by Definition 3.1, and thus can be transmitted using $\tilde{O}(1)$ only.

  After these local computations, for any $C' \in C_1$ and any edge $e \in M(C, C')$, the center sends $m_1(e)$ and $m_2(e)$ to the endpoint of e in C. (Note that this can amount to at most $\tilde{O}(n^{1-\varepsilon})$ messages on the edge from the center to w, w.h.p.,

since $|C_1| = \tilde{O}(n^{1-\varepsilon})$ w.h.p.) Upon reception, that endpoint node sends two different messages, containing respectively $m_1(e)$ and $m_2(e)$, through e. Over the entire cluster C, the first messages implement the indirect send sub-step, whereas the second messages (or packets) implement the direct send sub-step.

3.3 Message-Time Trade-offs for APSP

Consider the unweighted APSP problem, where the goal is for each node to output all of its distances to all other nodes. We give an unweighted APSP algorithm that achieves a natural message-time trade-off between the round-optimal (but non message-optimal) algorithm (see [5]) and the message-optimal (but not runtime-optimal) algorithm (see Section 2).

First, we point out that for $\varepsilon = O(1/\log n)$, the trade-off statement reduces to solving unweighted APSP $\tilde{O}(n^2)$ round and message complexities. This can be achieved by using the (weighted) APSP algorithm obtained in Section 2.

The remainder of the trade-off requires more sophisticated techniques, including the simulations described in Section 3.2. In what follows, we first show how to execute some polynomial in n amount of BFS computations (possibly of limited depth, depending on the range of ε) simultaneously, while getting a message-time trade-off.

More precisely, a BFS tree computation (up to some depth d) should ensure that each node v know its parent in some BFS of depth d rooted in some node u, if the distance between u, v is at most d. Note that typically, the random delays technique or congestion plus dilation framework — see Section 1.4 — can be used to run multiple BFS computations time-efficiently, in a way that takes advantage of the low congestion per edge induced by a BFS computation. Here, we show how to simulate multiple BFS algorithms with significantly lower message complexity, but at the cost of a higher runtime; here also, we leverage the low congestion per edge induced by a BFS computation. The more precise statement is captured by the following two auxiliary Lemmas 3.12 and 3.13, which play a key role in our unweighted APSP algorithm that achieves the remainder of the trade-off.

Lemma 3.12. *For any $\varepsilon \in [1/2, 1]$, there exists a* CONGEST *algorithm that computes n BFS trees (w.h.p.), using $\tilde{O}(n^{2-\varepsilon})$ rounds and sending (w.h.p.) $\tilde{O}(n^{2+\varepsilon})$ messages.*

Lemma 3.13. *For any $\varepsilon \in [\frac{1}{\Theta(\log n)}, 1/2]$, there exists a* CONGEST *algorithm that computes n BFS trees up to depth $\tilde{O}(n^{1-\varepsilon})$ (w.h.p.), using $\tilde{O}(n^{2-\varepsilon})$ rounds and sending (w.h.p.) $\tilde{O}(n^{2+\varepsilon})$ messages.*

With the above two lemmas, we can show how to obtain the remaining portions of the trade-off for unweighted APSP.

Trade-off for $\varepsilon \in [1/2, 1]$. To obtain the message-time trade-off for $\varepsilon \in [1/2, 1]$, we first apply Lemma 3.12, to obtain an algorithm running n independent BFS computations, using $\tilde{O}(n^{2-\varepsilon})$ rounds and sending (w.h.p.) $\tilde{O}(n^{2+\varepsilon})$ messages. Once all BFS computations are done, each node knows its distance to all other nodes by taking its depth in each of the computed BFS trees.

Trade-off for $\varepsilon \in [\frac{1}{\Theta(\log n)}, 1/2]$. This part of the trade-off is more complex. We first compute all distances between nearby pairs of nodes — that is, at most $\tilde{O}(n^{1-\varepsilon})$ hops apart — by applying Lemma 3.13. We obtain an algorithm that runs n independent BFS computations up to depth $\tilde{O}(n^{1-\varepsilon})$, using $\tilde{O}(n^{2-\varepsilon})$ rounds and sending (w.h.p.) $\tilde{O}(n^{2+\varepsilon})$ messages. Upon termination, each node v stores its depth in each BFS tree as its distance to the BFS root (whose ID v knows). Since we consider an undirected graph, this clearly allows each node to compute its distance to all other nodes within $\tilde{O}(n^{1-\varepsilon})$ hops.

As for computing the distances between any two nodes at least $\tilde{\Omega}(n^{1-\varepsilon})$ hops apart, we first sample $\tilde{\Theta}(n^{\varepsilon})$ nodes, called *landmarks*, uniformly at random in V, and from each such landmark, compute a BFS tree rooted at that landmark (without any of the techniques used above). Once all $\tilde{O}(n^{\varepsilon})$ BFS trees have been computed, each landmark (or root) gathers the information of all of the edges in the BFS tree (i.e., the IDs of any BFS edge's two endpoints) via an upcast operation. Then, each landmark broadcasts the information of the BFS tree (i.e, all ID pairs corresponding to all edges in the BFS tree) to all nodes. Finally, each node v computes its distance to any node u via any of these $\tilde{O}(n^{\varepsilon})$ BFS trees, and updates its distance to u if either it had no prior computed distance to u, or its distance to u in memory is higher.

It is straightforward enough to see that with high probability, for any two nodes u, v that are some $\tilde{\Omega}(n^{1-\varepsilon})$ hops apart, both u and v end up computing their distance between each other.

Theorem 1.2. *For any $\varepsilon \in [0, 1]$, unweighted APSP (on undirected graphs) can be solved (w.h.p.) in $\tilde{O}(n^{2-\varepsilon})$ rounds and $\tilde{O}(n^{2+\varepsilon})$ messages (w.h.p.) in* CONGEST.

4 Conclusions and Future Work

In this paper, we first obtained message-optimal polynomial-round distributed algorithms for several well-studied graph problems, including weighted APSP, that did not have prior any such algorithms. While these problems are very different, our results were all obtained by applying a unified framework to existing round-optimal algorithms for these problems. However, as noted earlier, our message-optimal algorithms are not round-optimal. We then showed that for unweighted APSP, we can construct a family of algorithms that smoothly exhibits a message-time trade-off across the entire spectrum — from message optimality (on one end) to time optimality (on the other end). To the best of our knowledge, this is the first such tradeoff result known for a fundamental graph-optimization problem.

Several key open questions remain. Does APSP admit singularly-optimal algorithms, i.e., algorithms optimal with respect to *both* message and time complexities? Another important research direction is whether one can use our framework to obtain message-optimal algorithms for other problems, e.g., maximum matching in general graphs and obtain message-time tradeoffs for *weighted* APSP, maximum matching, minimum cut etc.

Acknowledgments

Gopal Pandurangan was supported in part by Army Research Office (ARO) grant W911NF-231-0191 and National Science Foundation (NSF) grant CCF-2402837. Sriram V. Pemmaraju was supported in part by the National Science Foundation (NSF) grant CCF-2402835. Peter Robinson was supported in part by the National Science Foundation (NSF) grant CCF-2402836.

References

[1] Amir Abboud, Keren Censor-Hillel, and Seri Khoury. 2016. Near-Linear Lower Bounds for Distributed Distance Computations, Even in Sparse Networks. In *Proceedings of the 30th International Symposium on Distributed Computing, DISC*. 29–42.

[2] Amir Abboud, Keren Censor-Hillel, Seri Khoury, and Ami Paz. 2021. Smaller Cuts, Higher Lower Bounds. *ACM Trans. Algorithms* 17, 4 (2021), 30:1–30:40. doi:10.1145/3469834

[3] Surender Baswana and Sandeep Sen. 2007. A Simple and Linear Time Randomized Algorithm for Computing Sparse Spanners in Weighted Graphs. *Random Struct. Algorithms* 30, 4 (jul 2007), 532–563.

[4] Ruben Becker, Andreas Karrenbauer, Sebastian Krinninger, and Christoph Lenzen. 2016. Approximate Undirected Transshipment and Shortest Paths via Gradient Descent. *CoRR* abs/1607.05127 (2016). http://arxiv.org/abs/1607.05127

[5] Aaron Bernstein and Danupon Nanongkai. 2021. Distributed Exact Weighted All-Pairs Shortest Paths in Randomized Near-Linear Time. *SIAM J. Comput.* 52, 2 (2021), STOC19–112.

[6] Keren Censor-Hillel, Seri Khoury, and Ami Paz. 2017. Quadratic and Near-Quadratic Lower Bounds for the CONGEST Model. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*. 10:1–10:16. doi:10.4230/LIPIcs.DISC.2017.10

[7] Shiri Chechik and Doron Mukhtar. 2019. Reachability and Shortest Paths in the Broadcast CONGEST Model. In *33rd International Symposium on Distributed Computing (DISC 2019) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 146)*, Jukka Suomela (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 11:1–11:13. doi:10.4230/LIPIcs.DISC.2019.11

[8] Shiri Chechik and Doron Mukhtar. 2020. Single-Source Shortest Paths in the CONGEST Model with Improved Bound. In *Proceedings of the 39th Symposium on Principles of Distributed Computing* (Virtual Event, Italy) *(PODC '20)*. Association for Computing Machinery, New York, NY, USA, 464–473. doi:10.1145/3382734.3405729

[9] Andrew Drucker, Fabian Kuhn, and Rotem Oshman. 2014. On the Power of the Congested Clique Model. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing* (Paris, France) *(PODC '14)*. ACM, New York, NY, USA, 367–376. doi:10.1145/2611462.2611493

[10] Fabien Dufoulon, Shreyas Pai, Gopal Pandurangan, Sriram Pemmaraju, and Peter Robinson. 2025. *Message Optimality and Message-Time Trade-offs for APSP and Beyond*. Technical Report 2504.21781. arXiv. doi:10.48550/arXiv.2504.21781 Full version of this paper.

[11] Fabien Dufoulon, Shreyas Pai, Gopal Pandurangan, Sriram V. Pemmaraju, and Peter Robinson. 2024. The Message Complexity of Distributed Graph Optimization. In *15th Innovations in Theoretical Computer Science Conference (ITCS 2024) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 287)*, Venkatesan Guruswami (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 41:1–41:26. doi:10.4230/LIPIcs.ITCS.2024.41

[12] Michael Elkin. 2017. A Simple Deterministic Distributed MST Algorithm, with Near-Optimal Time and Message Complexities. In *Proceedings of the ACM Symposium on Principles of Distributed Computing* (Washington, DC, USA) *(PODC '17)*. ACM, New York, NY, USA, 157–163. doi:10.1145/3087801.3087823

[13] Michael Elkin. 2020. Distributed Exact Shortest Paths in Sublinear Time. *J. ACM* 67, 3, Article 15 (May 2020), 36 pages. doi:10.1145/3387161

[14] Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. 2012. Networks cannot compute their diameter in sublinear time. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*. 1150–1162.

[15] Mohsen Ghaffari. 2015. Near-Optimal Scheduling of Distributed Algorithms. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, Chryssis Georgiou and Paul G. Spirakis (Eds.). ACM, 3–12. doi:10.1145/2767386.2767417

[16] Mohsen Ghaffari and Anton Trygub. 2024. A Near-Optimal Low-Energy Deterministic Distributed SSSP with Ramifications on Congestion and APSP. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing, PODC 2024, Nantes, France, June 17-21, 2024*, Ran Gelles, Dennis Olivetti, and Petr Kuznetsov (Eds.). ACM, 401–411. doi:10.1145/3662158.3662812

[17] Mohsen Ghaffari and Goran Zuzic. 2022. Universally-Optimal Distributed Exact Min-Cut. In *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, Alessia Milani and Philipp Woelfel (Eds.). ACM, 281–291. doi:10.1145/3519270.3538429

[18] Robert Gmyr and Gopal Pandurangan. 2018. Time-Message Trade-Offs in Distributed Algorithms. In *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018 (LIPIcs, Vol. 121)*, Ulrich Schmid and Josef Widder (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 32:1–32:18. doi:10.4230/LIPICS.DISC.2018.32

[19] Bernhard Haeupler, D. Ellis Hershkowitz, and David Wajc. 2018. Round- and Message-Optimal Distributed Graph Algorithms. In *PODC*. New York, NY, USA, 119–128. doi:10.1145/3212734.3212737

[20] Stephan Holzer and Nathan Pinsker. 2015. Approximation of Distances and Shortest Paths in the Broadcast Congest Clique. In *19th International Conference on Principles of Distributed Systems, OPODIS 2015, December 14-17, 2015, Rennes, France*. 6:1–6:16. doi:10.4230/LIPIcs.OPODIS.2015.6

[21] Qiang-Sheng Hua, Haoqiang Fan, Lixiang Qian, Ming Ai, Yangyang Li, Xuanhua Shi, and Hai Jin. 2016. Brief Announcement: A Tight Distributed Algorithm for All Pairs Shortest Paths and Applications. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures* (Pacific Grove, California, USA) *(SPAA '16)*. Association for Computing Machinery, New York, NY, USA, 439–441. doi:10.1145/2935764.2935812

[22] Chien-Chung Huang, Danupon Nanongkai, and Thatchaphol Saranurak. 2017. Distributed Exact Weighted All-Pairs Shortest Paths in $\tilde{O}(n^{5/4})$ Rounds. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. 168–179. doi:10.1109/FOCS.2017.24

[23] Janne H. Korhonen and Joel Rybicki. 2018. Deterministic Subgraph Detection in Broadcast CONGEST. In *21st International Conference on Principles of Distributed Systems (OPODIS 2017) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 95)*, James Aspnes, Alysson Bessani, Pascal Felber, and João Leitão (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 4:1–4:16. doi:10.4230/LIPIcs.OPODIS.2017.4

[24] Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. 2015. On the Complexity of Universal Leader Election. *J. ACM* 62, 1 (2015), 7:1–7:27.

[25] Frank Thomson Leighton, Bruce M. Maggs, and Satish Rao. 1994. Packet Routing and Job-Shop Scheduling in *O*(Congestion + Dilation) Steps. *Comb.* 14, 2 (1994), 167–186. doi:10.1007/BF01215349

[26] Michael Luby. 1986. A Simple Parallel Algorithm for the Maximal Independent Set Problem. *SIAM J. Comput.* 15, 4 (1986), 1036–1053. doi:10.1137/0215074 arXiv:https://doi.org/10.1137/0215074

[27] Gary L. Miller, Richard Peng, and Shen Chen Xu. 2013. Parallel Graph Decompositions Using Random Shifts. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures* (Montréal, Québec, Canada) *(SPAA '13)*. Association for Computing Machinery, New York, NY, USA, 196–203. doi:10.1145/2486159.2486180

[28] Shreyas Pai and Sriram V. Pemmaraju. 2019. Connectivity Lower Bounds in Broadcast Congested Clique. In *PODC*. 256–258. doi:10.1145/3293611.3331569

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

[30] David Peleg. 2000. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics.

[31] Václav Rozhon, Christoph Grunau, Bernhard Haeupler, Goran Zuzic, and Jason Li. 2022. Undirected (1+ϵ)-shortest paths via minor-aggregates: near-optimal deterministic parallel and distributed algorithms. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, Stefano Leonardi and Anupam Gupta (Eds.). ACM, 478–487. doi:10.1145/3519935.3520074