Fall 2016

Efficient graph computing on the congested clique

Vivek Sardeshmukh
University of Iowa

EFFICIENT GRAPH COMPUTING ON THE CONGESTED CLIQUE

by

Vivek Sardeshmukh

A thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Computer Science
in the Graduate College of
The University of Iowa

December 2016

Thesis Supervisor: Professor Sriram V. Pemmaraju

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

PH.D. THESIS

This is to certify that the Ph.D. thesis of

Vivek Sardeshmukh

has been approved by the Examining Committee for the thesis requirement for the Doctor of Philosophy degree in Computer Science at the December 2016 graduation.

Thesis committee: _____
Sriram Pemmaraju, Thesis Supervisor

_____
Kasturi Varadarajan

_____
Sukumar Ghosh

_____
Aaron Stump

_____
Samuel Burer

ACKNOWLEDGEMENTS

This thesis, and all the results herein would not have been possible without the guidance of my advisor and mentor, Sriram Pemmaraju. I would like to thank him for all his support since my first day at Iowa. I have learned a lot from Sriram's generous and sage advise over the years, and I am deeply indebted to him for my growth, both as a researcher and as a person. Sriram has always been there, spending countless hours discussing research ideas, proof-reading my drafts, improving my technical presentations, and continuously encouraging me whenever I feel low by taking a walk around the campus. It has been a wonderful experience working so closely with him. Thank you Sriram.

I would also like to thank Kasturi Varadarajan. I learned a lot of tools for my research from his courses like Design and Analysis of Algorithms, Computational Geometry, and Algorithmic Excursions. Along with Sriram's guidance, it was his courses that made me comfortable with theoretical computer science.

I would also thank my committee members: Sam Burer, Sukumar Ghosh, Aaron Stump, and Kasturi Varadarajan for agreeing to be in my defense committee. I would also like to thank Gopal Pandurangan and Michele Scquizzato for collaborating with us on the first super-fast MST algorithm work.

I would like to thank my colleague and friend, James Hegeman, for the helpful research discussions, constant encouragement, and proof-reading my manuscripts. I want to thank my past and present colleagues and members of the Algorithms Reading

Group for many stimulating exchange of ideas, especially, Andrew Berns who helped me a lot in the beginning of this work.

I thank the Department of Computer Science for supporting me with several teaching assistantships. It was a privilege being part of this department.

A big shout out to all my friends in Iowa City for helping me to make this town my home for the last five years. I would like to thank Chaitanya, Namita, Prashant, Preethi, Rahil, Sampada, Sandeep, and Shabih for listening to my several practice talks and for the random discussions. I apologize in advance to those whose names I forget to mention here, but Thank you.

Finally, many thanks to my parents and my family for their unconditional love, support, and encouragement. Without them it would have been impossible for me to finish this five-year long journey. Above all, I would like to thank Pranalee without whose eternal support, I could not have been writing this thesis.

ABSTRACT

In this report, we initiate study on understanding a theoretical model for distributed computing called *Congested Clique*. This report presents constant-time and near-constant-time distributed algorithms for a variety of problems in the Congested Clique model.

We start by showing how to compute a 3-ruling set in expected $O(\log\log\log n)$ rounds and using this, we obtain a constant-approximation to metric facility location, also in expected $O(\log\log\log n)$ rounds. In addition, assuming an input metric space of constant doubling dimension, we obtain constant-round algorithms to compute maximal independent set on distance-threshold graphs and constant-factor approximation to the metric facility location problem. These results significantly improve on the running time of the fastest known algorithms for these problems in the Congested Clique setting.

Then, we study two fundamental graph problems, Graph Connectivity (GC) and Minimum Spanning Tree (MST), in the *Congested Clique* model, and present several new bounds on the time and message complexities of randomized algorithms for these problems. No non-trivial (i.e., super-constant) time lower bounds are known for either of the aforementioned problems; in particular, an important open question is whether or not constant-round algorithms exist for these problems. We make progress toward answering this question by presenting randomized Monte Carlo algorithms for both problems that run in $O(\log\log\log n)$ rounds (where n is the size of the clique). In

addition, assuming an input metric space of constant doubling dimension, we obtain constant-round algorithm the MST problem. Our results improve by an exponential factor on the long-standing (deterministic) time bound of $O(\log \log n)$ rounds for these problems due to Lotker et al. (SICOMP 2005). Our algorithms make use of several algorithmic tools including graph sketching, random sampling, and fast sorting.

Thus far there has been little work on understanding the *message complexity* of problems in the Congested Clique. In this report, we initiate a study on the message complexity of Congested Clique algorithms. We study two graph problems, *Graph Connectivity* (GC) and *Minimum Spanning Tree* (MST), in the Congested Clique model, focusing on the design of fast algorithms with *low message complexity*. Our motivation comes from recently established connections between the Congested Clique model and models of large-scale distributed computing such as MapReduce (Hegeman et al., SIROCCO 2014) and the "big data" model (Klauck et al., SODA 2015). For these connections to be fruitful, Congested Clique algorithms not only need to be fast, they also need to have low message complexity. While the aforementioned algorithms are fast, they have an $\Omega(n^2)$ message complexity, which makes them impractical in the context of the MapReduce and "big data" models.

This motivates our goal of achieving low message complexity, without sacrificing the speed of the algorithm. We start with the simpler GC problem and show that it can be solved in $O(\log \log \log n)$ rounds using only $O(n \operatorname{poly} \log n)$ messages. Then we derive subroutines to aid our earlier MST algorithm to run in $O(\log \log \log n)$ rounds using $O(m \operatorname{poly} \log n)$ messages on an m-edge input graph. Then, we present

an algorithm running in $O(\log^* n)$ rounds, with message complexity $\widetilde{O}(\sqrt{m \cdot n})$ and then build on this algorithm to derive a family of algorithms, containing for any ε, $0 < \varepsilon \leq 1$, an algorithm running in $O(\log^* n/\varepsilon)$ rounds, using $O(n^{1+\varepsilon}/\varepsilon)$ messages. Setting $\varepsilon = \log\log n/\log n$ leads to the first sub-logarithmic round Congested Clique MST algorithm that uses only $\widetilde{O}(n)$ messages.

Our results are a step toward understanding the power of randomization in the Congested Clique with respect to both time and message complexity.

PUBLIC ABSTRACT

In this report, we initiate study on understanding a theoretical model for distributed computing called *Congested Clique*. This report presents constant-time and near-constant-time distributed algorithms for a variety of problems in the Congested Clique model.

We start by showing how to compute a 3-ruling set in expected $O(\log\log\log n)$ rounds and using this, we obtain a constant-approximation to metric facility location, also in expected $O(\log\log\log n)$ rounds. In addition, assuming an input metric space of constant doubling dimension, we obtain constant-round algorithms to compute maximal independent set on distance-threshold graphs and constant-factor approximation to the metric facility location problem. These results significantly improve on the running time of the fastest known algorithms for these problems in the Congested Clique setting.

Then, we study two fundamental graph problems, Graph Connectivity (GC) and Minimum Spanning Tree (MST), in the *Congested Clique* model, and present several new bounds on the time and message complexities of randomized algorithms for these problems. No non-trivial (i.e., super-constant) time lower bounds are known for either of the aforementioned problems; in particular, an important open question is whether or not constant-round algorithms exist for these problems. We make progress toward answering this question by presenting randomized Monte Carlo algorithms for both problems that run in $O(\log\log\log n)$ rounds (where n is the size of the clique). In

addition, assuming an input metric space of constant doubling dimension, we obtain constant-round algorithm the MST problem. Our results improve by an exponential factor on the long-standing (deterministic) time bound of $O(\log \log n)$ rounds for these problems due to Lotker et al. (SICOMP 2005). Our algorithms make use of several algorithmic tools including graph sketching, random sampling, and fast sorting.

Thus far there has been little work on understanding the *message complexity* of problems in the Congested Clique. In this report, we initiate a study on the message complexity of Congested Clique algorithms. We study two graph problems, *Graph Connectivity* (GC) and *Minimum Spanning Tree* (MST), in the Congested Clique model, focusing on the design of fast algorithms with *low message complexity*. Our motivation comes from recently established connections between the Congested Clique model and models of large-scale distributed computing such as MapReduce (Hegeman et al., SIROCCO 2014) and the "big data" model (Klauck et al., SODA 2015). For these connections to be fruitful, Congested Clique algorithms not only need to be fast, they also need to have low message complexity. While the aforementioned algorithms are fast, they have an $\Omega(n^2)$ message complexity, which makes them impractical in the context of the MapReduce and "big data" models.

This motivates our goal of achieving low message complexity, without sacrificing the speed of the algorithm. We start with the simpler GC problem and show that it can be solved in $O(\log \log \log n)$ rounds using only $O(n \operatorname{poly} \log n)$ messages. Then we derive subroutines to aid our earlier MST algorithm to run in $O(\log \log \log n)$ rounds using $O(m \operatorname{poly} \log n)$ messages on an m-edge input graph. Then, we present

an algorithm running in $O(\log^* n)$ rounds, with message complexity $\widetilde{O}(\sqrt{m \cdot n})$ and then build on this algorithm to derive a family of algorithms, containing for any ε, $0 < \varepsilon \leq 1$, an algorithm running in $O(\log^* n/\varepsilon)$ rounds, using $O(n^{1+\varepsilon}/\varepsilon)$ messages. Setting $\varepsilon = \log\log n/\log n$ leads to the first sub-logarithmic round Congested Clique MST algorithm that uses only $\widetilde{O}(n)$ messages.

Our results are a step toward understanding the power of randomization in the Congested Clique with respect to both time and message complexity.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

Figure

CHAPTER 1
INTRODUCTION

Fundamental graph-theoretic problems such as computing Minimum Spanning Tree (MST) frequently arise in science and engineering domains. These problems show up as instances of models of real-world phenomena or as a subproblem of its encompassing problem, often proving to be a bottleneck in the overall computation. In today's era of Big Data, the problem is exacerbated due to the sheer scale of the graphs originating in domains such as bioinformatics, social network analysis, and other relational contexts. Hence, efficient processing of such large-scale graphs is paramount and acts as a significant challenge to the field of computer science. One way to gain this efficiency is by means of distributing the processing of the graphs to a network of machines which perform computation in parallel, communicating with each other as required. Due to the advent of "cloud computing", high-performance computing, overlay networks, etc. most of these networks are fully connected, that is, any machine can directly communicate with any other machine. To capture these fully-connected networks, in this report we focus on the *Congested Clique* model of distributed computing.

Distributed graph computing is sometimes necessary because the input is too large to fit into a single machine (e.g., the Facebook graph), sometimes because the input is inherently distributed (e.g., the Internet), sometimes because adding more processing power might speed-up the computation. A distributed system is generally a network of machines connected by *communication channels*. Communication channels

can be physical wires, wireless radio transmissions, or even logical entities consisting of mixture of wired and wireless connections. In the past, distributed graph computing is studied by letting the input graph act as communication network. That is, each node in the graph is a machine in the network and each edge in the input graph acts as a communication channel. This tight coupling of input graph and the network was motivated by several distributed systems such as the Internet. One of the important challenge in distributed computing is *congestion* [50]. The CONGEST model is designed to study the effect of congestion and *distance* on various problem by restricting size of messages that node can send over a communication channel in unit time. The lack of *local* information is one of the key challenges in the distributed computing [50] and hence, classical sequential algorithms are not sufficient to solve the problems in distributed manner. Few problems such as maximal independent set and coloring are *local* in nature. So a natural extension to the CONGEST model is to take out the congestion and focus on the *locality* alone. This model is called LOCAL model. Both, CONGEST and LOCAL model are widely studied and well understood by the community. The next natural extension to these model, in a theoretical point of view, is to just focus on congestion alone. That is, all the information is at most one hop away. We refer to this model as the *Congested Clique* model. More formally, the CONGEST model is a synchronous, message-passing model of distributed computation in which the amount of information that a node can transmit along an incident communication link in one round is restricted to $O(\log n)$ bits, where n is the size of the network [50]. As the name suggests, the CONGEST model focuses on

congestion as an obstacle to distributed computation. In this report, we focus on the design of distributed algorithms in the CONGEST model on a *clique* communication network; we call this the *Congested Clique* model. In the Congested Clique model, all information is nearby, i.e., at most one hop away, and so any difficulty in solving a problem is due to congestion alone.

Though initially the Congested Clique model was considered just a theoretical model, recently there has been work showing connection between real-world distributed systems such as MapReduce model [22] and Big Data model [31]. Also the rise in overlay networks and peer to peer (P2P) networks are another motivation to study the Congested Clique model.

1.1 Congested Clique Model

The *Congested Clique* model consists of n machines that can communicate with each other via an underlying complete network. A key feature of the model is the *bandwidth restriction* on the communication links, i.e., only a limited number of bits (typically $O(\log n)$ bits, as assumed here) can be sent along each link in each round. In the Congested Clique, since the diameter of the communication network is just one, every machine is within one hop of every other machine and thus all information is quite local. The main algorithmic issue lies then in dealing with the potential congestion caused by the bandwidth restrictions.

Each vertex $v \in V$ of the network has a distinct identifier of $O(\log n)$ bits. At the beginning of the computation, each vertex knows its own identity, the number n,

and the part of the input it gets assigned. There are then two possible variants of the model based on the amount of knowledge available at the vertices regarding the network. In the first one, each computing entity initially knows, in addition to its own identity, the identity of its $n-1$ neighbors, while in the second one a computing entity initially does not know the identities of its $n-1$ neighbors. Following, e.g., [5], we denote these two variants as *KT1* and *KT0*, respectively. Thus, in the *KT0* model, each node can send and receive messages along $n-1$ communication links (without loss of generality, numbered $1, 2, \ldots, n-1$), without being aware of the identity of nodes at the other end of the communication links. In this report we assume the KT1 model.

The computation proceeds in synchronous rounds. In each round each node can perform some local computation and send a (possibly different) message of $O(\log n)$ bits to each of its $n-1$ neighbors. It is assumed that both the computing entities and the communication links are fault-free. The Congested Clique model is therefore specifically geared toward understanding the role of the limited bandwidth as a fundamental obstacle in distributed computing, in contrast to other classical models for distributed computing that instead focus, e.g., on the effects of latency (the LOCAL model) or on the effects of both latency and limited bandwidth (the CONGEST model).

In distributed computing two complexity measures are usually relevant: the *time complexity* of a computation is the total number of rounds to complete the computation, while the *message complexity* of a computation is the total number of

messages exchanged to complete the computation. In this report, we consider both complexity measures.

1.2 Related Work and Motivation

Indeed, there has been a lot of recent work in studying various fundamental problems in the Congested Clique model, including facility location [19, 6], minimum spanning tree (MST) [39, 23], shortest paths and distances [8, 24, 45], triangle finding [12, 11], subgraph detection [12], ruling sets [6, 23], sorting [49, 37], and routing [37]. The modelling assumption in solving these problems is that the input graph $G = (V, E)$ is "embedded" in the Congested Clique, that is, each node of G is uniquely mapped to a machine and the edges of G are naturally mapped to the links between the corresponding machines (cf. Section 1.1).

Research on the Congested Clique has focused mostly on the *time complexity* (i.e., the number of synchronous *rounds*) of these problems. The complete network allows $\Theta(n^2)$ (different) messages (each message is of size $O(\log n)$ bits) to be exchanged in each round, and many of the time-efficient algorithms for various problems have exploited this vast parallel communication ability to give "super-fast" algorithms that run in a sub-logarithmic (in n) number of rounds. An important early result is the work of Lotker et al. [39], which presented an $O(\log \log n)$-round deterministic algorithm for the MST problem. This was a significant improvement at the time (only an $O(\log n)$-round algorithm was known [50]). Lotker et al. left open the question of whether or not an even faster algorithm was possible—in particular,

whether a *constant-round* algorithm could be possible for MST or for the (simpler) problem of graph connectivity (GC). Regarding lower bounds, almost nothing non-trivial is known.[1] In particular, for the GC and MST problems, no *super-constant* time lower bounds are known.[2] The situation is not promising from the lower bounds side: the recent results of [13] have proved that showing substantially super-constant lower bounds on time in the Congested Clique is as hard as proving long-open lower bounds in circuit complexity. However, this leaves open the important question of whether or not constant-time algorithms are possible for GC as well as the MST problem. Furthermore, there has been little work published on understanding the *message complexity* of problems in the Congested Clique. In particular, to the best of our knowledge, we are not aware of any work that addresses tradeoffs between message and time complexities in Congested Clique.

In many applications, message complexity is the dominant cost as it plays a major role in determining the running time and auxiliary resources (e.g., energy) consumed by the algorithm. For example, communication cost is one of the dominant costs in distributed computation on large-scale data in modern data centers [31]. While computation cost performed locally within a machine might be high, transferring huge amounts of data (e.g., petabytes) *across* machines during

[1]This is the case with respect to the standard *unicast/multicast* version of the Congested Clique—the model assumed in this report—where each node can send a *different* message (or no message at all) along each of its incident links in each round. Recently, time lower bounds have been shown in the weaker *broadcast* version of the Congested Clique—where machines can send only the *same* message across its incident links in a round—for some problems such as shortest paths [24] and the subgraph detection problem [12].

[2]This is true even in the broadcast version of the model.

computation can consume a substantial amount of time and resources and usually dominates other costs [31]. In the particular context of the Congested Clique, optimizing messages as well as time has direct applications to the performance of distributed algorithms in other models such as the Big Data (k-machine) model [31], which was recently introduced to study distributed computation on large-scale graphs. The above work shows how to "convert" algorithms (cf. Conversion theorem of [31]) designed in the Congested Clique model to the Big Data model; the running time in the Big Data model depends on *both* the time *and* the message complexities of the corresponding algorithm in the Congested Clique model. As a consequence, the fastest algorithm in the Congested Clique model need not yield the fastest algorithm in the Big Data model; on the contrary, a slower but more message efficient algorithm in the Congested Clique can yield a faster algorithm in the Big Data model. Another related motivation comes from the connection between the Congested Clique model and the MapReduce model. In [22] it is shown that if a Congested Clique algorithm runs in T rounds and, in addition, has moderate message complexity, then it can be simulated in the MapReduce model in $O(T)$ rounds.

1.3 Problem Specifications

In this section, we formally define the algorithmic problems. In this report we focus on graph problems such as minimum spanning tree, graph connectivity, and t-ruling set. As an application of t-ruling set we show how to solve metric facility location problem. All these problems are formally defined below.

1.3.1 Maximal Independent Set and t-Ruling Set

An *independent set* of a graph $G = (V, E)$ is a set S of vertices such that for every two vertices in S, there is no edge connecting the two. A *maximal independent set (MIS)* is an independent set that is not a subset of any other independent set. In other words, there is no vertex outside the independent set that may join it because it is maximal with respect to the independent set property. A *t-ruling set* of a graph $G = (V, E)$ is an independent set $I \subseteq V$ such that every vertex in G is at most t hops from some vertex in I. A t-ruling set, for constant t, is a natural generalization of an MIS and can stand as a proxy for an MIS in many instances.

The input to the t-ruling set problem (or the MIS problem) on a Congested Clique $H = (V, E_H)$ is a spanning subgraph $G = (V, E)$ of the underlying communication network H. Each node $v \in V$ is initially aware of all its neighbors in G. At the end of the t-ruling set algorithm, every node is required to know the identities of all nodes in the computed t-ruling set.

1.3.2 Metric Facility Location

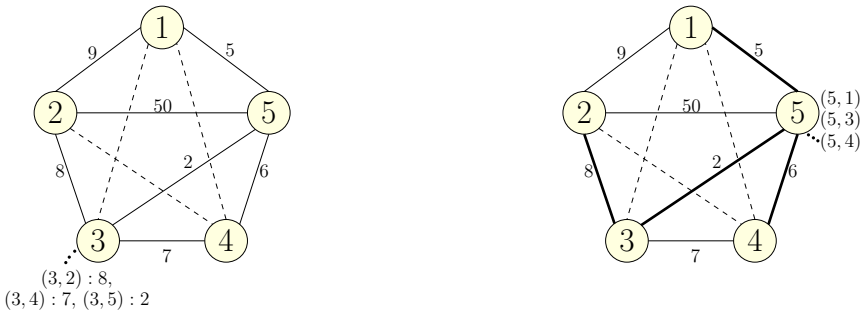
The input to the *metric facility location (MFL)* consists of a metric space (V, d) along with *facility opening costs* f_v associated with each node $v \in V$. The goal is to find a subset $F \subseteq V$ of nodes to *open* as facilities so as to minimize the facility opening costs plus connection costs, i.e., $\sum_{v \in F} f_v + \sum_{u \in V} D(u, F)$, where $D(u, F) := \min_{v \in F} d(u, v)$ is the *connection cost* of node u. Initially, each node $v \in V$ knows facility opening cost f_v and distances $d(v, w)$ for all $w \in V$.

1.3.3 Graph Connectivity

The input to the *graph connectivity (GC)* problem is a spanning subgraph $G = (V, E)$, $E \subseteq E_N$, of the underlying clique communication network $N = (V, E_N)$. The input to each node $v \in V$ consists of an $(n-1)$-bit vector where the i-th bit is associated with the i-th channel of v, indicates whether or not edge $(v, \mathrm{ID}(i)) \in E$, where $\mathrm{ID}(i)$ is the identifier of the node at the other end of v's i-th channel. At the end of the GC algorithm, we require that at least one machine knows whether G is connected or not.

1.3.4 Minimum Spanning Tree

The input to the *minimum spanning tree (MST)* problem is a a spanning subgraph $G = (V, E)$, $E \subseteq E_N$, of the underlying clique communication network $N = (V, E_N)$. The input to each node $v \in V$ consists of an $(n-1)$-bit vector where the i-th bit is associated with the i-th channel of v, indicates whether or not edge $(v, \mathrm{ID}(i)) \in E$, where $\mathrm{ID}(i)$ is the identifier of the node at the other end of v's i-th channel. Also, each node knows edge-weights of the incident edges. We assume that edge-weights can be represented using $O(\log n)$ bits. At the end of the MST algorithm, we require that each machine knows which of its incident edges belong to the output MST. . Figure 1.1 demonstrate the MST problem in the Congested Clique model on a 5-node graph.



(a) (b)

Figure 1.1: MST in the Congested Clique Model: input graph nodes and the network machines are shown in circles. The solid lines indicate the edges in the input graph along with weights. The solid lines and the dashed lines are the communication channels. Fig. (a) shows what machine 3 knows initially while the expected status of machine 5 in the end of computation is shown in Fig. (b).

1.4 Contributions

This section summarizes our contribution and also describes organization for the rest of the report.

1.4.1 Fast t-Ruling Set Computation and Application to Metric Facility Location

In Chapter 2 we present several constant-time or near-constant-time algorithms for fundamental problems in the Congested Clique setting. Specifically our results in Chapter 2 are:

- First, we present an algorithm that computes a 3-ruling set of G in expected $O(\log\log\log n)$ rounds, significantly improving the running time of the 2-ruling set algorithm of Berns et al. [7, 6].
- Via a reduction presented in Berns et al. [7, 6], this implies an expected $O(\log\log\log n)$-round algorithm for computing an $O(1)$-approximation for MFL. Again, this significantly improves on the running time of the fastest known algorithm for this problem.
- Secondly, we show how to compute MIS in growth-bounded graphs in $O(1)$ rounds.

Distributed algorithms that run in $O(\log\log n)$ rounds are typically analyzed by showing a doubly-exponential rate of progress; such progress, for example, is achieved if the number of nodes that have "successfully finished" grows by squaring after each iteration. The Congested Clique algorithms for MST due to Lotker et al. [40] and the above-mentioned MFL algorithm due to Berns et al. [7, 6] are both

examples of such phenomena. Our algorithm with triply-logarithmic running time, involves new techniques that seem applicable to Congested Clique algorithms in general. Our result raises the distinct possibility that other problems, e.g., MST, can also be solved in $O(\log\log\log n)$ rounds on a Congested Clique. In fact, our next set of results are motivated by this.

1.4.2 Fast Graph Connectivity Verification and Minimum Spanning Tree Construction

In Chapter 3 we focus on two fundamental graph problems in the Congested Clique, namely graph connectivity (GC) and minimum spanning tree (MST), and present several new results that make progress toward understanding the time and message complexities of randomized algorithms for these problems. In Chapter 3, we positively answer questions raised in Chapter 2. Moreover, we initiate study of message complexity of Congested Clique algorithms.

1.4.2.1 Faster Algorithms for GC and MST

Our first contribution consists of randomized (Monte Carlo) algorithms, running in $O(\log\log\log n)$ rounds and succeeding with high probability (w.h.p.), for both GC and MST[3]. Our results improve by an exponential factor on the long-standing time upper bound of $O(\log\log n)$ rounds for MST due to Lotker et al. [39]. It is worth mentioning that the Lotker et al. MST algorithm is deterministic, in contrast to ours, which uses randomness in a crucial way.

[3]Throughout this report, by "w.h.p." we mean with probability at least $1 - 1/n^{\Omega(1)}$.

It is worth emphasizing that if we allow $O(\text{polylog}\, n)$ bits per message, instead of $O(\log n)$ bits per message (which is the standard bound for the Congested Clique model), then our algorithm solves MST in $O(1)$ rounds. Lotker et al. point out that their algorithm also extends to a larger bandwidth setting; specifically, running in $O(\log 1/\varepsilon)$ rounds if each message is n^{ε} bits long. For example, this implies that the Lotker et al. algorithm would run in $O(1)$ rounds if each message was allowed to contain $\Theta(\sqrt{n})$ bits. This need for poly-sized messages should be contrasted with our MST algorithm which is capable of running in $O(1)$ rounds using only $O(\text{polylog}\, n)$-sized messages.

1.4.2.2 Focus on Message Complexity

Our $O(\log \log \log n)$-round randomized MST algorithm presented in Chapter 3 has a message complexity of $\Theta(n^2)$ (as does the algorithm of Lotker et al. [39]). As we improved on the time complexity using randomization, a natural question is whether we can improve the message complexity as well. By exploiting the synchronous nature of the model one can solve *any* problem fairly trivially with an algorithm that communicates only $O(n)$ bits. The $O(n)$ bits message complexity upper bound mentioned above is not particularly satisfying because the algorithm it depends on uses super-polynomially many rounds. This naturally leads to the question of whether MST (or GC) can be solved fast and using only a small number of messages. In Chapter 3, we provide a partial answer to this question in this chapter by presenting an MST algorithm that requires only $O(n\, \text{polylog}\, n)$ messages and $O(\text{polylog}\, n)$ rounds.

In Chapter 4 we make further progress towards this question.

1.4.3 Reducing Message Complexity of Graph Connectivity Verification

All of the MST algorithms mentioned above, essentially use the entire bandwidth of the Congested Clique model, i.e., they use $\Theta(n^2)$ messages. From these examples, one might (incorrectly!) conclude that "super-fast" Congested Clique algorithms are only possible when the entire bandwidth of the model is used. In our next set of results, we focus on the design of MST algorithms in the Congested Clique model that have low *message complexity*, while still remaining "super-fast." In our next set of contributions we partially answer questions raised in Chapter 3. These results are described in Chapter 4. We first focus on GC and show that it can be solved in $O(\log\log\log n)$ rounds using only $O(n \operatorname{poly}\log n)$ messages.

1.4.3.1 Low-message-complexity GC Algorithm

The GC algorithm in Chapter 3 that runs in $O(\log\log\log n)$ rounds, uses the fast parallel merge procedure of Lotker et al. [39] as a pre-processing step to reduce the size (number of vertices) of the input graph to $O(n/\operatorname{poly}\log n)$. This merge procedure uses $\Theta(n^2)$ messages per iteration and our next main technical contribution lies in showing how to reduce the message complexity of this merge procedure to $O(n \operatorname{poly}\log n)$ messages. Specifically, we show how to use linear sketches [1, 2, 42] to sample sufficiently many edges connecting different components in parallel so as to be useful to the fast parallel merge procedure of Lotker et al. [39]. An additional technique that we use to limit the message complexity of this algorithm

to $O(n \operatorname{poly} \log n)$ is the design of a simple routing primitive that has linear message complexity, while requiring only a constant number of rounds for completion. These results are described in Chapter 4.

1.4.3.2 Linear-message-complexity MST Algorithm

We then consider the more challenging MST problem and first in Chapter 4 show that MST can be solved in $O(\log \log \log n)$ rounds using $O(m \operatorname{poly} \log n)$ messages. In addition to the low-message-complexity routing primitive mentioned above, this result depends on a low-message-complexity sorting primitive (based on the Congested Clique sorting algorithm of [37]) that we present. While we do not know if *exact* MST can be solved in $O(\log \log \log n)$ rounds using $O(n \operatorname{poly} \log n)$ messages, we do show that for any $\varepsilon > 0$, a $(1+\varepsilon)$-approximation of MST can also be constructed in $O(\log \log \log n)$ rounds using only $O(n \operatorname{poly} \log n)$ messages. This final approximation algorithm result makes crucial use of the GC result and the exact-MST result mentioned above.

1.4.4 Reducing Message Complexity of Minimum Spanning Tree Construction

In Chapter 5, we positively answers questions raised in Chapter 3 and Chapter 4, by presenting an $O(\log^* n)$-round algorithm that uses $\widetilde{O}(\sqrt{m \cdot n})$ [4] messages for an n-node, m-edge input graph. Two points are worth noting about this message complexity upper bound: (i) it is bounded above by $\widetilde{O}(n^{1.5})$ for all values of m and is thus substantially sub-quadratic, independent of m and (ii) it is bounded above

[4] The notation $\widetilde{O}$ hides $\operatorname{poly}(\log n)$ factors.

by $o(m)$ for all values of m that are super-linear in n, i.e., when $m = \omega(n \operatorname{poly}(\log n))$. We then extend this result to design a family of algorithms parameterized by ε, $0 < \varepsilon \leq 1$, and running in $O(\log^* n/\varepsilon)$ rounds and using $\widetilde{O}(n^{1+\varepsilon}/\varepsilon)$ messages. If we set $\varepsilon = \log\log n/\log n$, we get an algorithm running in $O(\log^* n \cdot \log n/\log\log n)$ rounds and using $\widetilde{O}(n)$ messages. Thus we demonstrate the existence of a sub-logarithmic round MST algorithm using only $O(n \cdot \operatorname{poly}(\log n))$ messages, positively answering a question posed in Chapter 3 and Chapter 4. All of the round and message complexity bounds mentioned above hold with high probability (w.h.p.), i.e., with probability at least $1 - \frac{1}{n}$. Our results indicate that the power of the Congested Clique model lies not so much in its $\Theta(n^2)$ bandwidth as in the flexibility it provides – any communication link that is needed is present in the network, though most communication links may eventually not be needed.

Finally, in Chapter 6, we summarizes our understanding of the Congested Clique model and points out the open problems for future investigations.

CHAPTER 2
SUPER-FAST ALGORITHM FOR METRIC FACILITY LOCATION PROBLEM

2.1 Introduction

The input to the *metric facility location (MFL)* consists of a metric space (V, d) along with *facility opening costs* f_v associated with each node $v \in V$. The goal is to find a subset $F \subseteq V$ of nodes to *open* as facilities so as to minimize the facility opening costs plus connection costs, i.e., $\sum_{v \in F} f_v + \sum_{u \in V} D(u, F)$, where $D(u, F) := \min_{v \in F} d(u, v)$ is the *connection cost* of node u. The *facility location* problem is a more general version of the metric facility location problem without metric space.

The facility location problem has been used as an abstraction for the problem of locating resources in a wireless network [16]. Motivated by this application, several researchers have considered the facility location problem in a distributed setting [46, 47, 43]. More recently, Berns et al. [6] presented a $O(\log \log n)$-round algorithm that computes a $O(1)$-factor approximate solution to the MFL problem in the Congested Clique. Berns et al. achieved this result by reducing MFL to 2-ruling set problem and presented a $O(\log \log n)$-round 2-ruling set algorithm. In this chapter first we present a $O(\log \log \log n)$-round algorithm for 3-ruling set problem and using the reduction in [6] we obtain a $O(1)$-factor approximation to the MFL problem in $O(\log \log \log n)$ rounds in the Congested Clique improving it by an exponential factor. Then we

present a $O(1)$ round algorithm for the MIS problem in a more restricted setting. [1]

2.1.1 Related Work

A number of classical problems in distributed computing, e.g., maximal independent set (MIS), vertex coloring, edge coloring, maximal matching, shortest paths, etc., are well-defined in this setting. However, the difficulty of proving lower bounds in the Congested Clique model [13] means that it is not clear how quickly one should be able to solve any of these problems in this model. Note that the input G can be quite dense (e.g., have $\Theta(n^2)$ edges) and therefore any reasonably fast algorithm for the problem will have to be "truly" distributed in the sense that it cannot simply rely on shipping off the problem description to a single node for local computation. In this setting, the algorithm of Berns et al. [7, 6] that computes a *2-ruling set* of G in expected-$O(\log \log n)$ rounds is worth mentioning.

2.1.2 Main Results

In this chapter we present several constant-time or near-constant-time algorithms for fundamental problems in the Congested Clique setting.

- First, we present an algorithm that computes a 3-ruling set of G in expected $O(\log \log \log n)$ rounds, significantly improving the running time of the 2-ruling set algorithm of Berns et al. [7, 6].
- Via a reduction presented in Berns et al. [7, 6], this implies an expected $O(\log \log \log n)$-round algorithm for computing an $O(1)$-approximation for

[1] This chapter is derived from our work which appeared in 2014 [23].

MFL. Again, this significantly improves on the running time of the fastest known algorithm for this problem.

Distributed algorithms that run in $O(\log \log n)$ rounds are typically analyzed by showing a doubly-exponential rate of progress; such progress, for example, is achieved if the number of nodes that have "successfully finished" grows by squaring after each iteration. The Congested Clique algorithms for MST due to Lotker et al. [40] and the above-mentioned MFL algorithm due to Berns et al. [7, 6] are both examples of such phenomena. Our algorithm with triply-logarithmic running time, involves new techniques that seem applicable to Congested Clique algorithms in general. Our result raises the distinct possibility that other problems, e.g., MST, can also be solved in $O(\log \log \log n)$ rounds on a Congested Clique. In fact, our next set of results presented in the next chapter (Chapter 3) are motivated by this and we present $O(\log \log \log n)$-round MST algorithm.

2.2 Technical Preliminaries

2.2.1 Metric spaces, doubling dimension, and growth-bounded graphs

If $M = (V, d)$ is a metric space then we use $B_M(v, r)$ to denote the set of points $w \in V$ such that $d(v, w) \leq r$. We call $B_M(v, r)$ the *ball of radius r centered at v*. A metric space $M = (V, d)$ has *doubling dimension* ρ if for any $v \in V$ and $r \geq 0$, $B_M(v, r)$ is contained in the union of at most 2^ρ balls $B_M(u, r/2)$, $u \in V$. In this chapter, we work with metric spaces with constant doubling dimension, i.e., $\rho = O(1)$. Note that constant-dimensional Euclidean metric spaces are natural examples of

metric spaces with constant doubling dimension. In distributed computing literature, metric spaces of constant doubling dimension have been investigated in the context of wireless networks [10, 34]. For a graph $G = (V, E)$ and a node $v \in V$, let $B_G(v, r)$ denote the set of all vertices $u \in V$ that are at most r hops from v. A graph $G = (V, E)$ is said to have *bounded growth* (or said to be *growth-bounded*) if the size of any independent set in any ball $B_G(v, r)$, $v \in V$, $r \geq 0$, is bounded by $O(r^c)$ for some constant c. For any metric space (V, d) and $r \geq 0$, the graph $G_r = (V, E_r)$, where $E_r = \{\{u, v\} \in d(u, v) \leq r\}$ is called a *distance-threshold graph*. It is easy to see that if (V, d) has constant doubling dimension then a distance-threshold graph G_r, for any $r \geq 0$, is growth-bounded; this fact will play an important role in our algorithms. For a given metric space (V, d) the *aspect ratio* $\lambda(Y)$ of a subset of points $Y \subseteq V$ is the ratio of maximum of pair-wise distance between points in Y to the minimum of pair-wise distance between points in Y, i.e. $\lambda(Y) = \max\{d(u, v) \mid u, v \in Y\}/\min\{d(u, v) \mid u, v \in Y\}$. The following fact is easy to prove by applying the definition of doubling dimension: if (V, d) is a metric with doubling dimension ρ and $Y \subseteq V$ is a subset of points, then $|Y| \leq 2^{\rho \cdot \lceil \log_2 \lambda(Y) \rceil}$ where $\lambda(Y)$ is the aspect ratio of Y. We refer to this property as the *growth-bounded property* of the metric space (V, d). Distance-threshold graphs and more generally, growth-bounded graphs have attracted attention in the distributed computing community as flexible models of wireless networks [34]. Schneider and Wattenhofer [54] present a deterministic algorithm, running in $O(\log^* n)$ rounds, for computing an MIS on a growth-bounded graph.

2.2.2 Lenzen's routing protocol

A key algorithmic tool that allows us to design constant- and near-constant-time round algorithms is a recent deterministic routing protocol by Lenzen [37] that disseminates a large volume of information on a Congested Clique in constant rounds. The specific routing problem, called an *Information Distribution Task*, solved by Lenzen's protocol is the following. Each node $i \in V$ is given a set of $n' \leq n$ messages, each of size $O(\log n)$, $\{m_i^1, m_i^2, \ldots, m_i^{n'}\}$, with destinations $d(m_i^j) \in V$, $j \in [n']$. Messages are globally lexicographically ordered by their source i, destination $d(m_i^j)$, and j. Each node is also the destination of at most n messages. Lenzen's routing protocol solves the Information Distribution Task in $O(1)$ rounds.

2.2.3 General Notation

For a subset $S \subseteq V$, $G[S]$ denotes *induced* subgraph of G by set S; thus $G[S] = (S, E')$ where $E' = \{\{u, v\} \mid u, v \in S \text{ and } \{u, v\} \in E\}$. In the context of our MST algorithm we will interpret metric distances $d(u, v)$ as as edge weights; we will use $wt(u, v)$ and $d(u, v)$ interchangeably. Given an edge-weighted graph $G = (V, E)$ and an edge set $E' \subseteq E$, we denote the sum of all edge-weights in E' as $wt(E')$. We use Δ to denote the maximum degree of a graph; sometimes, to avoid ambiguity we use $\Delta(G)$ to denote maximum degree of graph G. All logarithms are assumed to have base 2 unless otherwise specified. We say an event occurs *with high probability* (w.h.p.), if the probability of that event is at least $(1 - 1/n^c)$ for a constant $c \geq 1$.

2.3 3-Ruling Sets in $O(\log\log\log n)$ Rounds

In this section, we show how nodes in V can use the underlying clique communication network H to compute, in expected-$O(\log\log\log n)$ rounds, a 3-ruling set of an arbitrary spanning subgraph G of H. At a high level, our 3-ruling set algorithm can be viewed as having three steps. In the first step, the graph is decomposed into $O(\log\log n)$ degree-based classes and at the end of this step every node knows the class it belongs to. In the next subsection, we describe this *degree-decomposition step* and show that it runs in expected $O(\log\log\log n)$ rounds. In the second step, each vertex v of the given graph G joins a set S independently with probability p_v, where p_v depends on v's class as defined in the degree-decomposition step. This *vertex-selection step* yields a set S that will be shown to have two properties: (i) the expected number of edges in the induced subgraph $G[S]$ is $O(n \cdot \text{poly}(\log n))$; and (ii) with high probability, every vertex in G is either in S or has a neighbor in S. Given the degree-decomposition, the vertex-selection step is elementary and requires no communication. In the third step, we use the 2-ruling set algorithm of Berns et al. [7, 6]. We show that, on an n-node graph with $O(n \cdot \text{poly}(\log n))$ edges, this algorithm runs in expected-$O(\log\log\log n)$ rounds. We will refer to this algorithm from [7, 6] as the *2-ruling set algorithm*. Putting these three steps together yields a 3-ruling set algorithm that runs in $O(\log\log\log n)$ rounds in expectation.

2.3.1 Degree-Decomposition Step

Let $G = (V, E)$ be an arbitrary graph. Let U_1 be the set of all nodes in G with degrees in the range $[n^{1/2}, n)$. Let V_1 be the remaining nodes, i.e., $V_1 = V \setminus U_1$. Let U_2 be the set of all nodes in V_1 with degrees in $G[V_1]$ belonging to the range $[n^{1/4}, n^{1/2})$. The decomposition continues in this manner until V is partitioned into sets $U_1, U_2, \ldots$. We now provide a more formal description. For $k = 0, 1, 2, \ldots$, let $D_k = n^{1/2^k}$. The D_k's will serve as degree thresholds and will lead to a vertex partition. Let $k^* = \lceil \log \log n \rceil$. Note that $1 < D_{k^*} \leq 2$. Let $V_0 = V$, $G_0 = G$, and $U_1 = \{v \in V_0 \mid \text{degree}_{G_0}(v) \in [D_1, D_0)\}$. For $1 \leq k < k^*$, let

$$V_k = V_{k-1} \setminus U_k, \qquad G_k = G[V_k], \qquad U_{k+1} = \{v \in V_k \mid \text{degree}_{G_k}(v) \in [D_{k+1}, D_k)\}$$

Let $V_{k^*} = V_{k^*-1} \setminus U_{k^*}$, $G_{k^*} = G[V_{k^*}]$, and $U_{k^*+1} = V_{k^*}$. See Figure 2.1 for an illustration of this decomposition. Let $N_G(v)$ denote the set of neighbors of vertex v in graph G. Here are some easy observations:

(i) For $0 \leq k \leq k^*$, $\Delta(G_k) < D_k$.

(ii) For $1 \leq k \leq k^* + 1$, if $v \in U_k$ then $|N_G(v) \cap V_{k-1}| < D_{k-1}$.

(iii) For $1 \leq k \leq k^* + 1$, if $v \in U_k$ then $|N_G(v) \cap U_j| < D_j$ for $j = 1, 2, \ldots k - 1$.

Now we describe algorithm to compute this degree-decomposition; in particular, we precisely describe how each node v computes an index $k(v) \in [k^* + 1]$ such that $v \in U_{k(v)}$. Below, we first describe at a high level a 2-phase approach that we use to compute the index $k(v)$ for each vertex v. Subsequently we will flesh out our approach with necessary details and show that it is correct and can be implemented

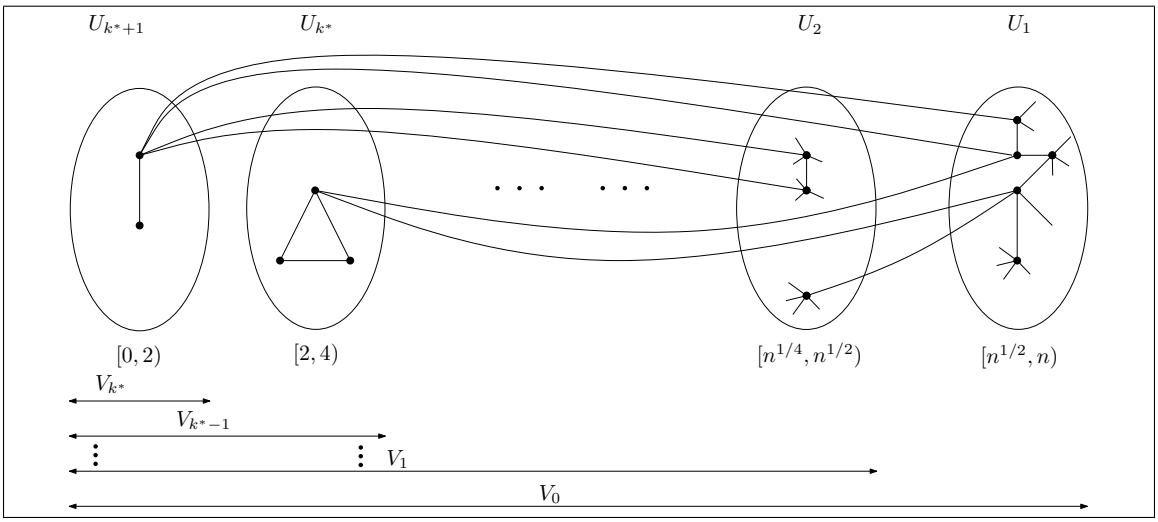


Figure 2.1: Degree-Decomposition Step. U_1 is the set of all nodes in G with degrees in the range $[n^{1/2}, n)$ and V_1 is the remaining nodes. U_2 is the set of all nodes in V_1 with degrees in $G[V_1]$ belonging to the range $[n^{1/4}, n^{1/2})$. The decomposition continues in this manner until all nodes belong to some U_k. We use k^* to denote $\lceil \log \log n \rceil$. Assuming that $\log \log n = k^*$, we see that U_k^* is the set of nodes that have degree in $G[V_{k^*-1}]$ in the range $[2, 4)$. Note that a node v that belongs to U_{k+1} could have degree in G that is much larger than $D_k = n^{1/2^k}$.

in $O(\log\log\log n)$ rounds on a Congested Clique.

Lazy phase: Let $t = \lceil 1 + \log\log\log n \rceil$. The sets $U_1, U_2, \ldots, U_t$ are identified in a leisurely manner, one-by-one, in $O(\log\log\log n)$ rounds. At the end of this phase each vertex $v \in \cup_{i=1}^{t} U_i$ knows the index $k(v) \in [t]$ such that $v \in U_{k(v)}$.

Speedy phase: The set of remaining vertices, namely V_t, induces a graph G_t whose maximum degree is less than

$$D_t \leq n^{1/2^{1+\log\log\log n}} = n^{1/(2\log\log n)}.$$

This upper bound on the maximum degree helps us compute the index values $k(v)$ for the remaining vertices at a faster rate. We first show that each vertex v in G_t can acquire knowledge of the graph induced by the ball $B_{G_t}(v, k^*)$ in $O(\log\log\log n)$ rounds via a fast *ball-growing algorithm*. (Recall that $k^* = \lceil \log\log n \rceil$.) We then show that $G[B_{G_t}(v, k^*)]$ contains enough information for v to determine $k(v) \in [k^* + 1]$ via local computation. Therefore, after each vertex $v \in V_t$ acquires complete knowledge of the radius-k^* ball centered at it, it can locally compute index $k(v)$ and proceed to the vertex-selection step.

We now present the *Lazy-phase algorithm* executed by all vertices $v \in G$.

Algorithm 2.1 Lazy-phase algorithm at vertex v

1. $k(v) \leftarrow 0$
2. **for** $i \leftarrow 1$ **to** t **do**
3. $\quad s(v) \leftarrow |\{u \in N_G(v) \mid 1 \leq k(u) < i\}|$
4. $\quad$ **if** $degree_G(v) - s(v) \in [D_i, D_{i-1})$ **then**
5. $\quad\quad k(v) \leftarrow i$
6. $\quad\quad$ Send $k(v)$ to all neighbors
7. $\quad\quad$ **break**
8. $\quad$ **end if**
9. **end for**

Lemma 2.1. *The Lazy-phase algorithm runs in $O(\log\log\log n)$ rounds and at the end of the algorithm, for each vertex $v \in \cup_{j=1}^{t} U_j$, $k(v)$ has a value in $[t]$ such that $v \in U_{k(v)}$. For any vertex $v \notin \cup_{j=1}^{t} U_j$, $k(v)$ is set to 0.*

Proof. Given that the sets $U_1, U_2, \ldots, U_i$ have been determined, and that the members of each are known to every node in the network, each node can locally determine its degree in $G_i = G[V_i]$ and thus determine its membership in U_{i+1}. Each node can then broadcast whether or not it has joined U_{i+1}, thus providing knowledge of U_{i+1} to every node in the network. It follows that the implementation of the Lazy-phase algorithm requires exactly $t = \lceil 1 + \log\log\log n \rceil$ rounds of communication to complete. □

We now present the *Speedy-phase algorithm* executed by vertex v. Note that the Speedy-phase algorithm is only executed at vertices v for which $k(v)$ is 0 after the Lazy-phase algorithm. In other words, the Speedy-phase algorithm is only executed at vertices v in G_t, the graph induced by vertices not in $\cup_{j=1}^{t} U_j$. The key idea of the

Speedy-phase algorithm is that once each node v in G_t has acquired knowledge of $G_t[B_{G_t}(v,r)]$, then in constant rounds of communication, each node v can "double" its knowledge, i.e., acquire knowledge of $G_t[B_{G_t}(v,2r)]$. This is done by each node v sending knowledge of $G_t[B_{G_t}(v,r)]$ to all nodes in $B_{G_t}(v,r)$; the key is to establish that this volume of communication can be achieved on a Congested Clique in constant rounds. This idea has appeared in a slightly different context in [38].

Algorithm 2.2 Speedy-phase algorithm at vertex v

1. ▷ Growing the ball $B_{G_t}(v,k^*)$
2. Each node sends a list of all of its neighbors in G_t to each of its neighbors (in G_t) ▷ After which each $v \in V_t$ knows $G[B_{G_t}(v,1)]$
3. **for** $i \leftarrow 0$ **to** $\lceil \log\log\log n \rceil - 1$ **do**
4. Send a description of $G[B_{G_t}(v,2^i)]$ to all nodes in $B_{G_t}(v,2^i)$
5. Construct $G[B_{G_t}(v,2^{i+1})]$ from $G[B_{G_t}(u,2^i)]$ received from all $u \in B_{G_t}(v,2^i)$
6. **end for**
7. Locally compute $k(v) \in [k^*+1]$ such that $v \in U_{k(v)}$

Lemma 2.2. *The Speedy-phase algorithm above runs in $O(\log\log\log n)$ rounds in the congested-clique model and when this algorithm completes execution, each vertex v in G_t knows $G[B_{G_t}(v,k^*)]$.*

Proof. Line 2 of the Speedy-phase algorithm can be completed in a constant number of rounds using Lenzen's routing protocol because each node needs only to send and receive $O(D_t)$ messages to/from $O(D_t)$ neighbors (each message listing a neighbor and destined for a neighbor), as the maximum degree of G_t is less than D_t.

In implementing the Speedy-phase algorithm, the key step is to perform Line 4 in $O(1)$ rounds of communication. If this can be done, then after $O(\log\log\log n)$ rounds, each node v remaining in V_t will have knowledge of its entire neighborhood graph out to a distance of $2^{\lceil \log\log\log n\rceil} \geq \lceil \log\log n\rceil = k^*$ hops away from v.

Since G_t has maximum degree less than D_t, the neighborhood graph $G[B_{G_t}(v, 2^i)]$ can be completely described by listing all $O(D_t^{2^i+1})$ edges. Thus, such a neighborhood can be communicated from v to another node (in particular, to any other node in $B_{G_t}(v, 2^i)$) via $O(D_t^{2^i+1}) = O(n^{(2^i+1)/2^t})$ messages of size $O(\log n)$. Therefore, to perform a given iteration of Line 4 within the Speedy-phase algorithm, each node will need to send (and receive) $O(n^{(2^i+1)/2^t})$ messages (of size $O(\log n)$) to $O(D_t^{2^i}) = O(n^{2^{i-t}})$ other nodes in the network. As above, we can use Lenzen's routing protocol to perform this task in $O(1)$ rounds as long as the total number of messages to be sent (and received) by each node is $O(n)$.

Thus, Line 4 of the Speedy-phase algorithm can be executed in a constant number of rounds if $n^{(2^{i+1}+1)/2^t} = O(n)$; in other words, if $2^{i+1}+1 \leq 2^t$, or $i \leq t-2 = \lceil \log\log\log n\rceil - 1$. This lower bound on the maximum value of i that still allows Line 4 to be completed in $O(1)$ rounds is precisely the final index in the for-loop (Line 3). This completes the proof. □

Lemma 2.3. *For any graph H and a vertex v in H, suppose that v knows the graph induced by $B_H(v, k^*)$. Then v can locally compute the index $k(v) \in [k^*+1]$ such that $v \in U_{k(v)}$.*

Proof. The proof is by induction. Whether a vertex u is in U_1 is determined by its

degree in H. Since v knows $H[B_H(v, k^*)]$ it can determine via local computation which $u \in B_H(v, k^* - 1)$ belong to U_1 and which don't. As the inductive hypothesis, suppose that for some $i \geq 1$, v has determined for all $u \in B_H(v, k^* - i)$ the following information:

(i) if $u \in \cup_{j=1}^{i} U_j$, then v knows $k(u) \in [i]$ such that $u \in U_{k(u)}$.

(ii) if $u \notin \cup_{j=1}^{i} U_j$, then v knows that $u \notin \cup_{j=1}^{i} U_j$.

Now consider a vertex $u \in B_H(v, k^* - i - 1)$ such that $u \notin \cup_{j=1}^{i} U_j$. In order to determine if $u \in U_{i+1}$, vertex v needs to check if the *residual degree* of u, defined as

$$r(u) := degree_H(u) - |N_H(u) \cap (\cup_{j=1}^{i} U_j)| \tag{2.1}$$

belongs to the interval $[D_{i+1}, D_i)$. In other words, we need to check that the degree of u after we have deleted all neighbors in $\cup_{j=1}^{i} U_j$ is in the range $[D_{i+1}, D_i)$. Given the information that v knows about all $u \in B(v, k^* - i)$ (by the inductive hypothesis), vertex v can compute the residual degree $r(u)$ for each $u \in B_H(v, k^* - i - 1)$. Therefore for all such u, vertex v can determine if $u \in U_{i+1}$ or not. This completes the inductive step of the proof.

Now since $B_H(v, 0) = \{v\}$, it follows from the above inductive argument that v can determine the index $k(v) \in [k^* + 1]$ such that $v \in U_{k(v)}$. $\square$

2.3.2 Vertex-Selection Step

Algorithm 2.3 Vertex-Selection Step

if $v \in U_k$ for $k = 1, 2, \ldots, k^*$ **then**

v is selected with probability $\min\left(\frac{2 \log n}{D_k}, 1\right)$

end if

if $v \in U_{k^*+1}$ **then**

v is selected with probability 1

end if

As mentioned earlier, the vertex-selection step randomly and independently samples nodes in G, with each node v sampled with a probability p_v that depends on the class $U_{k(v)}$ it belongs to. Specifically, if v belongs to U_k then v is independently selected with probability $\min(2 \log n / D_k, 1)$. Algorithm 2.3 shows pseudocode for the vertex-selection step. Let S be the set of vertices that are selected. Let $e(S)$ denote the set of edges in the induced graph $G[S]$.

Lemma 2.4. $E[|e(S)|] = O(n \cdot \log^2 n \cdot \log \log n)$.

Proof. Consider an arbitrary vertex $v \in V$ and let k, $1 \leq k \leq k^* + 1$ be such that $v \in U_k$. We will show that the expected number of edges between v and nodes in $\cup_{j \leq k} U_j$ is less than $4k \cdot \log^2 n$.

In the graph G, node v has fewer than D_{k-1} neighbors in U_k. Thus, if $1 \leq$

$k \leq k^*$, the expected number of edges in $e(S)$ between v and nodes in U_k is at most

$$\frac{2\log n}{D_k} \cdot \sum_{u \in N_G(v) \cap U_k} \frac{2\log n}{D_k} < \frac{4\log^2 n \cdot D_{k-1}}{D_k^2} = 4\log^2 n.$$

If $k = k^* + 1$, the number of edges between v and other nodes in U_{k^*+1} is at most 1.

In the graph G, node v has fewer than D_j neighbors in U_j, for $j < k$. Thus, if $1 \leq k \leq k^*$, the expected number of edges in $e(S)$ between v and nodes in U_j, $j < k$, is at most

$$\frac{2\log n}{D_k} \cdot \sum_{u \in N_G(v) \cap U_j} \frac{2\log n}{D_j} < \frac{4\log^2 n}{D_k} \leq 4\log^2 n.$$

If $k = k^* + 1$, the expected number of edges in $e(S)$ between v and nodes in U_j, $j < k$, is

$$1 \cdot \sum_{u \in N_G(v) \cap U_j} \frac{2\log n}{D_j} < 2\log n.$$

Hence, summing over j, the expected total number of edges in $e(S)$ between v and $\cup_{j \leq k} U_j$ is less than $4k \cdot \log^2 n$. Using the fact that $k \leq 1 + \log\log n$, we see that the expected total number of edges in $e(S)$ between v and $\cup_{j \leq k} U_j$ is $O(\log^2 n \cdot \log\log n)$. The result follows. □

Lemma 2.5. *For any $v \in V$, $\Pr(v$ is in S or v has a neighbor in $S) \geq 1 - 1/n^2$.*

Proof. Suppose that $v \in U_k$, for some $1 \leq k \leq k^*$. Vertex v has at least D_k neighbors in V_{k-1}. Each such neighbor is selected for S with probability at least $\min\{(2\log n)/D_k, 1\}$. If $2\log n \geq D_k$, than any of these neighbors is selected for S with probability 1, so v has a neighbor in S with probability 1. Otherwise, we have

$$\Pr(v \text{ has no neighbor in } S) \leq \left(1 - \frac{2\log n}{D_k}\right)^{D_k} \leq e^{-2\log n} \leq \frac{1}{n^{2.8}}$$

Also, if $v \in U_{k^*+1}$, then v is selected for S with probability 1. □

2.3.3 2-Ruling Set Algorithm

We now briefly describe the 2-ruling set of Berns et al. [7, 6] that runs on a Congested Clique in expected $O(\log \log n)$ rounds. This text is borrowed largely from [7, 6] and the reader is advised to consult these papers for a more detailed description. Pseudocode for the algorithm appears in Algorithm 2.4. The algorithm proceeds in *Iterations* and in each Iteration some number of nodes become inactive and we measure progress by the number of edges remaining in the graph induced by active nodes. In an Iteration i, each active node (S denotes the set of active nodes) joins a "Test" set T independently with probability $q = \sqrt{\frac{n}{m}}$ (Line 6), where m is the number of edges in the graph induced by active nodes. The probability q is set such that the expected number of edges in $G[T]$ is equal to n. If the number of edges in $G[T]$ is no more than $4n$, then we can ship off $G[T]$ to a single node and have that node locally compute an MIS. All of this takes a constant number of rounds and then we delete T and its neighborhood $N(T)$ from the active set S. (Lines 7-10). Because m, the number of edges in $G[S]$, decreases, the probability q rises (Line 12) while still having the expected number of edges in $G[T]$ during the next iteration bounded above by n.

Berns et al. [7, 6] have analyzed this algorithm to show that it requires expected $O(\log \log n)$ rounds. We now sketch this analysis to observe that for n-node graphs with $O(n \cdot \text{poly}(\log n))$ edges, this 2-ruling set algorithm requires an expected $O(\log \log \log n)$ rounds.

Lemma 2.6. *Given an n-vertex graph G with $O(n \cdot \text{poly}(\log n))$ edges, Algorithm 2.4,*

Algorithm 2.4 2-RulingSet

Input: (G, S)

Output: A 2-ruling set R of $G[S]$

1. $R \leftarrow \emptyset$
2. $m \leftarrow e[G[S]]$ (Each node x broadcasts its degree in $G[S]$ to all others)
3. $q \leftarrow \sqrt{\frac{n}{m}}$
4. **while** $m > 2n$ **do**
5. $\quad T \leftarrow \emptyset$
6. $\quad$ Each $x \in S$ joins T independently with probability q and broadcasts its choice.
7. $\quad$ **if** $e[G[T]] \leq 4n$ **then**
8. $\quad\quad L \leftarrow$LocalMIS$(G[T])$
9. $\quad\quad R \leftarrow R \cup L$
10. $\quad\quad S \leftarrow S \setminus (T \cup N(T))$
11. $\quad m \leftarrow e[G[S]]$
12. $\quad q \leftarrow \sqrt{\frac{n}{m}}$
13. $L \leftarrow$LocalMIS$(G[S])$
14. $R \leftarrow R \cup L$
15. **return** R.

2-RulingSet *(derived from [7, 6]), computes a 2-ruling set of G in expected-$O(\log\log\log n)$ rounds.*

Proof. Algorithm 2.4, 2-RulingSet computes a 2-ruling set on a subgraph of a Congested Clique in expected-$O(\log\log n)$ rounds [7, 6]. The analysis of this algorithm proceeds by defining $O(\log\log n)$ threshold values $L_k = n^{1+1/2^k}$, for $k = 0, 1, 2, \ldots$, and computing a bound on the expected number of rounds required for the number

of edges remaining in the unprocessed portion of the graph to fall from roughly L_i to roughly L_{i+1}. In Lemma 9 of [7], it is proved that this expected number of rounds is uniformly bounded by a constant for every k.

For our use of the 2-ruling set algorithm in the present work, we observe that, if the number of edges in the input subgraph (of the Congested Clique) is already sufficiently small, then the expected round-complexity of the 2-ruling set algorithm is also much less than would be the case in general. Specifically, we consider a 2-ruling set computation on an input subgraph having $O(n \cdot \text{poly}(\log n))$ edges – in this case, the computation begins having already reached threshold $L_{k'}$, where $n^{1+1/2^{k'}} \approx n \cdot \log^c n$ (for a constant c). More precisely, let $k' = \lfloor \log\log n - \log\log\log n - \log c \rfloor$; then $\frac{1}{2^{k'}} \geq \frac{c \log\log n}{\log n} = \log_n \log^c n$ and $n^{1+1/2^{k'}} \geq n \log^c n$.

Therefore, using the same analysis as occurs in the proof of Theorem 2 in [7] (and in which $\mathcal{T}(k)$ represents the number of iterations necessary to progress from having at most L_{k-1} edges remaining to at most L_k edges remaining), we see that the expected running time (in rounds) of the 2-Ruling Set algorithm applied to an input graph having only $O(n \log^c n)$ edges can be written as

$$\begin{aligned}
\mathbf{E}\left[O(1) + \sum_{k=k'}^{\log\log n} O(\mathcal{T}(k))\right] &= O(1) + \sum_{k=k'}^{\log\log n} O\left(\mathbf{E}[\mathcal{T}(k)]\right) \\
&= O(1) + \sum_{k=\lfloor \log\log n - \log\log\log n - \log c \rfloor}^{\log\log n} O(1) \\
&= O(\log\log\log n)
\end{aligned}$$

which completes the proof. □

2.3.4 Putting it all together

We now combine the algorithm for degree-decomposition step algorithm, the vertex-selection step algorithm, and the 2-ruling set algorithm in order to obtain a 3-ruling set algorithm that runs in $O(\log\log\log n)$ rounds in expectation.

Algorithm 2.5 3-Ruling Set Algorithm

1. Each node $v \in V$ uses the Lazy-phase and Speedy-phase algorithms to determine the index $k(v) \in [k^* + 1]$ such that $v \in U_{k(v)}$
2. Run the vertex-selection step to compute S
3. $I \leftarrow$ 2-RulingSet$(G[S])$

Lemma 2.7. *With probability at least* $1 - 1/n$*,* I *is a* 3*-ruling set of* G*.*

Proof. Consider a vertex $v \in V$. By Lemma 2.3.2, v has a neighbor in S with probability at least $1 - 1/n^2$. Since I is a 2-ruling set of $G[S]$, there is a node in I at distance at most 3 from v. Thus, with probability at least $1 - 1/n$ we have constructed a 3-ruling set for G. □

2.4 MIS in Growth Bounded Graphs in Constant Rounds

Given a metric space (V, d) with constant doubling dimension, we show in this section how to compute an MIS of a distance-threshold graph $G_r = (V, E_r)$, for any real $r \geq 0$, in a *constant* number of rounds on a Congested Clique. This algorithm uses a combination of deterministic and randomized techniques that do careful load balancing while maximizing parallelism in order to finish up in $O(1)$

rounds. This algorithm has two important implications. First, it implies that an $O(1)$-approximation to metric facility location on a metric of constant doubling dimension can be computed in $O(1)$ rounds in the Congested Clique model via the reduction presented in [7]. Second, via a reduction developed later in this report (in Chapter 3, Section 3.5), it also implies an $O(1)$-round $O(1)$-approximation algorithm for MST on a metric with constant doubling dimension in the Congested Clique model.

2.4.1 Simulation of the Schneider-Wattenhofer MIS algorithm

Before we describe our MIS algorithm, we describe an algorithmic tool that will prove quite useful. For some $r \geq 0$, let $G_r = (V, E_r)$ be a distance-threshold graph induced by the metric $M = (V, d)$. We know that G_r is growth-bounded and in particular the size of a largest independent set in a ball $B_{G_r}(v, r)$ for any $v \in V$ is $O(r^\rho)$, where ρ is the doubling dimension of (V, d). Schneider and Wattenhofer [54] present a deterministic $O(\log^* n)$-round algorithm to compute an MIS for growth-bounded graphs in the CONGEST model. Suppose that f is a constant such that the Schneider-Wattenhofer algorithms runs in at most $f \log^* n$ rounds (note that f depends on ρ). We can *simulate* the Schneider-Wattenhofer algorithm in the Congested Clique model by (i) having each node $v \in V$ grow a ball of radius $f \log^* n$, i.e., gather a description of the induced graph $G[B_{G_r}(v, f \log^* n)]$ and then (ii) having each node v *locally simulate* the Schneider-Wattenhofer algorithm using the description of $G[B_{G_r}(v, f \log^* n)]$. Note that since the Schneider-Wattenhofer algorithm takes at most $f \log^* n$ rounds, it suffices for each node $v \in V$ to know

the entire topology of $G[B_{G_r}(v, f \log^* n)]$ to determine if it should join the MIS. The "ball growing" step mentioned above can be implemented by using Lenzen's routing protocol as follows, provided Δ (the maximum degree of G_r) is not too large. Each node v can describe its neighborhood using at most Δ messages of size $O(\log n)$ each. Node v aims to send each of these Δ messages to every node w such that $d(v, w) \leq r \cdot f \log^* n$. In other words, v aims to send messages to all nodes in $B_M(v, r \cdot f \log^* n)$. Since $B_{G_r}(v, f \log^* n) \subseteq B_M(v, r \cdot f \log^* n)$, it follows that the messages sent by v are received by all nodes in $B_{G_r}(v, f \log^* n)$. We now bound the size of $B_M(v, r \cdot f \log^* n)$ as follows. Since M has doubling dimension ρ, the size of any MIS in $B_M(v, r \cdot f \log^* n)$ is $O((\log^* n)^\rho)$ and hence total number of nodes in $B_M(v, r \cdot f \log^* n)$ is $O(\Delta \cdot (\log^* n)^\rho)$. Therefore every node v has $O((\log^* n)^\rho \cdot \Delta^2)$ messages to send, each of size $O(\log n)$. Every node is the receiver of at most $O((\log^* n)^\rho \Delta^2)$ messages by similar arguments. Therefore, if $\Delta = O(\sqrt{n}/(\log^* n)^{\rho/2})$, we can use Lenzen's routing protocol to route these messages in $O(1)$ time. We refer this simulation of the Schneider-Wattenhofer algorithm [54] as Algorithm SW-MIS. The following theorem summarizes this simulation result.

Theorem 2.8. *If* $\Delta(G_r) = O(\sqrt{n}/(\log^* n)^{\rho/2})$ *then Algorithm* SW-MIS *computes an MIS of* G_r *in* $O(1)$ *rounds on a Congested Clique.*

2.4.2 Constant-Round MIS Algorithm

Our MIS algorithm consists of 4 phases. Next we describe, at a high level, what each phase accomplishes.

Phase 1: We compute vertex-subset $P \subseteq V$ such that (i) every vertex in V is at most one hop away from some vertex in P and (ii) $G_r[P]$ has maximum degree bounded above by $c \cdot \sqrt{n}$, for some constant $c > 0$.

Phase 2: We process the graph $G_r[P]$ and compute two subsets W and Q of P such that (i) every vertex in P of degree at least $c \cdot n^{1/4}$ is either in W or has a neighbor in W and (ii) $Q \subseteq W$ is an independent set such that every vertex in W is at most 2 hops from some vertex in Q. Thus, if we delete W and all neighbors of vertices in W what remains is a graph of maximum degree less than $c \cdot n^{1/4}$. Let V' denote the set $P \setminus (W \cup N(W))$. Thus, at the end of Phase 2, Q is a 3-ruling set of $G_r[W \cup N(W)]$ and $\Delta(G_r[V']) < c \cdot n^{1/4}$.

Phase 3: We compute an MIS R of the graph $G_r[V']$ by simply calling SW-MIS.

Phase 4: Since Q is a 3-ruling set of $G_r[W \cup N(W)]$ and R is an MIS of $G_r[V']$, we see that $Q \cup R$ is a 3-ruling set of $G_r[P]$ and thus a 4-ruling set of G_r. In the final phase, we start with the 4-ruling set $Q \cup R$ and expand this into an MIS I of G_r.

Phase 2 is randomized and runs in constant rounds w.h.p. The remaining phases are deterministic and run in constant rounds each. Algorithm LowDimensionalMIS summarizes our algorithm. We now describe each phase in more detail.

2.4.3 Phase 1: Reduce Degree to $O(\sqrt{n})$

Algorithm ReduceDegree describes Phase 1 of our algorithm. The algorithm consists of arbitrarily partitioning the vertex-set of G_r into $\sqrt{n}$ groups of size (roughly)

Algorithm 2.6 LowDimensionalMIS

Input: $G_r = (V, E_r)$

Output: A maximal independent set $I \subseteq V$ of G_r

1. $P \leftarrow$ ReduceDegree(G_r) ▷ Phase 1
2. $(W, Q) \leftarrow$ SampleAndPrune(G_r, P) ▷ Phase 2
3. $V' \leftarrow V \setminus (W \cup N(W))$; $R \leftarrow$ SW-MIS(G_r, V') ▷ Phase 3
4. $S \leftarrow Q \cup R$; $I \leftarrow$ RulingToMIS(S) ▷ Phase 4
5. **return** I

Algorithm 2.7 ReduceDegree (Phase 1)

Input: $G_r = (V, E_r)$

Output: $P \subseteq V$ such that (i) $V = P \cup N(P)$ and (ii) $\Delta(G_r[P]) < c \cdot \sqrt{n}$ for some constant $c > 0$.

1. Partition V (arbitrarily) into $\lceil \sqrt{n} \rceil$ subsets: $V_1, V_2, \ldots V_{\lceil \sqrt{n} \rceil}$, each of size at most $\sqrt{n}$
2. **for all** $i \leftarrow 1$ **to** $\lceil \sqrt{n} \rceil$ **in parallel do**
3. Send $G_r[V_i]$ to a vertex v_i with lowest ID in V_i
4. Vertex v_i executes $P_i \leftarrow$ LocalMIS$(G_r[V_i])$
5. **end for**
6. $P \leftarrow \cup_{i=1}^{\lceil \sqrt{n} \rceil} P_i$
7. **return** P

$\sqrt{n}$ each and then separately and in parallel computing an MIS of each part. Since each part has $\sqrt{n}$ vertices, each part induces a subgraph with at most n edges and therefore each such subgraph can be shipped off to a distinct node and MIS on each subgraph can be computed locally. (The subroutine LocalMIS in Line 4 refers to an unspecified MIS algorithm that is executed locally at a node.) Using the fact that

G_r is growth-bounded, we show that the union of all the MIS sets (set P, Line 5) induces a graph with maximum degree bounded by $c \cdot \sqrt{n}$ for some constant c. Also, we show that Phase 1 runs in constant rounds (Lemma 2.9).

Lemma 2.9. *Algorithm* ReduceDegree *completes in $O(1)$ rounds and returns a set P such that $\Delta(G_r[P]) < c \cdot n^{1/2}$ for some constant $c > 0$ (that depends on the doubling dimension of the underlying space).*

Proof. Algorithm ReduceDegree starts by arbitrarily partitioning V into $\lceil\sqrt{n}\rceil$ disjoint subsets $V_1, \ldots, V_{\lceil\sqrt{n}\rceil}$ each of size at most $\sqrt{n}$ which can be done in $O(1)$ rounds easily. Since $|V_i| \leq \sqrt{n}$, $G_r[V_i]$ contains at most n edges, for any $i \in \lceil\sqrt{n}\rceil$. Using Lenzen's routing protocol, all knowledge of $G_r[V_i]$ can be shipped off to a designated vertex v_i in V_i (e.g., vertex with smallest ID in V_i) in $O(1)$ rounds. The vertex v_i then computes an MIS P_i of $G_r[V_i]$ locally as shown in Line 4 of Algorithm ReduceDegree. Finally, v_i informs vertices in P_i of their selection into the MIS. The union of the P_i's, denoted P, is returned by the algorithm. This discussion shows that Algorithm ReduceDegree completes in $O(1)$ rounds.

Consider a vertex $u \in P_i$ for some $i \in [\lceil\sqrt{n}\rceil]$. In $G_r[P]$, vertex u cannot have neighbors in P_i since P_i is an independent set in $G_r[P]$. Consider a set P_j, $j \neq i$. The distance between any two vertices in $N(u) \cap P_j$ must be more than r (these nodes are independent) and it must be at most $2r$ (by the triangle inequality). Since the underlying metric space has doubling dimension ρ, it follows that $|N(u) \cap P_j| \leq 2^\rho$. Hence the degree of u in $G_r[P]$ is bounded above by $2^\rho \cdot (\lceil\sqrt{n}\rceil - 1)$. The result follows. □

Algorithm 2.8 SampleAndPrune (Phase 2)

Input: (G_r, P)

Output: (W, Q), $W \subseteq P$ such that $\{v \in P \mid \text{degree}_{G_r[P]}(v) \geq n^{1/4}\} \subseteq W \cup N(W)$; independent set $Q \subseteq W$ such that Q is a 2-ruling set of $G_r[W]$.

1. **for all** $v \in P$ **in parallel do**
2. Vertex $v \in P$ adds itself to W_i with probability $1/n^{1/4}$ for $i = 1, 2, \ldots, \lceil 2 \cdot \log n \rceil$.
3. **end for**
4. $W \leftarrow \cup_{i=1}^{\lceil 2 \log n \rceil} W_i$
5. **for all** $i \leftarrow 1$ **to** $\lceil 2 \log n \rceil$ **in parallel do**
6. Send $G_r[W_i]$ to a vertex w_i, where w_i is the vertex of rank i in the sequence of vertices in V sorted by increasing ID
7. Vertex w_i executes $X_i \leftarrow$ LocalMIS$(G_r[W_i])$
8. **end for**
9. $Q \leftarrow$ SW-MIS$(G_r[\cup_{i=1}^{\lceil 2 \log n \rceil} X_i])$
10. **return** (W, Q)

2.4.4 Phase 2: Sample and Prune

Algorithm SampleAndPrune implements Phase 2 of our MIS algorithm. It takes the induced subgraph $G_r[P]$ as input and starts by computing a set $W \subseteq P$ using a simple random sampling approach. Specifically, for each $i = 1, 2, \ldots, \lceil 2 \cdot \log n \rceil$, each vertex in P simply adds itself to a set W_i independently, with probability $1/n^{1/4}$. We start by proving a useful property of W.

Lemma 2.10. *Every node u with degree at least $n^{1/4}$ in $G_r[P]$ has a neighbor in W with probability at least $1 - \frac{1}{n^2}$.*

Proof. Let $u \in P$ be a node with degree at least $n^{1/4}$ in $G_r[P]$. For any neighbor v

of u, $\Pr(v \notin W) \leq \left(1 - \frac{1}{n^{1/4}}\right)^{\lceil 2 \log n \rceil}$. Therefore the probability that no neighbor of u is in W is at most $\left(1 - \frac{1}{n^{1/4}}\right)^{\lceil 2 \log n \rceil \cdot n^{1/4}}$. This is bounded above $e^{-\lceil 2 \log n \rceil}$, which is bounded above by $1/n^2$. □

After using random sampling to compute W, Algorithm SampleAndPrune then "prunes" W in constant rounds to construct a subset $Q \subseteq W$ such that Q is a 2-ruling set of W. In the rest of this subsection we prove that Algorithm SampleAndPrune does behave as claimed here.

Lemma 2.11. *The number of edges in $G_r[W_i]$ is $O(n)$ w.h.p., for each $i = 1, 2, \ldots, \lceil 2 \log n \rceil$.*

Proof. We first bound the size of the set W_i and the maximum degree of $G_r[W_i]$ for any $i = 1, 2, \ldots, \lceil 2 \log n \rceil$. Observe that $\mathbf{E}[|W_i|] = n^{3/4}$ and since nodes join W_i independently, an application of Chernoff's bound [14] yields $\Pr(|W_i| \leq 6n^{3/4}) \geq 1 - \frac{1}{n^2}$. To bound $\Delta(G_r[W_i])$ we use the fact that degree of any node in $G_r[P]$ is at most $\sqrt{n}$ and therefore the expected degree of any node in $G_r[W_i]$ is at most $n^{1/4}$. Another application of Chernoff's bound yields $\Pr(\text{degree}_{G_r[W_i]}(v) \leq 6n^{1/4}) \geq 1 - \frac{1}{n^2}$ for each node v. Using the union bound over all nodes $v \in W_i$ yields that with probability at least $1 - \frac{1}{n}$ every node in $G_r[W_i]$ has degree at most $6n^{1/4}$. Hence, with high probability, the number of edges in $G[W_i]$ is at most $36n$. □

Lemma 2.12. *The set $X := \cup_{i=1}^{\lceil 2 \log n \rceil} X_i \subseteq P$ is computed in constant rounds w.h.p. in Lines 4-6 of Algorithm* SampleAndPrune. *Furthermore, Every vertex in W is at most one hop away from some vertex in X.*

Proof. We argue that Line 5 can be implemented in $O(1)$ rounds w.h.p. By Lemma 2.11, each node has to send at most $O(n^{1/4})$ messages to w_i and w.h.p. eachw_i receives at most $O(n)$ messages. Therefore by Lenzen's routing protocol Line 5 takes $O(1)$ rounds. To repeat this for each $i = 1, 2, \ldots, \lceil 2 \log n \rceil$ in parallel, every node has to send at the most $\lceil 2 \log n \rceil \cdot n^{1/4}$ messages. Since w_i's are distinct no w_i needs to receive more than $O(n)$ messages.

Each $v \in W$ belongs to W_i for some i and is therefore at most one hop from some vertex in X_i. □

Lemma 2.13. *W.h.p. it takes constant number of rounds to compute Q. Furthermore, Q is a 2-ruling set of $G_r[W]$.*

Proof. Consider a node $v \in \cup_{i=1}^{\lceil 2 \log n \rceil} X_i$. Since each X_i is an independent set, by using the growth-bounded property of $G_r[X_i]$, we see that the number of neighbors of v in X_i is bounded above by a constant. Hence, the maximum degree in $G_r \left[\cup_{i=1}^{\lceil 2 \log n \rceil} X_i\right]$ is $O(\log n)$. Since the maximum degree of this growth-bounded graph is $O(\log n)$, by Theorem 2.8 an MIS of this graph can be computed in constant rounds by using SW-MIS.

A node $v \in W$ belongs to some W_i and is therefore at most one hop from some node in X_i. Also, every node in every X_i is at most one hop from some node in Q. Also, Q is independent and therefore Q is a 2-ruling set of $G_r[W]$. □

2.4.5 Phase 4: Ruling Set to MIS

Algorithm RulingToMIS implements Phase 4 of our MIS algorithm. The algorithm takes as input the graph G_r and the vertex subset $S = Q \cup R$ where Q and R are the outputs of Phase 2 and Phase 3, respectively. Note that Lemma 2.13 implies that S is a 4-ruling set of G_r. This property is used to cover G_r with balls of radius $4r$, centered at members of S.

Consider the graph $G_{9r} = (V, E_{9r})$ where $E_{9r} = \{\{u, v\} \mid u, v \in V \text{ and } d(u, v) \leq 9r\}$. In Lemma 2.14 we prove a constant upper bound on the maximum degree $\Delta(G_{9r}[S])$. This allows us to compute a proper vertex coloring of $G_{9r}[S]$ using a constant number of colors. This coloring guides the rest of the algorithm, providing a schedule for processing the vertices in the aforementioned balls centered at vertices in S. For each color i, the algorithm processes all vertices in S colored i in parallel. For each vertex $v \in S$ colored i, let B_v denote the subset of $B(v, 4r)$ of vertices still "active". The algorithm computes an MIS of the induced subgraph $G_r[B_v]$; this computation occurs in parallel for each v colored i. Since the vertex coloring is with respect to G_{9r}, two balls B_v and $B_{v'}$ that are processed in parallel do not intersect and in fact are not even connected by an edge. Thus processing in parallel all of the balls B_v for v colored i has no untoward consequences. We note that due to the growth bounded property, every independent set of $G_r[B_v]$ has a constant number of vertices. Hence, we can use a simple sequential algorithm to compute an MIS of $G_r[B_v]$ – repeatedly each vertex with smallest ID in its neighborhood joins the MIS and the graph is updated. We call this MIS algorithm

Algorithm 2.9 RulingToMIS (Phase 4)

Input: $(G_r, S = Q \cup R)$

Output: A maximal independent set $I \subseteq V$ of G_r

1. $E_{9r} \leftarrow \{\{u, v\} \mid u, v \in S \text{ and } d(u, v) \leq 9r\}$
2. $G_{9r}[S] \leftarrow (S, E_{9r})$
3. Send $G_{9r}[S]$ to a vertex v^* with lowest ID in S
4. Vertex v^* executes $\Psi \leftarrow$ LocalColoring$(G_{9r}[S])$ with color pallet $\{1, 2, \ldots, \gamma + 1\}$. Here γ is the constant from Lemma 2.14.
5. $V' \leftarrow V$
6. **for** $i = 1$ **to** $i = \gamma + 1$ **do**
7. **for all** $v \in S$ such that $\Psi(v) = i$ **in parallel do**
8. $B_v \leftarrow \{u \mid u \in V' \text{ and } d(u, v) \leq 4r\}$
9. $I_v \leftarrow$ SequentialMIS$(G_{9r}[B_v])$
10. **end for**
11. $V' \leftarrow V' \setminus \left(\cup_{v \in S \wedge \Psi(v) = i} (N(I_v))\right)$
12. **end for**
13. $I \leftarrow \cup_{v \in S} I_v$
14. **return** I

SequentialMIS and use it in Line 9 in Algorithm RulingToMIS. Since every vertex in V is at distance at most $4r$ from some vertex in S, every vertex in V is is some ball B_v and is eventually processed.

Lemma 2.14. *$\Delta(G_{9r}[S]) \leq \gamma$, where γ is a constant.*

Proof. Consider any node $v \in S$ and neighbors $N_{G_{9r}}(v)$ of v in $G_{9r}[S]$. By the triangle inequality, any pair of nodes in $N_{G_{9r}}(v)$ are at most distance $18r$ apart and by Lemma 2.13, at least distance r apart. Hence $N_{G_{9r}}(v) \cup v$ has a constant aspect

ratio and by the growth-bounded property, we have $|N_{G_{9r}}(v) \cup v| \leq 18^{\rho} = \gamma$. □

Lemma 2.15. *Algorithm* RulingToMIS *executes in a constant number of rounds.*

Proof. Since the maximum degree of $G_{9r}[S]$ is a constant, the entire description of $G_{9r}[S]$ can be shipped to a designated vertex v^* (e.g., a vertex with the smallest ID) using Lenzen's routing protocol in $O(1)$ rounds. Then v^* can compute a coloring of $G_{9r}[S]$ such that no two adjacent vertices have the same color. Notice that the maximum degree of $G_{9r}[S]$ is bounded above by γ, hence $\gamma + 1$ colors are sufficient.

The constant upper bound on the size of the color palette implies that the **for**-loop starting in Line 6 executes a constant number of iterations. In each iteration i, all nodes $v \in S$ colored i are processed. Specifically, an MIS of $G_r[B_v]$ is computed and since the size of every independent set in $G_r[B_v]$ is bounded above by a constant (by appealing to the growth-bounded property), Algorithm SequentialMIS terminates in constant rounds. Hence, each iteration of the outer-**for**-loop takes a constant number of rounds of communication. □

Lemma 2.16. *The set I computed by Algorithm* RulingToMIS *is an MIS of G_r.*

Proof. First we show that I is an independent set by contradiction. Suppose that for some $p, q \in I$, p and q are adjacent in G_r. Then it must be the case that both p and q were selected in the same iteration of the outer-**for**-loop; otherwise, the selection of one of the two nodes would render the other unavailable for selection. If p and q are selected in the same outer-**for**-loop iteration, it must be the case that $p \in B_v$ and $q \in B_{v'}$ where $v \neq v'$, but v and v' have the same color. Since $d(p, v) \leq 4r$,

$d(q, v') \leq 4r$, and $d(p, q) \leq r$, using the triangle inequality we see that $d(v, v') \leq 9r$. But, if this is the case then there is an edge between v and v' in $G_{9r}[S]$ and these two vertices would not have the same color, contradicting our earlier conclusion that v and v' have the same color.

We now prove that I is maximal. Since S is a 4-ruling set of G_r, every node $u \in V$ is in $B(v, 4r)$ for some $v \in S$. Suppose that v is colored i and therefore B_v is processed in iteration i of the outer-**for**-loop. If $u \in B_v$ then Algorithm SequentialMIS will either pick u or a neighbor to join the MIS. Otherwise, if $u \notin B_v$ then it must be the case that in an earlier iteration of the outer-**for**-loop, either u or a neighbor were selected to be in the MIS. □

2.5 Constant-Approximation to MFL

Berns et al. [7, 6] showed how to compute a constant-factor approximation to MFL in expected $O(\log \log n)$ rounds. (The algorithm presented in [6] runs in expected $O(\log \log n \cdot \log^* n)$ rounds, but this was subsequently improved to expected $O(\log \log n)$ in [7].) A high level description of this algorithm is as follows. Each node v locally computes a value $r_v \geq 0$ that is a function of its opening cost f_v and distances to other nodes $\{d(v, w) \mid w \in V\}$. Nodes with similar r_v-values join the same class; more precisely, a node v with $3^k \cdot r_m \leq r_v \leq 3^{k+1} \cdot r_m$, joins a class V_k. Here r_m is the minimum r_u-value over all nodes $u \in V$. For nodes in each class V_k, we construct a graph $H_k = (V_k, E_k)$, where the edge-set E_k is defined as $\{\{u, v\} \mid u, v \in V_k, d(u, v) \leq r_u + r_v\}$. In the rest of the algorithm, in order to figure

out which nodes to open as facilities, the algorithm computes a t-ruling set on each graph G_k. Analysis in [7, 6] then shows that the solution to facility location produced by this algorithm is an $O(t)$-approximation. In [7] it is shown how to compute a 2-ruling set in expected $O(\log \log n)$ rounds on a Congested Clique. Since the classes V_k form a partition of the nodes, the ruling set computations occur on disjoint sets of nodes and can proceed in parallel. This leads to a constant-factor approximation to MFL in expected $O(\log \log n)$ rounds.

The 3-ruling set algorithm and the MIS algorithm in the present chapter can replace the slower 2-ruling set and this yields the following result.

Theorem 2.17. *There exists a distributed algorithm that computes a constant-approximation to the metric facility location problem (w.h.p.) in the congested-clique model and which has an expected running time of $O(\log \log \log n)$ rounds. Additionally, if the input metric space has constant doubling dimension then a constant-approximation can be computed in constant rounds (w.h.p.)*

2.6 Conclusion

In a recent paper, Drucker et al. [13] show that the Congested Clique can simulate powerful classes of bounded-depth circuits, implying that even slightly super-constant lower bounds for the Congested Clique would give new lower bounds in circuit complexity. This provides some explanation for why there are no non-trivial lower bounds in the Congested Clique model. One could view this result as providing motivation for proving even stronger upper bounds. As shown in this chapter, it is

possible to design algorithms that run significantly faster that $\Theta(\log \log n)$ rounds for well-known problems. Continuing this program, we are interested in designing algorithms running in $o(\log \log n)$ rounds for MST and related problems such as connectivity verification.

CHAPTER 3
SUPER-FAST ALGORITHMS FOR GRAPH CONNECTIVITY AND MINIMUM SPANNING TREE

3.1 Introduction

In the previous chapter, we presented a $O(\log\log\log n)$-round algorithm for the metric facility location (MFL) problem. The Congested Clique algorithms for MST due to Lotker et al. [40] and the MFL algorithm due to Berns et al. [7, 6] have $O(\log\log n)$ round complexity. Our algorithm for MFL from earlier chapter with triply-logarithmic running time, involves new techniques that seem applicable to Congested Clique algorithms in general. This raised the question about the MST problem. Can MST also be solved in $O(\log\log\log n)$ rounds on a Congested Clique? In this chapter, we affirmatively answer this question by presenting a randomized $O(\log\log\log n)$-round MST algorithm[1].

3.1.1 Related Work

Research on the Congested Clique has focused mostly on the *time complexity* (i.e., the number of synchronous *rounds*) of these problems. The complete network allows $\Theta(n^2)$ (different) messages (each message is of size $O(\log n)$ bits) to be exchanged in each round, and many of the time-efficient algorithms for various problems have exploited this vast parallel communication ability to give "super-fast" algorithms that run in a sub-logarithmic (in n) number of rounds. An important

[1]This chapter is derived from our work which appeared in 2015 [21]

early result is the work of Lotker et al. [39], which presented an $O(\log \log n)$-round deterministic algorithm for the MST problem. This was a significant improvement at the time (only an $O(\log n)$-round algorithm was known [50]). Lotker et al. left open the question of whether or not an even faster algorithm was possible—in particular, whether a *constant-round* algorithm could be possible for MST or for the (simpler) problem of graph connectivity (GC). Regarding lower bounds, almost nothing non-trivial is known. In particular, for the GC and MST problems, no *super-constant* time lower bounds are known. The situation is not promising from the lower bounds side: the recent results of [13] have proved that showing substantially super-constant lower bounds on time in the Congested Clique is as hard as proving long-open lower bounds in circuit complexity. However, this leaves open the important question of whether or not constant-time algorithms are possible for GC as well as the MST problem.

Thus far there has been little work on understanding the *message complexity* of problems in the Congested Clique. Message complexity refers to the number of messages (typically of polylogarithmic size) sent and received by all machines over the course of an algorithm; in many applications, this is the dominant cost as it plays a major role in determining the running time and auxiliary resources (e.g., energy) consumed by the algorithm. For example, communication cost is one of the dominant costs in distributed computation on large-scale data in modern data centers [31]. In the particular context of the Congested Clique, optimizing messages as well as time has direct applications to the performance of distributed algorithms in other models such as the Big Data (k-machine) model [31], which was recently

introduced to study distributed computation on large-scale graphs. The above work shows how to "convert" algorithms (cf. Conversion theorem of [31]) designed in the Congested Clique model to the Big Data model; the running time in the Big Data model depends on *both* the time *and* the message complexities of the corresponding algorithm in the Congested Clique model. As a consequence, the fastest algorithm in the Congested Clique model need not yield the fastest algorithm in the Big Data model; on the contrary, a slower but more message efficient algorithm in the Congested Clique can yield a faster algorithm in the Big Data model. Another related motivation comes from the connection between the Congested Clique model and the MapReduce model. In [22] it is shown that if a Congested Clique algorithm runs in T rounds and, in addition, has moderate message complexity, then it can be simulated in the MapReduce model in $O(T)$ rounds. Furthermore, there has been little work published on understanding the *message complexity* of problems in the Congested Clique. In particular, to the best of our knowledge, we are not aware of any work that addresses tradeoffs between message and time complexities in Congested Clique.

3.1.2 Main Results

In this chapter we focus on two fundamental graph problems in the Congested Clique, namely graph connectivity (GC) and minimum spanning tree (MST), and present several new results that make progress toward understanding the time and message complexities of randomized algorithms for these problems.

3.1.2.1 Faster Algorithms for GC and MST

Our first contribution consists of randomized (Monte Carlo) algorithms, running in $O(\log\log\log n)$ rounds and succeeding with high probability (w.h.p.), for both GC and MST (cf. Section 3.3). Our results improve by an exponential factor on the long-standing time upper bound of $O(\log\log n)$ rounds for MST due to Lotker et al. [39]. It is worth mentioning that the Lotker et al. MST algorithm is deterministic, in contrast to ours, which uses randomness in a crucial way. Our algorithms make use of several tools including sketching, random sampling, and fast sorting. We first show how to solve GC in $O(\log\log\log n)$ rounds. We do this by making use of *linear sketches* [2, 1, 42] in addition to Lotker et al.'s algorithm. The latter is used as a "pre-processing" step to first decrease the size of the graph with which we must work. Specifically, we run the algorithm of Lotker et al. for $O(\log\log\log n)$ rounds and this yields enough MST edges so that the number of connected components induced by these edges shrinks to $O(n/\operatorname{polylog} n)$. This in turn lends itself to an application of sketching. Linear sketching [2, 1, 42] is a powerful technique which is helpful in efficiently determining an outgoing edge of a component. A *sketch* of a vertex (or a component) is a short $O(\operatorname{polylog} n)$-bit vector that efficiently encodes the neighborhood of the vertex. Sampling from this sketch gives a random (outgoing) edge of this vertex (component). A critically useful property arises from the linearity of the sketches: adding the sketches of a set of vertices gives the sketch of the component induced by vertex set; the edges between the nodes within a component (i.e., the intra-component edges) are automatically "cancelled", leaving only a sketch of the

outgoing edges. After reducing the size of the graph to $O(n/\operatorname{polylog} n)$, it is possible to check connectivity locally by simply sending the sketches to a single node in the clique. We use our connectivity algorithm as a key ingredient in our MST algorithm. MST is a more-challenging problem, and (likely) cannot be solved using sketches alone or by simply collecting information at a single node. However, by leveraging the fact that our connectivity algorithm (except the Lotker et al. part) uses only $O(n)$ messages, we can run several GC subroutines in parallel. In our MST algorithm, edges are partitioned into $O(\sqrt{n})$ groups by weight and each group of edges is processed by a separate GC subroutine; therefore, up to $\Theta(\sqrt{n})$ GC subroutines could be running in parallel. We note that the runtimes of our algorithms are dominated by the pre-processing step that employs the subroutine of Lotker et al. (which takes $O(\log\log\log n)$ rounds); all other parts require only constant time.

It is worth emphasizing that if we allow $O(\operatorname{polylog} n)$ bits per message, instead of $O(\log n)$ bits per message (which is the standard bound for the Congested Clique model), then our algorithm solves MST in $O(1)$ rounds. In other words, enlarging the per-link bandwidth to $O(\operatorname{polylog} n)$ obviates the need for using the Lotker et al. MST algorithm as a pre-processing step. Lotker et al. point out that their algorithm also extends to a larger bandwidth setting; specifically, running in $O(\log 1/\varepsilon)$ rounds if each message is n^{ε} bits long. For example, this implies that the Lotker et al. algorithm would run in $O(1)$ rounds if each message was allowed to contain $\Theta(\sqrt{n})$ bits. This need for poly-sized messages should be contrasted with our MST algorithm which is capable of running in $O(1)$ rounds using only $O(\operatorname{polylog} n)$-sized messages.

3.1.2.2 Focus on Message Complexity

Unlike time complexity, when we talk about message complexity there are subtle distinctions that must be made relating to the models. In particular, the distinction between having and not having initial knowledge of neighbors' IDs is relevant. Our $O(\log \log \log n)$-round randomized MST algorithm has a message complexity of $\Theta(n^2)$ (as does the algorithm of Lotker et al. [39]). As we improved on the time complexity using randomization, a natural question is whether we can improve the message complexity as well. It turns out that the answer depends on the distinction relating to initial knowledge, mentioned above. Specifically, Hegeman et al. [21] show that $\Omega(n^2)$ messages are needed by any (randomized) algorithm (regardless of the number of rounds) to solve the GC (and hence the MST) problem if each machine in the clique has (initial) knowledge of only itself (the so-called *KT0* model, cf. Section 1.1). This result improves on the $\Omega(n^2)$ MST lower bound of Korach et al. [32], which applies only to deterministic algorithms. On the other hand, if machines begin with knowledge of their neighbors' IDs (the so-called *KT1* model, cf. Section 1.1), it turns out that by exploiting the synchronous nature of the model one can solve *any* problem fairly trivially with an algorithm that communicates only $O(n)$ bits. The $O(n)$ bits message complexity upper bound mentioned above is not particularly satisfying because the algorithm it depends on uses super-polynomially many rounds. This naturally leads to the question of whether MST (or GC) can be solved fast and using only a small number of messages. We provide a partial answer to this question in this chapter by presenting an MST algorithm in the *KT1* model

that requires only $O(n \operatorname{polylog} n)$ messages and $O(\operatorname{polylog} n)$ rounds. This result is obtained by adapting to the Congested Clique model the algorithm in [1] that combines the use of linear sketches with a standard Borůvka-type MST algorithm.

3.2 Graph Connectivity Verification in $O(\log \log \log n)$ Rounds

In this section we present a randomized algorithm in the Congested Clique model that computes an MST in $O(\log \log \log n)$ rounds, w.h.p. As a first step toward this algorithm, we present a randomized algorithm that solves GC w.h.p. in $O(\log \log \log n)$ rounds. Both algorithms use $\Theta(n^2)$ messages. Our GC algorithm constructs a *maximal spanning forest* of the input graph (i.e., a spanning forest with as many trees as the number of components in the input graph), and at the end of the algorithm every node will know such a spanning forest.

3.2.1 Linear Sketches of a Graph

A key tool used by our algorithm is *linear sketches* [1, 2, 42]. An important aspect of using sketches for connectivity is working with an appropriate graph representation. As described in [42], we use the following graph representation. For each node $v \in V$, we define the incidence vector $\mathbf{a}_v \in \{-1, 0, 1\}^{\binom{n}{2}}$ which describes the edges incident on node v as follows:

$$\mathbf{a}_v((x,y)) = \begin{cases} 0 & \text{if } \{x,y\} \notin E \\ 1 & \text{if } \{x,y\} \in E \text{ and } v = x < y \\ -1 & \text{if } \{x,y\} \in E \text{ and } x < y = v. \end{cases}$$

With this representation it is easy to see the following property of these vectors: for any subset of nodes S, the non-zero entries of $\sum_{v \in S} \mathbf{a}_v$ corresponds exactly to the edges in the cut $(S, V \setminus S)$.

Once we have this representation, the next step is to project these vectors into lower dimensional space, i.e., *sketch space*. Specifically, for each vector $\mathbf{a}_v$, we compute a random $O(\text{poly} \log n)$–dimensional sketch $\mathbf{s}_v$, such that two properties are satisfied: (i) sampling from the sketch $\mathbf{s}_v$ returns a non-zero entry of $\mathbf{a}_v$ with uniform probability (over all non-zero entries in $\mathbf{a}_v$) and (ii) when nodes in a connected component are merged, the sketch of the new "super node" is obtained by coordination-wise addition of the sketches of the nodes in the component. The first property is referred as ℓ_0-sampling in the streaming literature [9, 42, 26] and the second property is referred as linearity (hence, the name linear sketches). The graph sketches used in [1, 2, 42] rely on the ℓ_0-sampling algorithm by Jowhari et al. [26]. Sketches constructed using the Jowhari et al. [26] approach are small, using only $\Theta(\log^2 n)$ bits per sketch and are obtained by using a (random) linear projection. Specifically, the approach of Jowhari et al. [26] requires the construction of a random $O(\log^2 n) \times \binom{n}{2}$ matrix L, such that $\mathbf{s}_v = L \cdot \mathbf{a}_v$. Note that this implies that the sketch of the component obtained by merging neighboring nodes u and v is simply the sum of the sketches $\mathbf{s}_u$ and $\mathbf{s}_v$:

$$s_{u+v} = L \cdot (\mathbf{a}_u + \mathbf{a}_v) = L \cdot \mathbf{a}_u + L \cdot \mathbf{a}_v = \mathbf{s}_u + \mathbf{s}_v.$$

To ensure this linearity property all nodes need to compute the same matrix L and thus need access to shared randomness, i.e., polynomially many mutually independent random bits. Sharing this volume of information is not feasible, given how fast we

require our algorithms to be. So instead, we appeal to the ℓ_0-sampling algorithm of Cormode and Firmani [9] which requires, for the construction of the matrix L, a family of $\Theta(\log n)$-wise independent hash functions. As we make precise below, avoiding the requirement of full independence reduces the volume of information that needs to be shared considerably.

To be more precise, let $\mathcal{H}_k$ denote a family of k-wise independent hash functions. For positive real x, let $[x]$ denote the set $\{1, 2, \ldots, \lceil x \rceil\}$. Let $h : [N] \to [N^3]$ be a randomly selected hash function from $\mathcal{H}_k$, where $N = \binom{n}{2}$. For each $r \in [c \log N]$ for constant $c > 1$, let $g_r : [N] \to [2 \log N]$, be randomly selected from $\mathcal{H}_2$. Given $\mathcal{H}_k$, $k = \Theta(\log n)$ and $\mathcal{H}_2$, Cormode and Firmani show that one can construct a $O(\log^4 N) = O(\log^4 n)$-bits linear sketch $\mathbf{s}_v$ of $\mathbf{a}_v$ such that their ℓ_0-sampler succeeds with probability at least $1 - \frac{1}{N^c}$ and, conditioned on this, outputs a non-zero entry of $\mathbf{a}_v$ with probability $\frac{1}{N'} + N^{-c}$, where N' is the number of non-zero elements in $\mathbf{a}_v$. For the linearity property to hold, the same hash functions h and $\{g_r\}_r$ need to be used by all nodes v. For our purpose what this means is that the $\Theta(\log n)$-wise independent hash function h and the $O(\log n)$, pair-wise independent hash functions $\{g_r\}_r$ need to be shared among all nodes in the network. A k-wise independent hash function whose range is polynomial in n can be constructed using $\Theta(k \log n)$ mutually independent random bits [3]. Therefore, this implies that $\Theta(\log^2 n)$ mutually independent random bits are sufficient to generate h and the $\Theta(\log n)$ g_r's. Thus $\Theta(\log^2 n)$ mutually independent random bits need to be shared among all nodes in the network and this will allows every node v to construct a sketch $\mathbf{s}_v$ of size $O(\log^4 n)$. This sharing

of $O(\log^2 n)$ bits can be achieved in the following simple way. Designate $\Theta(\log n)$ nodes for generating $\lceil \log n \rceil$ random bits each. Each of these designated node then sends these $\lceil \log n \rceil$ bits (using a constant number of messages each) to all other nodes. In the applications of linear sketches to GC and MST, we need every node v to compute $t = \Theta(\log n)$ independent sketches $\mathbf{s}_v^1, \mathbf{s}_v^2, \ldots, \mathbf{s}_v^t$, such that each family $\{\mathbf{s}_v^j\}_v$, $1 \leq j \leq t$ has the linearity property. Using the simple approach describe above, $\Theta(\log^2 n)$ nodes to could designated to generate and share in $O(1)$ rounds all the mutually independent random bits needed for generating all the sketches. We summarize this in the following theorem.

Theorem 3.1. *Given a graph $G = (V, E)$, $n = |V|$, there is a Congested Clique algorithm running in $O(1)$ rounds, at the end of which every node $v \in V$ has computed an independent collection of $t = \Theta(\log n)$ sketches, $\mathbf{s}_v^1, \mathbf{s}_v^2, \ldots, \mathbf{s}_v^t$, such that each family $\{\mathbf{s}_v^j\}_v$, $1 \leq j \leq t$ has the linearity property. The size of each computed sketch is $O(\log^4 n)$ bits. The ℓ_0-sampling algorithm on each sketch $\mathbf{s}_v^j$ returns an edge in $\mathbf{a}_v$ with probability $1/(\text{non-zero entries in } \mathbf{a_v}) + n^{-2}$.*

3.2.2 Using Linear Sketches to Solve GC

In this section we describe how to utilize *linear sketches* to solve GC w.h.p. on a Congested Clique in $O(\log \log \log n)$ rounds. Our algorithm runs in two phases. Initially, the input graph can be viewed as having n components, one for each vertex. In the first phase, we reduce the number of components to $O(n/\log^4 n)$ by running the deterministic MST algorithm of Lotker et al. [39] for $O(\log \log \log n)$ rounds. Phase 2

operates on the resulting *component graph*. This is the graph whose vertices are the components computed in Phase 1 and whose edges represent adjacencies between components. Each component leader (e.g., node with minimum ID in the component) computes $\Theta(\log n)$ independent linear sketches of its neighborhood *in the component graph*. Since this graph has $O\left(n/\log^4 n\right)$ vertices and each linear sketch has size $O(\log^4 n)$ bits, the entire volume of all linear sketches at all nodes has size $O(n \log n)$ bits. Thus, if we want to send all linear sketches to a single (global) leader machine, we would have to solve a routing problem in which each sender has $O(\log^4 n)$ messages (of size $O(\log n)$ each) and the receiver (leader) is required to receive $O(n)$ messages. This problem can be solved using, for example, Lenzen's routing protocol [37], in $O(1)$ rounds. The rest of the algorithm is simply local computation by the leader followed by the leader communicating the output, which is of size $O(n)$, to all nodes in an additional $O(1)$ rounds. We now provide the most important details.

The Lotker et al. MST algorithm takes an edge-weighted clique as input. The algorithm runs in phases, taking constant number of communication rounds per phase. At the end of phase $k \geq 0$, the algorithm has computed a partition $\mathcal{F}^k = \{{F_1}^k, {F_2}^k, \ldots, {F_m}^k\}$ of the nodes of G into *clusters*, where each cluster is a connected component of the graph induced by the edges selected thus far. Furthermore, for each cluster $F \in \mathcal{F}^k$, the algorithm has computed a minimum spanning tree $T(F)$. It is worth noting that at the end of Phase k every node in the network knows the partition $\mathcal{F}^k$ and the collection $\{T(F) \mid F \in \mathcal{F}^k\}$ of trees. It is shown that at the end of phase k the size of the smallest cluster is at least $2^{2^{k-1}}$ and hence $|\mathcal{F}^k| \leq n/2^{2^{k-1}}$.

In the following, we refer to the Lotker et al. algorithm as the CC-MST algorithm. Let CC-MST(G, k) denote the execution of CC-MST on an edge-weighted clique graph G for k phases.

Theorem 3.2 (Lotker et al. [39])**.** CC-MST *computes an MST of an n-node edge-weighted clique in $O(\log \log n)$ rounds. At the end of phase k,* CC-MST *has computed a vertex-partition $\mathcal{F}^k$ and a collection of trees $\mathcal{T}^k = \{T(F) \mid F \in \mathcal{F}^k\}$ with the following properties: (i) $|F| \geq 2^{2^{k-1}}$ for all $F \in \mathcal{F}^k$, (ii) every node knows $\mathcal{F}^k$ and $\mathcal{T}^k$, and (iii) if the largest weight of an edge in $T(F)$, for cluster $F \in \mathcal{F}^k$ is w, then there is no edge with weight $w' < w$ connecting F to a different cluster $F' \in \mathcal{F}^k$.*

Algorithm ReduceComponents describes Phase 1 of our GC algorithm. The input to this algorithm is an arbitrary graph G (not a clique and not edge-weighted) and the algorithm returns a forest $\mathcal{T}_1$ and a component graph G_1 induced by the edges in this forest. After Steps 2-3 of the algorithm, each node in the network knows the forest $\mathcal{T}_1$, by Theorem 3.2. In Step 4, the subroutine BuildComponentGraph computes the component graph G_1 of the forest $\mathcal{T}_1$ using one round of communication, as follows. Each node u examines each incident edge $\{u, v\}$ and if v belongs to a different connected component, then u send a message to the component leader of v's component. (Note that if u has two neighbors v_1 and v_2 that belong to the same connected component, distinct from u's connected component, then u only sends one message to the component leader of the component containing v_1 and v_2.) Each component leader v, processes each of the messages it has received in the previous step, and if it has received a message from a node u, then it marks

the leader of u's component as a neighbor in the component graph. Thus, at the end of BuildComponentGraph, every component leader knows all neighboring component leaders in the component graph.

A tree T in forest $\mathcal{T}_1$ is called *finished* if it is a spanning tree of a connected component of G; otherwise we call T *unfinished*. Finished trees correspond to isolated nodes in the component graph and play no further role in the algorithm. (In fact, if we only wanted to verify connectivity, as opposed to computing a maximal spanning forest, we could have the algorithm stop and report "disconnected" as soon as a finished tree, not spanning the entire graph, is detected.) Unfinished trees (represented by their component leaders) can be viewed as vertices of the graph that will processed in Phase 2. Note that at the end of Algorithm ReduceComponents, it is guaranteed that every node knows the ID of the leader of the component it belongs to and every component leader knows *incident* inter-component edges. Now we prove the following lemma that bounds the number of vertices in the graph that will be processed in Phase 2 of the GC algorithm.

Lemma 3.3. *The number of unfinished trees in* $\mathcal{T}_1$ *are* $O\left(\frac{n}{\log^4 n}\right)$.

Proof. In Step 1, we build a weighted clique from the input graph G by assigning to every edge in G, the weight 1; non-adjacent pairs of vertices are assigned weight ∞. Step 2 simply executes CC-MST on this weighted clique for $\log\log\log n + 3$ iterations, which returns a set of clusters $\mathcal{F}$ and a forest $\mathcal{T}_\infty$ of trees, one spanning tree per cluster. By Theorem 3.2(i), every cluster in $\mathcal{F}$ has size at least $\log^4 n$. Now note that some edges of weight ∞ might have been selected by CC-MST to be

Algorithm 3.1 Phase 1: ReduceComponents

Input: A graph $G = (V, E)$.

Output: $\mathcal{T}_1$, a spanning forest of G with at most $O(n/\log^3 n)$ unfinished trees and G_1, the component graph induced by the edges of $\mathcal{T}_1$.

1. Assign unit weights to edges in G to obtain a weighted graph G_w; make G_w a clique by adding edges not in G and assign weight ∞ to these newly added edges.
2. $(\mathcal{F}, \mathcal{T}_\infty) \leftarrow$ CC-MST$(G_w, \lceil \log\log\log n + 3 \rceil)$
3. $\mathcal{T}_1 \leftarrow \mathcal{T}_\infty \setminus \{\{u, v\} \in E(\mathcal{T}_\infty) \mid wt(u, v) = \infty\}$
4. $G_1 \leftarrow$ BuildComponentGraph$(G, \mathcal{T}_1)$
5. **return** $(\mathcal{T}_1, G_1)$

part of $\mathcal{T}_\infty$; and in Step 3 we discard these edges. By Theorem 3.2(iii), if a tree $T \in \mathcal{T}_\infty$ contains an edge of weight ∞, it is finished because all edges incident on T and connecting to a different tree in $\mathcal{T}_\infty$ have weight ∞ (i.e., they are non-edges in G). Thus no unfinished tree in $\mathcal{T}_\infty$ contains an edge of weight ∞. This implies that no unfinished tree is fragmented in Step 3 of Algorithm ReduceComponents and thus each unfinished tree has size at least $\log^4 n$. Therefore, there can be at most $O(n/\log^4 n)$ unfinished trees. □

Phase 2 runs on the component graph G_1 returned by Phase 1. Note that G_1 has $O(n/\log^4 n)$ non-isolated nodes and the $\Theta(\log n)$, $O(\log^4 n)$-bit-sized linear sketches computed for each non-isolated node would result in a total volume of $O(n \log n)$ bits of information, which can be sent to a single node in $O(1)$ rounds using Lenzen's routing protocol. At a high level, this is what happens in Phase 2 followed by local computation of the maximal spanning forest. Phase 2 is described in more detail in Algorithm SketchAndSpan below. Let v^* denote the vertex in V with minimum ID.

Algorithm 3.2 Phase 2: SketchAndSpan

Input: $G_1 = (V_1, E_1)$, $V_1 \subseteq V$.

Output: $\mathcal{T}_2$, a maximal spanning forest of G_1

1. Each vertex $v \in V_1$ that is not isolated computes sketches ${s_v}^i$ for $i = 1, 2, \ldots, c \log n$ of its neighborhood.
2. Each vertex $v \in V_1$ sends these $c \log n$ sketches to v^*.
3. v^* uses these sketches to locally sample edges between connected components to compute a maximal spanning forest $\mathcal{T}_2$ of G_1.
4. v^* assigns each edge in $\mathcal{T}_2$ to a node in V such that each node is assigned a single edge. v^* then sends each edge to its assigned node. Each node in V then broadcast the edge it received from v^* so that all nodes now know $\mathcal{T}_2$.
5. **return** $\mathcal{T}_2$

Our final GC algorithm executes Phase 1 (Algorithm ReduceComponents) followed by Phase 2 (Algorithm SketchAndSpan). Edges in $\mathcal{T}_2$ are inter-component edges and each such edge needs to be mapped to a real edge in input graph G. For each edge $\{C_1, C_2\}$ in $\mathcal{T}_2$ the leaders of components C_1 and C_2 know edges in G that have induced edge $\{C_1, C_2\}$. One of the leaders, say the one with smaller ID, picks an edge in G corresponding to $\{C_1, C_2\}$. Leaders send all their picked edges to v^*. Denote by $\mathcal{T}_2'$ the set of the all picked edges. Since $\mathcal{T}_2$ is a forest, v^* is the target of fewer than n edges and this communication takes $O(1)$ rounds.

Theorem 3.4 (GC Algorithm)**.** *The GC problem can be solved in $O(\log \log \log n)$ rounds w.h.p. in the Congested Clique model. Furthermore, if the bandwidth of each communication link was $O(\log^5 n)$ bits, instead of $O(\log n)$ bits, then GC could be*

solved in $O(1)$ rounds.

Remark 3.5. *It is worth noting that this approach of reducing number of components and then using linear-sketch-based algorithm to solve the GC problem can be used to solve the* bipartiteness *problem in $O(\log\log\log n)$ rounds w.h.p. and also the k-*edge-connectivity *problem in $O(k\log\log\log n)$ rounds w.h.p. using the approach of Ahn et al. [1].*

3.3 Minimum Spanning Tree in in $O(\log\log\log n)$ Rounds

In this section we show how to obtain an exact solution to the MST problem on a Congested Clique. The algorithm starts (in Step 1) with a pre-processing phase in which:

(i) the number of components is reduced from n to $O(n/\log^4 n)$ using the Lotker et al. MST algorithm, similar to Phase 1 of our GC algorithm and

(ii) the number of edges is reduced to $O(n^{3/2})$ by using the classical sampling result of Karger, Klein, and Tarjan (KKT sampling) [28].

Part (ii) of the pre-processing phase runs in $O(1)$ rounds and thus the running time of this phase is dominated by Part (i), in which $O(\log\log\log n)$ rounds of the Lotker et al. MST algorithm are executed. The use of KKT sampling yields two MST subproblems, each with $O(n^{3/2})$ edges and $O(n/\log^4 n)$ vertices. Following the pre-processing phase, in the main phase of our algorithm, we solve each of the two above-mentioned MST problems in $O(1)$ rounds. At a high level, this MST algorithm partitions by edge-weight the $O(n^{3/2})$ edges in the graph into $O(\sqrt{n})$ groups of size n

each. We then solve $O(\sqrt{n})$ instances of the GC problem in parallel. We now provide details of this algorithm.

3.3.1 Pre-Processing: Reducing Number of Components and Edges

We first reduce the number of components to at most $O(n/\log^4 n)$ components by executing CC-MST for $\lceil \log\log\log n + 3 \rceil$ phases, similar to Phase 1 our GC algorithm. Let $\mathcal{T}_1$ be the spanning forest and G_1 be the component graph obtained by executing the above step. Here, we think of the component graph G_1 as being edge-weighted, with the weight of an edge connecting components C and C' set to the minimum weight of an edge between a node in C and a node in C'. By Theorem 3.2, $\mathcal{T}_1$ is a subset of a MST of G. Our goal now is to complete this MST by determining which edges in G_1 are in the MST.

Karger, Klein, and Tarjan [28] present a randomized linear-time algorithm to find a MST in an edge-weighted graph in a sequential setting (RAM model). A key component of their algorithm is a random edge sampling step to discard edges that cannot be in the MST. For completeness we state their sampling result and the necessary terminology.

Definition (F-light edge [28])**.** *Let F be a forest in a graph G and let $F(u,v)$ denote the path (if any) connecting u and v in F. Let $wt_F(u,v)$ denote the maximum weight of an edge on $F(u,v)$ (if there is no path then $wt_F(u,v) = \infty$). We call an edge $\{u,v\}$* F-heavy *if $wt(u,v) > wt_F(u,v)$, and* F-light *otherwise.*

Lemma 3.6 (KKT Sampling Lemma [28])**.** *Let H be a subgraph obtained from G*

by including each edge independently with probability p, and let F be the minimum spanning forest of H. The number of F-light edges in G is at most n/p, w.h.p.

The implication of the above lemma is that if we set $p = 1/\sqrt{n}$ then the number of sampled edges in H and the number of F-light edges in G both are $O(n^{3/2})$ w.h.p. Crucially, none of the F-heavy edges can be in an MST of G. Therefore if we compute a minimum spanning forest F of H, then we can discard all F-heavy edges and compute a minimum spanning forest of the graph induced by the remaining F-light edges in G. We have thus reduced the problem into two MST problems: (i) compute a minimum spanning forest F of H where the number of edges in H is $O(n^{3/2})$ w.h.p. and (ii) compute a minimum spanning forest of the graph induced by F-light edges in G. Note that these two problems cannot be solved in parallel since the latter problem depends on the output of the first problem. Specifically, after problem (i) has been solved we need to identify all F-light edges; these will serve as input to problem (ii). Identifying F-light edges is easy because after problem (i) has been solved every node knows F and can therefore determine which incident edges are F-light.

Algorithm 3.3 summarizes our approach. In the beginning of Algorithm Exact-MST every node knows weights of incident edges and at the end of the execution every node knows all the edges that are in the MST computed by the algorithm. The subroutine BuildComponentGraph invoked in Step 2 now builds an *edge-weighted* component graph. Like the unweighted version of BuildComponentGraph, this subroutine also runs in $O(1)$ rounds. The only difference is that each node u (in a component C) considers all edges to a component C' ($\neq C$) and informs the leader of C' about the edge between u and C' of smallest weight. After this round of communication, component leaders have enough

information to determine the smallest weight edge to every other component. The subroutine SQ-MST is called twice (in Steps 4 and 6), to compute a minimum spanning forest of a graph with $O(n/\log^4 n)$ vertices and $O(n^{3/2})$ edges. We describe SQ-MST in detail in the next subsection and show that it runs in $O(1)$ rounds w.h.p. Algorithm SQ-MST comes with the guarantee that at the end of its execution, all nodes know the MST computed by it.

Algorithm 3.3 EXACT-MST

Input: An edge-weighted clique $G(V, E)$

Output: An MST of G

1. $(\mathcal{F}, \mathcal{T}_1) \leftarrow$CC-MST$(G, \lceil \log\log\log n + 3 \rceil)$
2. $G_1 \leftarrow$ BUILDCOMPONENTGRAPH$(G, \mathcal{T}_1)$
3. $H \leftarrow$ a subgraph of G_1 obtained by sampling each edge in G_1 independently with probability $\frac{1}{\sqrt{n}}$
4. $F \leftarrow$ SQ-MST(H)
5. $E_\ell \leftarrow \{\{u, v\} \in E(G_1) \mid \{u, v\}$ is F-light$\}$
6. $\mathcal{T}_2 \leftarrow$ SQ-MST(E_ℓ)
7. **return** $\mathcal{T}_1 \cup \mathcal{T}_2$

3.3.2 Computing MST of $O(n^{3/2})$-size Graph

We now describe Algorithm SQ-MST, which computes in $O(1)$ rounds an MST of a subgraph $G' = (V', E')$ of G with $O(n^{3/2})$ edges and $O(n/\log^4 n)$ vertices. (Pseudocode appears in Algorithm 3.4.) The bounds on number of vertices and number of edges are critical to ensuring that our MST algorithm runs in $O(1)$ rounds. The algorithm starts with edges in E' being sorted, i.e., each node computes the *rank*

$r(e)$ of each incident edge e in a sorted (by edge-weights) sequence of all edges in E'. This sorting problem can be solved in $O(1)$ rounds on the Congested Clique by using Lenzen's distributed sorting algorithm [37]. Then each node partitions (in Step 2) the incident edges based on their ranks. Thus we partition E' into $O(\sqrt{n})$ sets $E_1, E_2, \ldots, E_p$ ($p = O(\sqrt{n})$) each containing n edges (E_p might have less than n edges) such that E_1 contains all the edges whose ranks are in the range 1 to n, E_2 contains the edges with ranks between $n+1$ and $2n$, and so on. Since nodes know ranks of incident edges, each node can identify, for each incident edge e, an index $i \in [p]$ such that $e \in E_i$. In the next step (Step 3) we gather each set E_i at a *guardian* node $g(i)$. This can be done in $O(1)$ rounds as well, using Lenzen's routing protocol, because (i) the number of edges incident on a node is less than n and therefore each node is the sender of less than n edges, and (ii) $|E_i| \leq n$ and therefore each node is the receiver of at most n edges. The role of a guardian node $g(i)$ is to determine which of the edges in E_i are a part of the MST. Specifically, $g(i)$ wants to know for each edge $e = \{u, v\} \in E_i$ whether there is a path between u and v in the graph induced by edges with ranks less than $r(e)$. (Note that these are edges of weight no greater than e). That is, for each edge $e \in E_i$, $g(i)$ needs to determine whether there is a path between u and v in the graph induced by edges $\cup_{k=0}^{i-1} E_k \cup \{e_\ell \in E_i \mid r(e_\ell) < r(e)\}$. Let G_i be the subgraph of G' induced by the edge set $\cup_{k=0}^{i-1} E_k$. One way to solve this problem would be for $g(i)$ to compute a maximal spanning forest $\mathcal{T}_i$ of G_i. Then $g(i)$ could locally check uv-connectivity in the graph $\mathcal{T}_i \cup \{e_\ell \in E_i \mid r(e_\ell) < r(e)\}$. Thus each $g(i)$ needs to have available a solution to GC on the graph G_i. There are $O(\sqrt{n})$

such guardians — one for each part E_i and hence the challenge is executing $O(\sqrt{n})$ instances of GC computations in parallel in the Congested Clique network.

What provides crucial help in permitting these $p = O(\sqrt{n})$ GC instances to run in parallel is that G' has $O(n/\log^4 n)$ nodes and $O(n^{3/2})$ edges. Note that since G' has $O(n/\log^4 n)$ nodes, Phase 1 of GC is not required and only Phase 2 of GC needs to be executed in parallel on $O(\sqrt{n})$ instances. The main communication step in Phase 2 of the GC algorithm is nodes sending their $\Theta(\log n)$ linear sketches to single node for local computation. Each node $v \in V'$ has a set of incident edges belonging to each graph G_i, $1 \leq i \leq p$. Thus v computes $\Theta(p \log n) = \Theta(\sqrt{n} \log n)$ different linear sketches, each of size $\Theta(\log^4 n)$ bits. Therefore, in total each node v has $O(\sqrt{n} \cdot \log^4 n)$ different $O(\log n)$-sized messages to send. On the receiver's side, a guardian $g(i)$ is the target of $O(n)$ messages of size $O(\log n)$ bits. This communication can be completed in $O(1)$ using Lenzen's routing protocol, and thus we obtain the following theorem.

Theorem 3.7. *Algorithm* Exact-MST *computes an MST of an n-node edge-weighted clique in $O(\log \log \log n)$ rounds w.h.p. on the Congested Clique. Furthermore, if the bandwidth of each communication link is $O(\log^5 n)$ bits, then MST can be computed in $O(1)$ rounds.*

3.4 MST in $O(\text{polylog}\, n)$ Rounds and $O(n \,\text{polylog}\, n)$ Messages

In this subsection we show that if we allow the use of $O(\text{polylog}\, n)$ rounds, then we can obtain an algorithm that solves MST using only $O(n \,\text{polylog}\, n)$ messages

in the Congested Clique model. This should be contrasted with our $O(\log\log\log n)$-round algorithm, which uses $\Theta(n^2)$ messages. This algorithm is an adaption of the sketch-based algorithm in [48, 1, 27].

Theorem 3.8. *An MST can be computed in the Congested Clique model in $O(\log^5 n)$ rounds using $O(n \log^5 n)$ messages (of size $O(\log n)$-bits each).*

Proof. The algorithm proceeds in $O(\log n)$ *phases*, where in each phase a minimum-weight outgoing edge (MWOE) incident on each node is selected (w.h.p.) and the resulting connected components are merged together to form a new node. Components are indicated by their component label; all nodes in a component hold the ID of the leader of that component. Initially, each node is in a component on its own.

Consider an arbitrary phase of the algorithm. Each component leader generates $O(\log^2 n)$ mutually independent random bits and sends these to each node in its component. (Recall from the description of linear sketches in Section 3.2.1 that the Cormode-Firmani [9] construction of linear sketches requires $O(\log^2 n)$ mutually independent random bits.) This communication can be done naively, taking $O(\log n)$ rounds and using a total of $O(n \log n)$ messages. Each node in the graph uses the received random bits to compute an $O(\log^4 n)$-bit linear sketch of its neighborhood with respect to the original graph. Each node in the graph then sends its sketch to its component leader, simply using $O(\log^3 n)$ rounds. Each component leader computes the sum of the received linear sketches and then samples an outgoing edge, w.h.p. Suppose that a component leader v has obtained an outgoing edge with weight w_v.

Node v sends w_v to all its followers who then delete all incident edges of weight more than w_v and obtain new, possibly smaller, neighborhoods. The entire process is repeated $O(\log n)$ times, at which point v has found a MWOE, w.h.p. Each MWOE is then sent to the node v^* with minimum ID, which then merges components, updates labels, and then informs nodes of their component labels. This completes the current phase and yields the theorem. □

We don't know whether MST problem can be solved in $O(1)$ rounds or not but in the next section we show a constant-factor approximation to MST can be obtained in $O(1)$ rounds given that the weights in the input graph are from a constant doubling dimension metric space.

3.5 Constant-Approximation to MST of Graphs in Metric Space

For a metric space (V, d), define a *metric graph* $G = (V, E)$ as the clique on set V with each edge $\{u, v\}$ having weight $d(u, v)$. In this section we present a constant-round algorithm for computing a constant-factor approximation of an MST of given metric graph $G = (V, E)$ with constant doubling dimension.

Damian et al. presented a $O(\log^* n)$-round algorithm for constructing a light spanner of a doubling dimension metric graph in the $\mathcal{LOCAL}$ model. The techniques in [10] cannot be directly used in our setting due to (i) bandwidth constraints; (ii) the sequential dependency of the construction in [10]; (iii) our need for a $O(1)$-round reduction; and (iv) our requirement for an $O(1)$-approximation to MST. Our techniques avoid these obstacles and provide an $O(1)$-round reduction in the

congested-clique setting. Our overall approach is as follows. We start by showing how to "sparsify" G and construct a spanning subgraph $\hat{G} = (V, \hat{E})$, $\hat{E} \subseteq E$, such that $wt(MST(\hat{G})) = O(wt(MST(G)))$. Thus computing an MST on $\hat{G}$ yields an $O(1)$-approximation to an MST on G. The sparsification is achieved via the construction of a collection of maximal independent sets (MIS) *in parallel* on different distance-threshold subgraphs of G. Thus we have reduced the problem of constructing a constant-approximation of an MST on the metric graph G to two problems: (i) the MIS problem on distance-threshold graphs and (ii) the problem of computing an MST of a sparse graph $\hat{G}$. Using the fact that the underlying metric space (V, d) has constant doubling dimension, we show that $\hat{G}$ has linear (in $|V|$) number of edges. As a result, problem (ii) can be easily solved in constant number of rounds by simply shipping $\hat{G}$ to a single node for local MST computation. In Chapter 2:Section 2.4, we have already shown how to compute an MIS of a distance-threshold graph in a constant doubling dimensional space on a Congested Clique in constant number of rounds. Finally, we show that due to the particular bandwidth usage of our MIS algorithm, we can run all of the requisite MIS computations in parallel in constant rounds.

3.5.1 Metric MST Algorithm

We now present our algorithm in detail; the reader is encouraged to follow along the pseudocode in Algorithm 3.5. We partition the edge set E of the metric graph into two subsets E_ℓ (*light* edges) and E_h (*heavy* edges) as follows. Let $d_m =$

$\max\{d(u,v) \mid \{u,v\} \in E\}$ denote the diameter of the metric space [2]. Define $E_\ell = \{\{u,v\} \mid d(u,v) \leq d_m/n^3\}$ and $E_\hbar = E \setminus E_\ell$. We deal with these two subsets E_ℓ and $E_\hbar$ separately.

First consider the set of light edges E_ℓ and note that $G[E_\ell]$ may have several components. We would like to select an edge set $\hat{E}_\ell$ such that (i) any pair of vertices that are in the same connected component in $G[E_\ell]$ are also in the same connected component in $G[\hat{E}_\ell]$, and (ii) $wt(\hat{E}_\ell) = O(wt(MST(G)))$. (Note that one can define $\hat{E}_\ell = E_\ell$ to have these two properties but we want to "sparsify" E_ℓ, ideally we would like to have $|\hat{E}_\ell| = O(n)$ and we show this for metric with constant doubling dimension.) The algorithm for selecting $\hat{E}_\ell$ is as follows. Let S be an MIS of the distance-threshold graph G_r, where $r = d_m/n^2$. (This MIS computation is not on graph induced by E_ℓ, notice the r. This is done to obtain certain properties of $\hat{E}_\ell$ described above.) Define $\hat{E}_\ell = \{\{u,v\} \mid u \in S \text{ and } d(u,v) \leq 2 \cdot d_m/n^2\}$. Note that $\hat{E}_\ell$ may not be a subset of E_ℓ.

Now we consider the set $E_\hbar$ of heavy edges. Let $c_1 > 1$ be a constant. Let h be the smallest positive integer such that $c_1^h \geq n^3$. Observe that $h = \left\lceil \frac{3 \log n}{\log c_1} \right\rceil$. Let $r_0 = d_m/c_1^h$ (note that for any heavy edge $\{u,v\}$, $d(u,v) > r_0$) and let $r_i = c_1 \cdot r_{i-1}$, for $i > 0$. We construct $\hat{E}_\hbar$ in *layers* as follows. Let $V_0 = V$ and V_i for $0 < i \leq h$ is an MIS of the subgraph $G[E_i]$ where $E_i = \{\{u,v\} \mid d(u,v) \leq r_i\}$. Let $c_2 > c_1 + 2$ be a constant. Define $\hat{E}_i$, the edge set at the layer i as: $\hat{E}_i = \{\{u,v\} \mid u,v \in V_i \text{ and } d(u,v) \leq c_2 \cdot r_i\}$.

---

[2] If the size of the encoding of distances is more than $O(\log n)$ bits then it is suffices to know only most-significant $\log n$-bits of encoding of d_m to act as "proxy" for d_m which will only increase the approximation factor by a constant.

We define $\hat{E}_{\hbar} = \cup_{i=1}^{h} \hat{E}_i$ and $\hat{E} = \hat{E}_{\hbar} \cup \hat{E}_{\ell}$. A key feature of our algorithm is that a layer $\hat{E}_i$ does not depend on other layers and therefore these layers can be constructed in parallel. We then call an as-yet-unspecified algorithm called MST-Sparse that quickly computes an exact MST of $\hat{G} = G[\hat{E}]$ in the Congested Clique model.

In the rest of the section, we first prove that the graph $\hat{G} = (V, \hat{E})$ contains a spanning tree whose weight is within a constant-factor of the weight of a minimum spanning tree of G. This result is true for an arbitrary metric space (V, d). We then suppose that (V, d) has constant doubling dimension and show that for such metric spaces $|\hat{E}| = O(|V|)$ and therefore MST-Sparse has a simple $O(1)$ implementation in the Congested Clique model. We finally combine this result with the MIS algorithm from the previous chapter to obtain an $O(1)$-approximation to MST in a metric space with constant doubling dimension in $O(1)$ rounds in the Congested Clique model.

In the analysis that follows, we separately analyze the processing of light edges and heavy edges. We first show the *constant-approximation property* of $\hat{G}$ which doesn't require metric to be of constant doubling dimension. Later we show if the underlying metric has constant doubling dimension then Algorithm 3.5 runs in constant rounds w.h.p..

3.5.2 Constant-Approximation Property

Let $\mathcal{T}$ be an MST of graph $G = (V, E)$. Let $\hat{\mathcal{T}}$ be a MST of the graph $\hat{G} = (V, \hat{E})$. We now prove that $wt(\hat{\mathcal{T}}) = O(wt(\mathcal{T}))$. First we claim that the connectivity that edges in E_{ℓ} (i.e., the light edges) provide is preserved by the edges selected

into $\hat{E}_\ell$ (Lemma 3.9) and the total weight of these selected edges is not too high (Lemma 3.10). Later we prove a similar claim for heavy edges (Lemma 3.11).

Lemma 3.9. *For any vertices s and t in V, if there is a s-t path in $G[E_\ell]$ then there exists an s-t path in $G[\hat{E}_\ell]$.*

Proof. Consider an edge $\{u, v\} \in E_\ell$. If $\{u, v\} \in \hat{E}_\ell$ then we are done. If $\{u, v\} \notin \hat{E}_\ell$ then we show that there exists a vertex w such that $\{u, w\}, \{v, w\} \in \hat{E}_\ell$. Since $\{u, v\} \in E_\ell$, $d(u, v) \leq d_m/n^3$. Furthermore, since $\{u, v\} \notin \hat{E}_\ell$ it means both u and v are not in S, an MIS of G_r, $r = d_m/n^2$. Hence there is a vertex $w \in S$ such that $d(u, w) \leq d_m/n^2$. By the definition of $\hat{E}_\ell$, $\{u, w\} \in \hat{E}_\ell$. By the triangle inequality, we have $d(v, w) \leq d_m/n^2 + d_m/n^3$ which implies $\{v, w\} \in \hat{E}_\ell$. The lemma follows by repeatedly applying above result to each edge of the given s-t path. □

Lemma 3.10. $wt(\hat{E}_\ell) = O(wt(\mathcal{T}))$.

Proof. The weight of each edge in $\hat{E}$ is at most $2d_m/n^2$ and since there are at most n^2 edges in $\hat{E}_\ell$ (trivially), we see that $wt(\hat{E}_\ell) = O(d_m)$. We obtain the lemma by using the fact that the total weight of any spanning tree is bounded below by d_m. □

Consider an edge $\{u, v\} \in E(\mathcal{T})$. Let $C(u)$ and $C(v)$ be the components containing u and v respectively in the graph $\mathcal{T} \setminus \{u, v\}$.

Lemma 3.11. *If $\{u, v\} \in E(\mathcal{T}) \cap E_h$ then there exists an edge $\{u', v'\} \in \hat{E}$ such that (i) $d(u', v') \leq c_2 \cdot d(u, v)$ and (ii) $u' \in C(u)$ and $v' \in C(v)$.*

Proof. Let i be the largest integer such that $r_i < d(u, v)$. Hence $d(u, v) \leq r_{i+1} = c_1 \cdot r_i \leq (c_2 - 2) \cdot r_i$ (since c_2 was chosen to be greater than $c_1 + 2$).

Let u' and v' be the nearest nodes in the MIS V_i of $G[E_i]$ from u and v respectively. Note that u' could be u and v' could be v. Thus $d(u,u') \leq r_i$ and $d(v,v') \leq r_i$. By the triangle inequality we have, $d(u',v') \leq d(u',u) + d(u,v) + d(v,v') \leq r_i + (c_2 - 2) \cdot r_i + r_i \leq c_2 \cdot r_i < c_2 \cdot d(u,v)$. Hence, $(u',v') \in \hat{E}_i$ and also note that $d(u',v') \leq \alpha \cdot d(u,v)$ where α is any constant greater than c_2. Now note that $\{u,v\}$ is the lightest edge between a vertex in $C(u)$ and a vertex in $C(v)$ by virtue of being an MST edge. Therefore, it is the case that $u' \in C(u)$ and $v' \in C(v)$ since $d(u,u') < d(u,v)$ and $d(v,v') < d(u,v)$. □

This lemma implies that for every cut (X,Y) of G and an MST edge $\{u,v\}$ that crosses the cut, there is an edge $\{u',v'\}$ in $\hat{G}$ also crossing cut (X,Y) with weight within a constant factor of the weight of $\{u,v\}$. The following result follows from this observation and properties of $\hat{E}_\ell$ proved earlier.

Theorem 3.12. *Algorithm 3.5 computes a spanning tree $\hat{\mathcal{T}}$ of G such that $wt(\hat{\mathcal{T}}) = O\left(wt\left(MST\left(G\right)\right)\right)$.*

3.5.3 Constant Running Time

The result of the previous subsection does not require that the underlying metric space (V,d) have constant doubling dimension. Now we assume that (V,d) has constant doubling dimension and in this setting we show that Algorithm Metric-MST-Approximation can be implemented in *constant* rounds. Even though the algorithm is described in a "sequential" style in Algorithm 3.5, it is easy to verify that most of the steps can be easily implemented in constant rounds in the

Congested Clique model. However, to finish the analysis we need to show: (i) that ComputeMIS executes in constant rounds, (ii) that the $h = O(\log n)$ calls to ComputeMIS in Line 10 can be executed in parallel in constant rounds, and (iii) that MST-Sparse in Line 13 can be implemented in constant rounds. In the following, we show (iii) by simply showing that $\hat{G}$ has linear number of edges. In the previous section, we have shown (i) and later in this section we show (ii).

We first show $|\hat{E}_\ell| = O(n)$ in Lemma 3.13 and then argue about heavy edges.

Lemma 3.13. $|\hat{E}_\ell| = O(n)$.

Proof. For any edge $\{u, v\} \in \hat{E}_\ell$ either u or v or both belong to S (by construction). We orient edges such that an edge is directed towards the node in S. If both end points are in S then we add two oppositely directed edges. We prove that the out-degree of a node is bounded by a constant.

Consider a node u. Let $N_o(u)$ be the set of endpoints of all outgoing edges of u. If $|N_o(u)| < 2$ then we are done, therefore consider the case $|N_o(u)| \geq 2$. Consider any two nodes $v_i, v_j \in N_o(u)$. By construction we have, $d(u, v_i) \leq 2 \cdot d_m/n^2$ and $d(u, v_j) \leq 2 \cdot d_m/n^2$. Therefore by the triangle inequality, $d(v_i, v_j) \leq 4 \cdot d_m/n^2$. Also, by the definition of orientation $v_i, v_j \in S$ and therefore by the definition of S we have, $d(v_i, v_j) > d_m/n^2$. Hence the aspect ratio of $N_o(u)$ is at most 4. By the growth-bounded property, we have $|N_o(u)| = O(1)$. Hence, $|\hat{E}_\ell| = O(n)$. □

Now we show $|\hat{E}_h| = O(n)$. We first show in the following lemma two useful properties of vertex-neighborhoods in the graph induced by $\hat{E}_i$.

Lemma 3.14. *For each $u \in V_i$, (i) $|N_i(u)| \leq c_3$ where $c_3 = {c_2}^{O(\rho)}$ and (ii) $N_i(u) \cup \{u\}$ induces a clique in $G[E_j]$ for all $i > 0$ and $j \geq i + \delta$ where $\delta = \left\lceil \frac{\log 2c_2}{\log c_1} \right\rceil$.*

Proof. We first show that the aspect ratio of $N_i(u)$ is bounded by $2c_2$. This follows from two facts: (a) any two points in $N_i(u)$ are at least distance r_i apart, and (b) any point in $N_i(u)$ is at distance at most $c_2 \cdot r_i$ from u and therefore, by using the triangle inequality, any two points in $N_i(u)$ are at most $2c_2 \cdot r_i$ apart. Then using the bound from the growth-bounded property we obtain the result claimed in part (i).

Now we show part (ii) of the claim. If $|N_i(u)| = 0$ then we are done. If $|N_i(u)| = 1$ then let $v \in N_i(u)$. This implies $d(u, v) \leq c_2 \cdot r_i \ < c_1^{\delta} \cdot r_i = r_{i+\delta}$ which implies $\{u, v\} \in E_j, j \geq i + \delta$.

Now assume $|N_i(u)| > 1$. Consider any two distinct vertices $v, w \in N_i(u)$. Since $\{u, v\}, \{u, w\} \in \hat{E}_i$ we have $d(u, v) \leq c_2 \cdot r_i$ and $d(u, w) \leq c_2 \cdot r_i$. By the triangle inequality, $d(v, w) \leq 2c_2 \cdot r_i \leq c_1^{\delta} \cdot r_i = c_{i+\delta}$. Therefore $\{v, w\} \in E_{i+\delta}$ and hence we have $\{v, w\} \in E_j$, for all $j \geq i + \delta$. □

The implication of the above result is that $|\hat{E}_i|$ is linear in size. Since we use $O(\log n)$ layers in the algorithm, it immediately follows that $|\hat{E}_h|$ is $O(n \log n)$. However, part (ii) of the above result implies that only one of the nodes in $N_i(u)$ will be present in V_j, $j \geq i + \delta$ since V_j is an independent set of $G[E_j]$. This helps us show the sharper bound of $|\hat{E}_h| = O(n)$ in the following.

Without loss of generality assume that h is a multiple of δ (if not, add at most

$\delta - 1$ empty layers $\hat{E}_{h+1}, \hat{E}_{h+2}, \ldots$ to ensure that this is the case). Let

$$\beta(j) = \bigcup_{i=(j-1)\delta+1}^{j\delta} \hat{E}_i \qquad \text{for } j = 1, 2, \ldots, \frac{h}{\delta}$$

be a partition of the layers $\hat{E}_i$ into *bands* of δ consecutive layers. Let $\hat{E}_{odd} = \cup_{j:odd}\beta(j)$ and $\hat{E}_{even} = \cup_{j:even}\beta(j)$.

Lemma 3.15. *$|\hat{E}_{odd}| = O(n)$, $|\hat{E}_{even}| = O(n)$ and therefore $|\hat{E}| = O(n)$.*

Proof. We prove the claim for $\hat{E}_{odd}$. The proof is essentially the same for $\hat{E}_{even}$. We aim to prove the following claim by induction on k (for odd k): for some constant $C > 0$,

$$\left| \bigcup_{j:odd \geq k} \beta(j) \right| \leq C \cdot \left| \bigcup_{j:odd \geq k} V(j) \right|, \tag{3.1}$$

where $V(j)$ is the set of vertices such that every vertex in $V(j)$ has some incident edge in $\beta(j)$. Setting $k = 1$ in the above inequality, we see that $|\hat{E}_{odd}| = |\cup_{j:odd \geq k} \beta(j)| = O(n)$. To prove the base case, let k' be the largest odd integer less than or equal to h/δ. Then, $\cup_{j:odd \geq k'}\beta(j) = \beta(k')$ and $\cup_{j:odd \geq k'}V(j) = V(k')$. Consider a vertex $v \in V(k')$. By Lemma 3.14, there are at most c_3 edges incident on v from any layer. There are δ layers in $\beta(k')$ and therefore there are at most $c_3\delta$ edges from $\beta(k')$ incident on any vertex $v \in V(k')$. Hence, $|\beta(k')| \leq c_3\delta|V(k')|$. Therefore, for any constant $C \geq c_3\delta$, it is the case that $|\cup_{j \geq k'} \beta(j)| \leq C \cdot |\cup_{j \geq k'} V(j)|$.

Taking (3.1) to be the inductive hypothesis, let us now consider $|\cup_{j \geq k-2} \beta(j)|$. Then,

$$\left| \bigcup_{j:odd \geq k-2} \beta(j) \right| \leq \left| \bigcup_{j:odd \geq k} \beta(j) \right| + |\beta(k-2)| \leq C \cdot \left| \bigcup_{j:odd \geq k} V(j) \right| + c_3\delta \cdot |V(k-2)|. \tag{3.2}$$

The second inequality is obtained by applying the inductive hypothesis and the inequality $|\beta(k-2)| \leq c_3\delta|V(k-2)|$. By Lemma 3.14, at most half the vertices in $V(k-2)$ appear in $\cup_{j\geq k}V(k)$. Therefore, $|V(k-2) \setminus (\cup_{j\geq k}V(j))| \geq |V(k-2)|/2$. Hence,

$$\left|\bigcup_{j:odd\geq k-2}\beta(j)\right| \leq C\cdot\left|\bigcup_{j:odd\geq k}V(j)\right| + 2c_3\delta\cdot\left|V(k-2)\setminus(\bigcup_{j:odd\geq k}V(j))\right|.$$

Picking $C \geq 2c_3\delta$, we then see that

$$\left|\bigcup_{j:odd\geq k-2}\beta(j)\right| \leq C\cdot\left(\left|\bigcup_{j:odd\geq k}V(j)\right| + \left|V(k-2)\setminus\left(\bigcup_{j:odd\geq k}V(j)\right)\right|\right) = C\cdot\left|\bigcup_{j:odd\geq k-2}V(j)\right|.$$

The result follows by induction. □

3.5.4 Many MIS Computations in Parallel

In this section, we argue that Algorithm 2.6 LowDimensionalMIS can be executed on the $O(\log n)$ different distance threshold graphs in parallel on a Congested Clique. Table 3.1 shows number of messages sent/received per node in the execution of Algorithm 2.6 and from this it is easy to see that Line 8 of Phase 2 can be executed as it is using Lenzen's routing protocol in $O(1)$ rounds for all the $O(\log n)$ layers in parallel due to their low communication requirements. For Lines 4-6 of Phase 2 we do the following load balancing via a *designated receiver scheme*: each vertex has to send at most $O(n^{1/4}\log n)$ messages in an execution of Phase 2 for a layer. Therefore, for $O(\log n)$ layers one node is responsible of sending $O(n^{1/4}\log^2 n)$ messages. There are only $\lceil 2\log n\rceil$ receivers needed for in an execution at a layer. For all layers the number of receivers needed are $O(\log^2 n)$. Hence we can designate different receivers

such that no receiver gets more than $O(n)$ messages in execution of Phase 2 for all layers. Similar designated receiver scheme is applied for the execution of Phase 1.

For parallel execution of Line 9 (SEQUENTIALMIS) of Phase 4 for all $O(\log n)$ layers we use the following *message encoding scheme*: Each vertex v constructs a $O(\log n)$-length bit string specifying 1 at position ℓ if v is in MIS for the layer ℓ otherwise 0. Each vertex v broadcasts this string. For a layer ℓ, each vertex considers only ℓ^{th} bit of this message.

Table 3.1: Number of messages sent/received per node in the execution of Algorithm 2.6

Phase	Line	Analysis	Number of messages to send per node	Number of receivers	Number of messages to receive per receiver
1	2-4	Lemma 2.9	$O(n^{1/2})$	$n^{1/2}$	$O(n)$
2	4-6	Lemma 2.12	$O(n^{1/4}\log n)$	$\lceil 2\log n \rceil$	$O(n)$
	8	Lemma 2.13	$O\left(\text{poly}(\log n)\right)$	n	$O\left(\text{poly}(\log n)\right)$
3	-	Thm. 2.8	$O(n^{1/2}\,\text{poly}(\log^* n))$	n	$O(n^{1/2}\,\text{poly}(\log^* n))$
4	3	Lemma 2.15	$O(1)$	1	$O(n)$
	9	Lemma 2.15	1 (1-bit)	n	n

3.6 Conclusion

Our work makes progress in understanding both the time and message complexity of two important graph problems, graph connectivity and minimum spanning tree, in the Congested Clique. We improve the upper bound on the round complexity of MST in the Congested Clique model significantly, presenting an $O(\log \log \log n)$-round algorithm, and this makes the question of whether there is an $O(1)$-round MST algorithm in the Congested Clique model even more tantalizing. Our work also suggests new questions, that simultaneously focus on both round and message complexity. For example, is it possible to design sub-logarithmic GC or MST algorithms that use $O(n \operatorname{polylog} n)$ messages?

Algorithm 3.4 SQ-MST

Input: A weighted subgraph $G'(V', E', wt)$ with $O\left(\frac{n}{\log^4 n}\right)$ vertices and $O(n^{3/2})$ edges

Output: An MST of G'

1. $r(E) \leftarrow$ DistributedSort(E); each edge $e \in E'$ is assigned a rank $r(e)$, in non-decreasing order of edge-weights.
2. Partition edges in E based on their ranks $r(e)$ into p partitions $E_1, E_2, \ldots E_p$ ($p = O(\sqrt{n})$), each partition having n edges (E_p might have less than n edges) such that E_1 contains edges with ranks $1, 2, \ldots, n$; E_2 contains edges with ranks $n+1, n+2, \ldots, 2n$; and so on.
3. Let $g(i)$ be the node in G with ID i. Assign $g(i)$ as the "guardian" of part E_i. Nodes send edges in E_i to $g(i)$.
4. **for** $i = 1$ **to** $i = p$ **in parallel do**
5. Let $G_i = (V', \cup_{j=1}^{i-1} E_j)$. Each vertex $v \in V'$ constructs $t = \Theta(\log n)$ sketches $\mathbf{s}_v^{i,1}, \mathbf{s}_v^{i,2}, \ldots, \mathbf{s}_v^{i,t}$. of its neighborhood with respect to G_i.
6. Each node $v \in V'$ sends the sketch collection $\{\mathbf{s}_v^{i,j}\}_{j=1}^{t}$ to $g(i)$.
7. $g(i)$ executes locally:

   (a) $\mathcal{T}_i \leftarrow$ SpanningForest(G_i) (based on linear sketches received)

   (b) $g(i)$ processes edges in E_i in rank-based order.

   for each edge $e_j = \{u, v\}$ in $e_1, e_2, \ldots$:

   if there is a path between u and v in $\mathcal{T}_i \cup \{e_\ell \mid \ell < j\}$ then discard e_j

   else add e_j to $\mathcal{M}_i$.
8. **end for**
9. **return** $\cup_{i=1}^{p} \mathcal{M}_i$

Algorithm 3.5 METRIC-MST-APPROXIMATION

Input: A metric graph $G = (V, E)$ on metric space (V, d)

Output: A tree $\hat{\mathcal{T}}$ such that $wt(\hat{\mathcal{T}}) = O\left(wt\left(MST\left(G\right)\right)\right)$

1. $d_m = \max\{d(u,v) \mid \{u,v\} \in E\}$
2. $E_\ell \leftarrow \left\{\{u,v\} \mid d(u,v) \leq \frac{d_m}{n^3}\right\}$ ▷ Processing light edges
3. $S \leftarrow$ COMPUTEMIS$(G[E_0])$ where $E_0 \leftarrow \left\{\{u,v\} \mid d(u,v) \leq \frac{d_m}{n^2}\right\}$
4. $\hat{E}_\ell \leftarrow \left\{\{u,v\} \mid u \in S \text{ and } d(u,v) \leq \frac{2 \cdot d_m}{n^2}\right\}$
5. $E_h \leftarrow \left\{\{u,v\} \mid d(u,v) > \frac{d_m}{n^3}\right\}$ ▷ Processing heavy edges
6. $h \leftarrow \left\lceil \frac{3 \log n}{\log c_1} \right\rceil$; $r_0 \leftarrow \frac{d_m}{c_1^h}$
7. **for** $i = 1$ **to** h **in parallel do**
8. $\quad r_i \leftarrow (c_1)^i \cdot r_0$
9. $\quad E_i \leftarrow \{\{u,v\} \mid d(u,v) \leq r_i\}$
10. $\quad V_i \leftarrow$COMPUTEMIS$(G[E_i])$
11. $\quad \hat{E}_i \leftarrow \{\{u,v\} \mid u,v \in V_i \text{ and } d(u,v) \leq c_2 \cdot r_i\}$
12. **end for**
13. $\hat{E}_h \leftarrow \cup_{i=1}^{h} \hat{E}_i$; $\hat{E} \leftarrow \hat{E}_\ell \cup \hat{E}_h$
14. **return** MST-SPARSE$(G[\hat{E}])$

CHAPTER 4
REDUCING MESSAGE COMPLEXITY OF GRAPH CONNECTIVITY AND MINIMUM SPANNING TREE

4.1 Introduction

In earlier chapter (Chapter 3), it was shown that there are randomized (Monte Carlo) algorithms that solve the *Graph Connectivity* (GC) problem and the *minimum spanning tree* (MST) problem, both in $O(\log\log\log n)$ rounds. This result improved on the previous fastest (deterministic) MST algorithm of Lotker et al. [39], that ran in $O(\log\log n)$ rounds. While the above mentioned algorithms are exceedingly fast, their speed seems to arise from the use of the entire bandwidth of the Congested Clique, i.e., essentially all $\Theta(n^2)$ possible messages. Both algorithms consist of steps in which $\Theta(n^2)$ messages are exchanged, in the worst case. In this chapter, we investigate the question of whether GC and MST can be solved as fast, if we impose stringent restrictions on the *message complexity*, i.e., the total number of messages that are sent during the course of the algorithm. Specifically, we consider two natural restrictions on the message complexity: near-linear in n, i.e., $O(n \operatorname{poly}\log n)$ and near-linear in m, i.e., $O(m \operatorname{poly}\log n)$, where n and m are the number of vertices and edges respectively of the input graph. This focus on the message complexity of Congested Clique algorithms (in addition to round complexity) is motivated by connections between the Congested Clique model and models of large-scale distributed computing such as MapReduce [22] and the "big data" model [31]. These papers [22, 31] prove simulation theorems that respectively show how an algorithm in the Congested Clique model can

be simulated in the MapReduce model and in the "big data" model. The motivation for these theorems is simple: due to the simplicity of the Congested Clique model, it is far easier to design algorithms in this model as compared to the MapReduce or the "big data" model. As might be expected, the above mentioned simulation theorems yield efficient algorithms in the underlying model, if the initial Congested Clique algorithm is not only fast (has small round complexity), *but also has small message complexity*. More generally, the Congested Clique model can be viewed an abstraction of overlay networks (as mentioned in [39]) or as an abstraction of high-performance computing clusters, similar in spirit to the BSP model [56]. For Congested Clique algorithms to be meaningful in these contexts, it is critical that they not only be fast, but have low message complexity as well.

4.1.1 Notation

We denote $m = |E|$, the number of edges in an input graph. All logarithms are assumed to have base 2 unless otherwise specified. Throughout, by "w.h.p." we mean with probability at least $1 - n^{-\Omega(1)}$. For ease of exposition, we assume nodes have distinct IDs in the range $[0, n-1]$. We denote a set of integers $1, 2, \ldots, x$ by $[x]$.

4.1.2 Related Work

In [21], it is shown that one can compute MST on a Congested Clique using $O(n \operatorname{poly}\log n)$ messages but it also requires $O(\operatorname{poly}\log n)$ rounds. This result is a direct application of linear sketches [1, 2, 42]. Recently, King et al. [30] showed how to construct an MST in a distributed network with $o(m)$ communication. Their

algorithm can be adopted to Congested Clique model to run in $O(\text{poly}\log n)$ rounds using $O(n\,\text{poly}\log n)$ messages. King et al. [30] obtained this result via developing a linear sketch type short representation of neighborhood of a vertex. Both [21] and [30] results rely on a representation which allows sampling an outgoing edge from a component quickly.

4.1.3 Main Results

We first focus on GC and show that it can be solved in $O(\log\log\log n)$ rounds using only $O(n\,\text{poly}\,\log n)$ messages.

4.1.3.1 Low-message-complexity GC Algorithm

The GC algorithm in [21] that runs in $O(\log\log\log n)$ rounds, uses the fast parallel merge procedure of Lotker et al. [39] as a pre-processing step to reduce the size (number of vertices) of the input graph to $O(n/\,\text{poly}\log n)$. This merge procedure uses $\Theta(n^2)$ messages per iteration and our main technical contribution lies in showing how to reduce the message complexity of this merge procedure to $O(n\,\text{poly}\log n)$ messages. Specifically, we show how to use linear sketches [1, 2, 42] to sample sufficiently many edges connecting different components in parallel so as to be useful to the fast parallel merge procedure of Lotker et al. [39]. An additional technique that we use to limit the message complexity of this algorithm to $O(n\,\text{poly}\log n)$ is the design of a simple routing primitive that has linear message complexity, while requiring only a constant number of rounds for completion. Many recent Congested Clique algorithms have relied on the deterministic routing protocol due to Lenzen

[37] that runs in constant rounds on the Congested Clique. While this subroutine is extremely useful for designing fast Congested Clique algorithms, the number of messages is not a resource it tries to explicitly conserve. Specifically, Lenzen's routing protocol uses $\Omega(n^{1.5})$ messages, independent of the number of messages that need to be routed. We observe that our GC algorithm does not require the full power of Lenzen's routing protocol and our routing primitive suffices for all the routing needs of our GC algorithm.

4.1.3.2 Linear-message-complexity MST Algorithm

We then consider the more challenging MST problem and first show that MST can be solved in $O(\log\log\log n)$ rounds using $O(m \operatorname{poly}\log n)$ messages. In addition to the low-message-complexity routing primitive mentioned above, this result depends on a low-message-complexity sorting primitive (based on the Congested Clique sorting algorithm of [37]) that we present. While we do not know if *exact* MST can be solved in $O(\log\log\log n)$ rounds using $O(n \operatorname{poly}\log n)$ messages, we do show that for any $\varepsilon > 0$, a $(1+\varepsilon)$-approximation of MST can also be constructed in $O(\log\log\log n)$ rounds using only $O(n \operatorname{poly}\log n)$ messages. This final approximation algorithm result makes crucial use of the GC result and the exact-MST result mentioned above.

4.2 Technical Preliminaries

In this section we describe the low-message subroutines for the routing and sorting problem that we use in our algorithms.

4.2.1 Routing

Theorem 4.1 (Randomized Scatter-Gather (RSG))**.** *There are k messages that need to be delivered and each node is source of up to n messages and each node is destination of up to $c \cdot n^{1-\varepsilon}$ messages, where $\varepsilon > 0$ and $c \geq 1$ are constants. Then there exists an algorithm that, with probability at least $1 - \frac{1}{n}$, delivers all k messages within $\left\lceil \frac{3c}{\varepsilon} \right\rceil$ rounds using $2k$ messages.*

Proof. Each node v distributes messages it needs to send, uniformly at random among all nodes, with the constraint that no node gets more than one message. Each intermediate node then sends the received messages to the specified destinations. If an intermediate node receives several messages intended for the same destination, it sends these one-by-one in separate rounds. We show that w.h.p. no intermediate node will receive more than $\left\lceil \frac{3c}{\varepsilon} \right\rceil$ messages intended for the same destination and hence every intermediate node can deliver all messages to destinations in $\left\lceil \frac{3c}{\varepsilon} \right\rceil$ rounds.

Let M_w be the set of messages from all senders intended for w and let $r_w = |M_w| \leq c \cdot n^{1-\varepsilon}$ be the total number of messages intended for w. Consider a node u. Let $X_w(u)$ be the random variable denoting the number of messages intended for w, received by u in the first step. For $m \in M_w$, let $Y_m(u) \in \{0, 1\}$ indicate if m was sent to u in the first step. Hence $X_w(u) = \sum_{m \in M_w} Y_m(u)$. Since u was chosen uniformly at random as the intermediate destination for messages intended to w, we have $\mathbf{E}[X_w(u)] \leq \frac{cn^{1-\varepsilon}}{n} = c \cdot n^{-\varepsilon}$. Notice that if for any subset of messages in M_w if the sources of these messages is different then the corresponding indicator variables are independent. On the other hand if the source of these messages is the same

then they are negatively correlated [14]. Therefore by Chernoff's bound [14] we have, $\Pr(X_w(u) > c') \leq n^{-2}$ where $c' \leq \left\lceil \frac{3c}{\varepsilon} \right\rceil$. By the union bound, with probability at least $1 - n^{-1}$, each intermediate node will receive at most $\left\lceil \frac{3c}{\varepsilon} \right\rceil$ messages intended for each node and hence can be delivered in less than $\lceil \frac{3c}{\varepsilon} \rceil$ rounds. □

By using techniques from [6, 11], we obtain the following result for a particular case of the routing problem.

Theorem 4.2 (Deterministic Scatter-Gather (DSG)). *A subset of nodes hold a bulk of messages intended to a node v^* such that the total number of messages is $k \leq cn$. Then there exists a deterministic algorithm that delivers all k messages within $2\left\lceil \frac{k}{n} \right\rceil + 2$ rounds using $2k + 2$ messages. Moreover, this can be extended to a scenario where there is a set $V^* \subseteq V$ of destinations and every message needs to be delivered to every node in V^*. In this case, the algorithm terminates in $2\left\lceil \frac{k}{n} \right\rceil + 2$ rounds using $(2k + 2)|V^*|$ messages.*

Proof. Let $V' \subseteq V$ be the set of nodes who holds messages for v^*. Let a_i be the number of messages at node v_i. We first order all messages in an arbitrary order as follows. Each node v_i orders its a_i messages arbitrary $m_{i_1}, \ldots, m_{i_{a_i}}$ which induces a global order, obtained by local node orders based on IDs. Nodes can learn the global index of a message in 2 rounds of communication: node v_i sends a_i to v^* and then v^* sends the starting index to v_i. Each node sends the message with index j to an intermediate destination node $j \bmod n$ which takes at most $\left\lceil \frac{k}{n} \right\rceil$ rounds. Hence each intermediate node receives at most $\left\lceil \frac{k}{n} \right\rceil$ messages which gets delivered to v^* in

additional $\lceil \frac{k}{n} \rceil$ rounds. To extend this to multiple v^*, the intermediate node sends the copy of message to each such $v^* \in V^*$. Therefore the above algorithm delivers all k messages as claimed. □

Now consider the reverse scenario:

Theorem 4.3 (Deterministic Gather-Scatter)**.** *A node v^* holds a bulk of messages intended for a subset of nodes $R \subseteq V$ such that the total number of messages is $k \leq n$ and each message needs to delivered to* all *nodes in R. Then there exists a deterministic algorithm that delivers all k messages within 2 rounds using $k + k \cdot |R|$ messages.*

Proof. Node v^* sends each message m_i to a *supporter node* s_i. Since $k < n$, an one-to-one mapping of m_i to s_i is possible and hence this can be done in a single round and uses k messages. Each supporter node then broadcast the received message to all nodes in R. This requires one round and $k \cdot |R|$ messages. □

4.2.2 Sorting Subroutine

Consider the following problem: given k keys of size $O(\log n)$ each from a totally ordered universe such that each node has up to n keys. The goal is to learn the rank of each of these keys in a global ordered enumeration of all k keys. Patt-Shamir and Teplitsky [49] designed a randomized algorithm that solved this problem in $O(\log \log n)$ rounds which was later improved to $O(1)$ rounds by the deterministic algorithm of Lenzen [37]. But, both the algorithms [49, 37] have $\Omega(n^{1.5})$ message complexity regardless of the number of keys to sort. We provide a randomized

algorithm which reduces the problem to the similar problem as above but on a smaller clique. Our algorithm solves the problem for $k = O(n^{2-\varepsilon}), \varepsilon > 0$ in $O(1)$ rounds using $O(k)$ messages w.h.p.

The high level idea of our Algorithm DistributedSort is to redistribute k keys to $\lfloor\sqrt{k}\rfloor$ nodes and then sort them using Lenzen's sorting algorithm [37] on the clique induced by these $\lfloor\sqrt{k}\rfloor$ nodes in $O(1)$ rounds with $O(k)$ messages. For the redistribution, we rely on our low-message routing schemes (RSG and DSG). Let k_v be the number of keys v has. Each node v sends k_v to node v^*. Notice that, $k = \sum_{w \in V} k_w$. Let $idx_w = \sum_{u:ID(u)<ID(w)} k_u$ for all $w \in V$. For each $w \in V$, v^* sends idx_w to w . Order keys present at each node v arbitrarily. Assign labels to keys starting from idx_v. Set destination of the key with label i to node ($i \mod \lfloor\sqrt{k}\rfloor$). At this point the input is divided among $\lfloor\sqrt{k}\rfloor$ nodes, each holding up to $\lceil\sqrt{k}\rceil$ keys. Let V_μ denote the set of nodes with IDs in the range $[0, \lfloor\sqrt{k}\rfloor - 1]$. Nodes in V_μ executes Lenzen's sorting algorithm [37] and learn the global index of the keys in sorted order. Each key with its rank in global sorted order is sent back to the original node (by reversing the route applied earlier to this key).

Theorem 4.4 (Distributed Sorting)**.** *Given $k = O(n^{2-\varepsilon})$ comparable keys of size $O(\log n)$ each such that each node has up to n keys for some constant $\varepsilon > 0$. Then, Algorithm* DistributedSort *requires $O\left(\frac{1}{\varepsilon}\right)$ rounds and $O(k)$ messages w.h.p., such that at the end of the execution each node knows the rank of each key it has.*

Proof. We first show that the redistribution of keys among $\lfloor\sqrt{k}\rfloor$ nodes takes $O(1)$ rounds and $O(k)$ messages. Since each of the $\lfloor\sqrt{k}\rfloor$ nodes need to receive

$\lceil \sqrt{k} \rceil = O(n^{1-\varepsilon})$ keys, the keys can be routed using RSG scheme (Theorem 4.1) in $O(1)$ rounds and $O(k)$ messages. Nodes in V_μ can now execute Lenzen's sorting algorithm [37] which takes $O(1)$ rounds and $O(k)$ messages. The reverse routing of these keys takes another $O(1)$ rounds and $O(k)$ messages. Therefore, in total Algorithm DistributedSort required $O(1)$ rounds and $O(k)$ messages. □

4.3 Graph Connectivity

In this section, we present a randomized (Monte Carlo) algorithm for GC problem on a Congested Clique, running in $O(\log \log \log n)$ rounds *using* $O(n \operatorname{poly} \log n)$ *messages*, w.h.p. Our GC algorithm constructs a maximal spanning forest of the input graph (i.e., a spanning forest with as many trees as the number of components in the input graph) and at the end of the algorithm every node knows which of its incident edges are part of the forest and knows whether the graph is connected or not. We make use of *linear sketches [1, 2, 42]* and *a modified version of Lotker et al. [39] MST algorithm.*

4.3.1 Graph Sketches

A key tool used by our algorithm is *linear sketches* [1, 2, 42]. We described this concept in-depth in Chapter 3 (see Section 3.2.1 therein). Since it is a key tool we summarize it below for convenience.

Let $\mathbf{a}_v$ denote a vector whose non-zero entries represent edges incident on v. A *linear sketch* of $\mathbf{a}_v$ is a low-dimensional random vector $\mathbf{s}_v$, typically of size $O(\operatorname{poly}(\log n))$, with two properties: (i) sampling from the sketch $\mathbf{s}_v$ returns a non-

zero entry of $\mathbf{a}_v$ with uniform probability (over all non-zero entries in $\mathbf{a}_v$) and (ii) when nodes in a connected component are merged, the sketch of the new "super node" is obtained by coordination-wise addition of the sketches of the nodes in the component. The first property is referred to as ℓ_0-*sampling* in the streaming literature [9, 42, 26] and the second property is *linearity*. The graph sketches used in [1, 2, 42] rely on the ℓ_0-sampling algorithm by Jowhari et al. [26]. Sketches constructed using the Jowhari et al. [26] approach use $\Theta(\log^2 n)$ bits per sketch, but require polynomially many mutually independent random bits to be shared among all nodes in the network. Sharing this volume of information is not feasible; it takes too many rounds and too many messages. So instead, we appeal to the ℓ_0-sampling algorithm of Cormode and Firmani [9] which requires a family of $\Theta(\log n)$-wise independent hash functions, as opposed to hash functions with full-independence. In the earlier chapter (Chapter 3) we provided details of how the Cormode-Firmani approach can be used in the Congested Clique model to construct graph sketches. We summarize the result in the following theorem.

Theorem 4.5. *Given a graph $G = (V, E)$, $n = |V|$, there is a Congested Clique algorithm running in $O(1)$ round, using $O(n \operatorname{poly}\log n)$ messages, at the end of which every node $v \in V$ has computed a sketch of $\mathbf{a}_v$. The size of the computed sketch of a node is $O(\log^4 n)$ bits. The ℓ_0-sampling algorithm on this sketch returns an edge in $\mathbf{a}_v$ with probability $1/(\text{number of non-zero entries in } \mathbf{a_v}) \pm n^{-c}$. Moreover, if $s = O(\operatorname{poly}\log n)$ sketches per node need to be constructed in parallel then it requires $O(1)$ rounds and $O(s \cdot n \operatorname{poly}\log n)$ messages.*

4.3.2 Overview of The Graph Connectivity Algorithm

Similar to GC algorithm in [21], our GC algorithm runs in two stages. Initially, the input graph can be viewed as having n components — one for each vertex. In the first stage, we reduce the number of components to $O\left(\frac{n}{\log^4 n}\right)$ by running the *modified version of Lotker et al. algorithm* for $O(\log\log\log n)$ phases (subsection 4.3.3. Stage 2 operates on the resulting *component graph*. This is the graph whose vertices are the components computed in Stage 1 and whose edges represent adjacencies between components. We show that the reduction in the number of components due to Stage 1 helps to finish Stage 2 in another $O(1)$ rounds using only $O(n)$ messages (subsection 4.3.4). Both Stage 1 and Stage 2 requires communication of at most $O(n \operatorname{poly}\log n)$ messages overall.

First we describe the Lotker et al. [39] MST algorithm for an edge-weighted clique graph. The Lotker et al. algorithm runs in phases, taking constant number of communication rounds per phase. At the start of phase k, each component is of size at least $\mu = 2^{2^{k-2}}$ and we have MST of each component. At the end of phase $k \geq 0$, the algorithm has computed a partition $\mathcal{F}^k$ of the nodes of G into components such that for each $F \in \mathcal{F}^k$, $|F| \geq \mu^2$. Furthermore, for each component $F \in \mathcal{F}^k$, the algorithm has computed a minimum spanning tree $T(F)$. The sets $\mathcal{F}^k$ and $\mathcal{T}^k (= \{T(F) : F \in \mathcal{F}^k\})$ are known to *every* node at the end of the phase k. The goal of each component leader in phase k is to select edges so that at least μ outgoing edges are picked that are connecting to μ distinct components. These edges are sent to v^* (the node with minimal ID) and v^* locally inspects these edges and decides

which of these edges are added to $\mathcal{T}^k$ and updates $\mathcal{F}^k$ accordingly (*merge procedure*). v^* then distributes $\mathcal{F}^k$ and $\mathcal{T}^k$ to all nodes. It is shown that at the end of phase k the size of the smallest cluster is at least $\mu^2 = 2^{2^{k-1}}$ and hence $|\mathcal{F}^k| \leq n/2^{2^{k-1}}$. Hence after $\log\log n$ such phases, the algorithm terminates. On the other hand, communicating entire $\mathcal{T}^k$ to all nodes takes $\Theta(n^2)$ messages in each phase. This algorithm can be adopted to an m-edge graph and the message complexity can be reduced to $\Theta(m)$ per phase by communicating only the component labels to neighbors in each phase instead of communicating the entire $\mathcal{T}^k$. Hence, one can modify Lotker et al. algorithm to run in $O(\log\log n)$ rounds using $O(m\log\log n)$ messages. But in case of Graph Connectivity we want to reduce the message complexity further to $O(n\,\mathrm{poly}\log n)$. In the following subsection we show how to felicitate the merge procedure for the first $O(\log\log\log n)$ phases.

4.3.3 Stage 1: Fast Parallel Merge via Sketches

The goal of Stage 1 is to reduce the number of components to at most $O\left(\frac{n}{\log^4 n}\right)$ in $O(\log\log\log n)$ rounds using only $O(n\,\mathrm{poly}\log n)$ messages. As described above, if we run Lotker et al. algorithm for $O(\log\log\log n)$ phases then the number of components are reduced as needed but it requires $\Theta(m\log\log\log n)$ messages. To get around this critical issue we propose the following algorithm using linear sketches. Consider a phase $k \leq \lceil\log\log\log n\rceil + 3$. As said earlier, in a phase k each component needs to pick μ edges connecting to μ components in $O(1)$ rounds. If a component leader has sketches of all nodes in its component then using these

sketches it can sample one edge incident on its component. The challenges are: (i) there is no restriction on the size of a component, for example, if a component is of size $\omega(n/\log^4 n)$ then communicating sketches of nodes in it to the leader require more than constant rounds of communication. (ii) how many sketches are required to pick at least μ edges to μ *distinct* components? To tackle challenge (i), we show that we can restrict the growth of components to $O(\text{poly}\log n)$ without affecting the progress of the algorithm up to $O(\log\log\log n)$ phases. If the size of any component is restricted to $O(\text{poly}\log n)$ then each node in the component can communicate its sketch to the component leader in $O(1)$ rounds using $O(n\,\text{poly}\log n)$ messages by executing RSG scheme (Theorem 4.1). To address challenge (ii), we analyze the number of edges need to be sampled to make progress (in terms of size of components) and given the size restriction in (i) we argue that $O(\text{poly}\log n)$ sketches are sufficient.

Let v^* be the node in the graph with minimum ID. At the start of the phase k, v^* knows the set of edges $\mathcal{T}^{k-1}$ ($\mathcal{T}^0 = \emptyset$) which induces the set of components $\mathcal{F}^{k-1}$ (for each $F_i \in \mathcal{F}^0$, $F_i = \{v_i\}$). For each $F \in \mathcal{F}^{k-1}$, if $|F| \geq \left\lfloor \log^4 n \right\rfloor$ then the *status* of component F and all the nodes in F is *inactive*, otherwise the status is *active*. Moreover, each node in component $F \in \mathcal{F}^{k-1}$ knows its component label $\ell(F)$, which is the node with the minimum ID in F, and knows its status. $\ell(F)$ acts as the component leader of F. At the end of the phase, a set of edges $\mathcal{T}^k$ is known to v^* such that the size of any component induced by $\mathcal{T}^k$ is at least $2^{2^{k-1}}$.

Algorithm 4.1 describes phase $k < \lceil \log\log\log n \rceil + 3$ of Stage 1. Let μ be the size of the smallest component (known to every component leader at the

beginning of the phase). The goal is to provide enough information to each component leader so that the component leader can sample μ edges connecting to μ distinct components. This is achieved in Steps 1-2. Specifically, in Step 1 each active node computes $t = \Theta(\log^{13} n)$ sketches (Theorem 4.5). These sketches are gathered at respective component leaders by RSG scheme (Theorem 4.1) in $O(1)$ rounds using only $O(n \operatorname{poly} \log n)$ messages. In Step 2, each active component leader samples t edges, one from each sketch. We show that from these t edges, a leader of an active component F can pick μ edges connecting to μ distinct components or an inactive component (Lemma 4.8). These μ edges get delivered to v^* in Step 2(b) and 3. In Step 4, v^* locally executes the merge procedure (Lotker et al. [39]) on the edges received from Step 3 and the set $\mathcal{T}^{k-1}$. The merge procedure takes received edges and $\mathcal{T}^{k-1}$ and computes a spanning forest $\mathcal{T}^k$. v^* then sends a message to each node notifying it of its (possibly) new component label with respect to the set of components F^k induced by edges in $\mathcal{T}^k$. If the size of the component F is more than $\left\lfloor \log^4 n \right\rfloor$ then $v*$ also notifies $\ell(F)$ to become inactive in subsequent phases.

Lemma 4.6. *Step 1 of Algorithm 4.1 can be implemented in $O(1)$ rounds and using $O(n \log^{16} n)$ messages w.h.p.*

Proof. A node needs to send $O(\log^{13} n)$ sketches. Each sketch is of size $O(\log^4 n)$ bits (Theorem 4.5). Hence a node needs to send $O(\log^{17} n)$ bits in total. On the other hand, the leader of the active component F is a receiver of at most $O(|F| \cdot \log^{17} n)$ bits. F is active only if $|F| < \log^4 n$. Therefore, in any phase $k \leq \lceil \log \log \log n \rceil + 3$, a leader of active component needs to receive at most $O(\log^{21} n)$ bits. These messages

can be routed using RSG scheme (Theorem 4.1) which requires $O(1)$ rounds and $O(n \log^{16} n)$ messages w.h.p. □

Lemma 4.7. *Let S be a subset of edges incident on an active component $F \in \mathcal{F}^{k-1}$ such that $|S| \geq \log^{12} n$. Let $\mu = \min\{|F'| : F' \in \mathcal{F}^{k-1}\}$. Then S contains either an edge connecting F to an inactive component or μ edges connecting F to μ different active components in $\mathcal{F}^{k-1}$.*

Proof. Let f be the size of the largest active component in $\mathcal{F}^{k-1}$ at the beginning of phase k. Consider an active component F. If S has an edge which connects F to an inactive component then we are done. Therefore, consider the case where S doesn't contain an edge which connects F to an inactive component. The number of edges having exactly one end point in F and the other in F' is at most f^2 for all active neighbors F' of F. Therefore, if we have f^3 distinct edges incident on F, then by pigeonhole principle, at least f of these edges connect F to f distinct active components. By the definition of active components, we have $f < \left\lfloor \log^4 n \right\rfloor$. Therefore, any subset of incident edges on F of size at least $f^3 < \left\lceil \log^{12} n \right\rceil$ has the claimed property. □

Lemma 4.8. *In Step 2, w.h.p., a leader node of an active component $F \in \mathcal{F}^{k-1}$ possesses sufficient information to sample μ outgoing edges such that these sampled edges connect to μ different components or contain an edge connecting to an inactive component where $\mu = \min\{|F'| : F' \in \mathcal{F}^{k-1}\}$.*

Proof. Since $\mu \leq |F|$, by Lemma 4.7, $\left\lceil \log^{12} n \right\rceil$ distinct incident edges on F are

sufficient to obtain μ sampled edges satisfying the above claimed property. We show that the leader of F possesses sufficient information to compute a set of incident edges on F with size at least $s = \left\lceil \log^{12} n \right\rceil$.

If we sample $O(s \log s)$ edges incident on F uniformly and independently at random, then the sample set has s distinct samples w.h.p. by coupon collector argument as follows: Let T denote the number of sampling steps required to obtain s distinct edges. Let t_i $(1 \leq i \leq s)$ denote the number of sampling steps to collect the i^{th} distinct edge after $i-1$ distinct edges have been obtained. $T = \sum_{i=1}^{s} t_i$. The probability of sampling a new edge given iâĹŠ1 edges is $p_i = (s-(i-1))/s$. Therefore, t_i has geometric distribution with expectation $1/p_i$. By the linearity of expectations we have, $\mathbf{E}[T] = \sum_{i=1}^{s} \frac{1}{p_i} = s \cdot H_s$ (where H_s is the s^{th} harmonic number. It is easy to show that $\Pr(T > 3 \log s) < n^{-2}$. Therefore, $\Theta(s \log s)$ samples suffices to obtain a set of size s w.h.p.

Now, to execute a j^{th}, $j \in [1, \Theta(s \log s)]$ sampling step, we sample an edge from j^{th} sketch which returns an incident edge on F with near-uniform probability. That is, we get an incident edge with probability $\frac{1}{s} \pm n^{-2}$ (Theorem 4.5) instead of $\frac{1}{s}$. Notice that, if we replace the uniform probability with the above probability in the earlier analysis then the number of samples required is $\Theta(s \log s) + O(n^{-2})$. Hence $O(s \log s) + o(1) = O(\log^{13} n)$ sketches are sufficient. □

To aid the analysis of Algorithm 4.1 we define the following terms.

Definition. *Fake and real edges All the edges in the component graph have weight* 1 *and they are called* real *edges. For an active component* $F \in \mathcal{F}^{k-1}$ *if degree of* F

in the component graph is less than μ then then the algorithm considers fake *edges connecting F to distinct components with weight ∞.*

Theorem 4.9. *At the end of phase $k \leq \lceil \log\log\log n \rceil + 3$ in Algorithm 4.1, the size of the smallest component in $\mathcal{F}^k$ is at least $2^{2^{k-1}}$ w.h.p.*

Proof. If $F \in \mathcal{F}^{k-1}$ is inactive then $|F| \geq \log^4 n \geq 2^{2^{k-1}}$ for any $k \leq \lceil \log\log\log n \rceil + 3$. Lotker et al. show that if for each component $F \in \mathcal{F}^{k-1}$, δ(= size of the smallest component in $\mathcal{F}^{k-1}$) edges connecting F to δ distinct components are received from each component then the size of the smallest component induced by selected edges and $\mathcal{T}^{k-1}$ is at least δ^2. Therefore, in our case, if v^* receives μ edges connecting F to μ distinct components for every active component $F \in \mathcal{F}^{k-1}$ then we are done. So consider the following cases.

If degree of F in the component graph is less than μ then v^* assumes that the remaining edges are fake edges. v^* constructs a minimum spanning forest $\mathcal{T}^k$ of the edges in $\mathcal{T}^{k-1}$, edges received, and the fake edges. The claim still holds by applying the similar arguments as in Lotker et al. [39]. Because of the fake edges some of the components may not represent the connected components of the input graph, we deal with them separately in Stage 2.

On the other hand, if v^* received μ edges from an active component F but these do not connect F to distinct components. Then, by Lemmas 4.7 and 4.8, it must be the case that at least one of μ edges connect F to an inactive component w.h.p. In this case, the size of this newly formed component is at least $\log^4 n \geq 2^{2^{k-1}}$. □

4.3.4 Stage 2: Finishing Up the Connected Component Construction

We execute Algorithm 4.1 for $\lceil \log\log\log n \rceil + 3$ phases. Let $\mathcal{T}_\infty$ denote the forest returned by the last phase. Recall that $\mathcal{T}_\infty$ might have fake edges and hence the components induced by $\mathcal{T}_\infty$ may not be connected components. Let $\mathcal{T}_1 = \mathcal{T}_\infty \setminus \{\{u,v\} \in \mathcal{T}_\infty \mid wt(u,v) = \infty\}$ denote the forest with only real edges. A tree T in forest $\mathcal{T}_1$ is called *finished* if it is a spanning tree of a connected component of G; otherwise we call T *unfinished*. We ignore all finished trees in the rest of our algorithm.

Lemma 4.10. *After executing* $\lceil \log\log\log n \rceil + 3$ *phases of Algorithm 4.1 the number of unfinished trees is at most* $O\left(\frac{n}{\log^4 n}\right)$. *Moreover, executing these many phases require* $O(\log\log\log n)$ *rounds and* $O(n \log^{16} n \log\log\log n)$ *messages w.h.p.*

Proof. By Theorem 4.9, after $\lceil \log\log\log n \rceil + 3$ phases the size of the smallest component is at least $\log^4 n$ w.h.p. Observe that if trees $T_1, T_2 \in \mathcal{T}_1$ are connected by a fake edge in $\mathcal{T}_\infty$ then both T_1 and T_2 are finished trees: a fake edge $\{u,v\}$ is added to $\mathcal{T}_\infty$ by v^* if there are no real edges in G which crosses the cut induced by deleting the edge $\{u,v\}$ from $\mathcal{T}_\infty$. Unfinished trees don't contain any fake edges and hence the size of unfinished trees remain the same. Therefore, by Theorem 4.9 there can be at most $O\left(\frac{n}{\log^4 n}\right)$ unfinished trees w.h.p.

By Lemma 4.6, Step 1 of a single phase requires $O(1)$ rounds and $O(n \log^{16} n)$ messages. The rest of steps of a phase finishes in $O(1)$ rounds using $O(n)$ messages. Therefore, to execute $\lceil \log\log\log n \rceil + 3$ phases, we need $O(\log\log\log n)$ rounds and $O(n \log^{16} n \log\log\log n)$ messages. □

These unfinished trees will be viewed as vertices of the graph that will be processed in Stage 2. At the end of this algorithm it is guaranteed that every node knows the ID of the leader of the component it belongs to.

Stage 2 runs on the component graph G_1 induced by unfinished trees in $\mathcal{T}_1$. At a high level, each node in G_1 computes $O(\log n)$ sketches of its neighborhood in G_1 and all the sketches are gathered at a single node v^* so that v^* can locally compute a spanning forest using the received sketches. The challenge is to compute these sketches with respect to G_1. Note that, if each component leader knows the incident inter-component edges (i.e., edges in G_1) then it is trivial to compute sketches with respect to G_1 locally. Also, since G_1 has $O\left(\frac{n}{\log^4 n}\right)$ non-isolated nodes (each non-isolated node corresponds to an unfinished tree) and $O(\log n)$ sketches each of size $O(\log^4 n)$ bits computed for each non-isolated node would result in a total volume of $O(n \log n)$ information to be sent to v^*. Thus *all* sketches can be sent to v^* in $O(1)$ rounds and $O(n)$ messages using the DSG routing scheme (Theorem 4.2). Therefore, it remains to show that how to compute sketches with respect to G_1 without initially having knowledge of the incident edges of G_1.

Consider the components induced by unfinished trees in $\mathcal{T}_1$. Call a component *small* if the number of nodes in the component is at most $\lfloor \sqrt{n} \rfloor$, otherwise call it *large*. Nodes in all small components compute $O(\log n)$ sketches with respect to G and these need to be shipped to the respective component leaders. Since these components are small, the respective component leader has to receive at most $O(\sqrt{n} \cdot \log^4 n \cdot \log n) = O(\sqrt{n} \log^5 n)$ bits and hence can be delivered using the RSG routing scheme in $O(1)$

rounds using $O(n \log^4 n)$ messages. Component leader then merge these sketches to obtain $O(\log n)$ sketches of the component by the linearity property. Now we consider the case of large components. For a large component of size t, we divide the large component into $s = \left\lceil \frac{t}{\sqrt{n}} \right\rceil$ blocks. Each block does the similar things as did by a small component. The sketches of $s = O(\sqrt{n})$ are gathered at the component leader similarly.

Each component leader now has sketches of its neighborhood in G_1. As described earlier, each component leader send these sketches to v^*. The rest of the algorithm is simply local computation by v^* followed by v^* communicating the output, which is of size $O(n)$ in an additional $O(1)$ rounds. The following lemma is easy to prove.

Lemma 4.11. *Stage 2 of GC algorithm requires $O(1)$ rounds of communication and $O(n \log^4 n)$ messages.*

Theorem 4.12 (GC Algorithm)**.** *The GC problem can be solved in $O(\log \log \log n)$ rounds using $O(n \log^{16} n \log \log \log n)$ messages in the Congested Clique model w.h.p.*

4.4 Exact MST Algorithm in $O(\log \log \log n)$ Rounds using $O(m \operatorname{poly} \log n)$ Messages

In this section we present Algorithm Exact-MST which computes an exact MST of a m-edge graph in $O(\log \log \log n)$ rounds using $O(m \operatorname{poly} \log n)$ messages. In the next section we show how to utilize this algorithm along with the GC algorithm to obtain a $(1 + \varepsilon)$-approximation to MST in $O(\log \log \log n)$ rounds using only

$O(n \operatorname{poly} \log n)$ messages.

Similar to the algorithm in [21], Algorithm Exact-MST starts with reducing the number of components and number of edges. First, it reduces the number of component from n to $O\left(\frac{n}{\log^4 n}\right)$ using a variant of the MST algorithm of Lotker et al. in $O(\log \log \log n)$ rounds using $O(m \log \log \log n)$ messages. Then, the number of edges is reduced from m to $O(\sqrt{mn})$ by using the sampling result of Karger, Klein, and Tarjan (KKT sampling) [28] in another $O(1)$ rounds using $O(m)$ messages. The KKT sampling yields two MST subproblems, each with $O(\sqrt{mn})$ edges (and $O(n/\log^4 n)$ vertices). We solve these subproblems in $O(1)$ rounds using $O(m)$ messages each using the distributed sorting and GC algorithm discussed earlier in the chapter.

Karger, Klein, and Tarjan [28] present a randomized linear-time algorithm to find an MST in a edge-weighted graph in a sequential setting (RAM model). They achieved this result via a random edge sampling step to discard edges that cannot be in the MST. For completeness we state their sampling result and the necessary terminology.

Definition (F-light edge [28])**.** *Let F be a forest in a graph G and let $F(u, v)$ denote the path (if any) connecting u and v in F. Let $wt_F(u, v)$ denote the maximum weight of an edge on $F(u, v)$ (if there is no path then $wt_F(u, v) = \infty$). We call an edge $\{u, v\}$ is* F-heavy *if $wt(u, v) > wt_F(u, v)$, and* F-light *otherwise.*

Lemma 4.13 (KKT Sampling Lemma [28])**.** *Let H be a subgraph obtained from G by including each edge independently with probability p, and let F be the minimum spanning forest of H. The number of F-light edges in G is at most n/p w.h.p.*

Observe that if we sample edges with $p = \sqrt{n/m}$ then the number of sampled edges in H and the number of F-light edges in G both are $O(\sqrt{mn})$ w.h.p. Also, none of the F-heavy edges can be in an MST of G. Therefore if we compute a minimum spanning forest F of H, then we can discard all F-heavy edges and it is sufficient to compute a minimum spanning forest of the graph induced by the remaining F-light edges in G. Thus, we have reduced the problem into these subproblems: (i) compute a minimum spanning forest F of H, (ii) compute F-light edges in G, and (iii) compute a minimum spanning forest of the graph induced by these F-light edges. Notice that input to problem (i) and problem (iii) has the same size ($O(\sqrt{mn})$ edges and $O(n/\log^4 n)$ vertices). Though the problems (i) and (iii) are identical we cannot solve them in parallel since input to problem (iii) depends on the output of problem (i). We show that (i) and (iii) can be solved in $O(1)$ rounds and $O(m)$ messages. Identifying F-light edges (problem (ii)) is easy if after problem (i) has been solved every node knows F and can therefore determine which incident edges are F-light. But that will incur message complexity of $O(n^2)$. We develop a simple algorithm that computes F-light edges in $O(1)$ rounds and $O(m)$ messages using DSG scheme (see Lemma 4.16).

The pseudocode of our exact MST computation is outlined in Algorithm 4.2. In the beginning of Exact-MST every node knows weights of incident edges and at the end of the execution every node knows which of its incident edges are part of the MST. Algorithm SQ-MST computes a minimum spanning forest of a graph with $O(n/\log^4 n)$ vertices and $O(\sqrt{mn})$ edges and at the end of the execution of this

algorithm, all nodes know which of its incident edges are in the computed forest. In the next subsection we describe this algorithm and show that it runs in $O(1)$ rounds using $O(m)$ messages w.h.p. The following lemma follows directly from the discussion on Lotker et al. MST algorithm presented in Subsection 4.3.2.

Lemma 4.14. *Step 1 of Algorithm* Exact-MST *can be implemented in* $O(\log \log \log n)$ *rounds using* $O(m \log \log \log n)$ *messages.*

Lemma 4.15. BuildComponentGraph *routine (Step 2 of Algorithm* Exact-MST*) can be implemented using* $O(m)$ *messages and* $O(1)$ *rounds.*

Proof. At the end of Step 1, each node knows its component label with respect to $\mathcal{T}_1$. Each node communicates its label with its neighbors. For each incident edge $\{u, v\}$, node v sends a message to $\ell(u)$ if $\ell(u) \neq \ell(v)$ notifying the existence of an inter-component edge between $\ell(u)$ and $\ell(v)$ to the respective component leaders. Hence in two rounds, each component leader knows the incident inter-component edges. □

Lemma 4.16 (F-light Edge Identification)**.** *Computing F-light edges (Step 5) can be done in* $O(1)$ *rounds using* $O(m)$ *messages.*

Proof. Input to the F-light edge identification algorithm is a spanning forest, i.e., each node knows which of its incident edges are part of F. At the end of the algorithm each node knows which of its incident edges are F-light. Let v^* be the node with ID 0. Let $S \subseteq V$ be the set of nodes with IDs in the range $[0, \lceil \frac{m}{n} \rceil]$. We call nodes in S as *supporters*. All the edges in F are gathered at these supporters in $O(1)$ rounds and $O(|S| \cdot |F|) = O(m)$ messages using DSG scheme (Theorem 4.2). At this point,

all nodes in S have entire F stored locally. Now we can query about m edges by asking any of these supporter nodes. That is, m edges needs to be partitioned into $\left\lceil \frac{m}{n} \right\rceil$ partitions each having at most n edges and each partition is sent to a supporter. Supporter then locally decides which of the edges are F-light and sends the answer back. Above two routing steps can be implemented using DSG scheme in $O(1)$ rounds using $O(m)$ messages. □

4.4.1 MST Algorithm for Sub-problems

In this section, we describe how to compute a minimum spanning forest of a graph with $O(n/\log^4 n)$ nodes and $O(\sqrt{mn})$ edges in $O(1)$ rounds using $O(m)$ messages. The algorithm is similar to the algorithm in [21]. The SQ-MST algorithm in [21] runs in $O(1)$ rounds using $O(n^2)$ messages. Here we improve the algorithm (when $m = o(n^2)$) and reduce the message complexity to $O(m)$.

Lemma 4.17. *Step 3 of Algorithm* SQ-MST *can be executed using* $O(\sqrt{mn})$ *messages in* $O(1)$ *rounds w.h.p.*

Proof. Each node needs to send its incident edges to the respective guardians based on the edge-ranks. Each guardian needs to receive n edges (one guardian may have to receive less than n edges). Note that, because of this RSG scheme cannot be used to deliver these messages. Therefore we design the following algorithm.

Initially all the $O(\sqrt{mn})$ edges are distributed evenly among the n machines based on ranks, that is, an edge with rank $r(e)$ needs to be delivered to machine with ID $r(e) \bmod n$. This way, each machine has to receive at most $O(\sqrt{\frac{m}{n}}) = O(\sqrt{n})$

edges and hence can be delivered using RSG scheme in $O(1)$ rounds and $O(\sqrt{mn})$ messages w.h.p. Then, observe that each machine now has edges which need to be delivered to distinct guardians and hence can be sent directly in one round. The total messages used is twice the number of edges that need to be delivered. □

Lemma 4.18. *Step 6 of Algorithm* SQ-MST *can be executed using $O(n)$ messages in $O(1)$ rounds.*

Proof. Let $p = \left\lceil \sqrt{\frac{m}{n}} \right\rceil$ be the number of guardians (refer Step 2 and 3). For each $i \in [p]$, each node needs to send sketches with respect to G_i to $g(i)$. For each $i \in [p]$, each $g(i)$ needs to receive at most $O(n)$ messages ($O(\log^4 n)$ messages from each of $O\left(\frac{n}{\log^4 n}\right)$ nodes). Note that, because of this RSG routing scheme cannot be used to deliver these messages. Therefore we design the following algorithm which is similar to Lemma 4.17.

Partition n machines into $\lceil \log^4 n \rceil$-size partitions. Let $P(i)$ be the i^{th} partition which has machines with IDs in the range $[i \cdot \lceil \log^4 n \rceil, (i+1) \cdot \lceil \log^4 n \rceil)$, for $i = 0, 1, \ldots, O(n/\log^4 n)$. Arrange machines in each partition based on their IDs. Arrange guardians and senders in ascending order based on their IDs. Each sender arranges its messages based on this order as well, that is, the first $\lceil \log^4 n \rceil$ messages are intended for the first guardian and so on. The i^{th} sender send its j^{th} message intended for k^{th} guardian to j^{th} machine in $P((k+i-1) \bmod p)$. Note that every sender has to send at most $O(1)$ messages to the same machine and hence this can be done in $O(1)$ rounds. Also observe that, every machine gets at most $O(1)$ messages intended for the same guardian and hence delivered in another $O(1)$ rounds. Hence all messages

can be delivered in $O(1)$ rounds using $O(n)$ messages. □

Theorem 4.19 (SQ-MST)**.** *Algorithm* SQ-MST *computes a minimum spanning forest of a graph* $G' = (V', E', wt)$ *with* $|V'| = O\left(\frac{n}{\log^4 n}\right)$ *vertices and* $|E'| = O(\sqrt{mn})$ *edges in* $O(1)$ *rounds and* $O(\sqrt{mn})$ *messages w.h.p.*

Proof. Step 1 can be implemented in $O(1)$ rounds and $O(\sqrt{mn})$ messages using Theorem 4.4. From Lemma 4.17 and Lemma 4.18, Step 3 and Step 6 can be executed using $O(\sqrt{mn})$ messages and $O(1)$ rounds. The rest is local computation followed by communicating output which is of size $O(n)$. □

Theorem 4.20 (Exact MST)**.** *Algorithm* Exact-MST *computes an MST of a* m*-edge input graph in* $O(\log \log \log n)$ *rounds using* $O(m \log \log \log n)$ *messages w.h.p.*

4.5 $(1+\varepsilon)$-Approximate MST in $O(\log \log \log n)$ Rounds using $O(n \operatorname{poly} \log n)$ Messages

In this section we show how to compute a $(1+\varepsilon)$-approximation to MST in $O(\log \log \log n)$ rounds using only $O(n \operatorname{poly} \log n)$ messages. We develop this low-message complexity algorithm using the algorithms developed earlier in this chapter. Specifically, we use GC algorithm and exact MST algorithm as subroutines.

Let W be the max edge weight. Consider the geometric series $W, W/(1+\varepsilon), W/(1+\varepsilon)^2, \ldots$ and round down each edge weight to the nearest element in this sequence. Weights that are less than W/n^2 can be rounded to 0. Thus we get $O(\log n)$ distinct edge weights. We separately consider edges of each weight and run GC algorithm on this subset of edges. We run $O(\log n)$ instances of the GC

algorithm in parallel to compute a maximal forest for each edge-subset (induced by a weight). All of this takes $O(\log\log\log n)$ rounds and uses $O(n \operatorname{poly}\log n)$ messages w.h.p. Now the union of the maximal forests yields a set of $O(n \log n)$ edges. We execute Exact-MST (Algorithm 4.2) on this $O(n \log n)$-edge graph which takes another $O(\log\log\log n)$ rounds and $O(n \operatorname{poly}\log n)$ messages w.h.p. Approx-MST (Algorithm 4.4) presents pseudocode for computing a $(1+\varepsilon)$-approximate MST on graph G.

Lemma 4.21. *$O\left(\frac{\log n}{\log(1+\varepsilon)}\right)$ instances of GC Algorithm (Theorem 4.12) can be executed in parallel on the Congested Clique. The parallel execution requires $O(\log\log\log n)$ rounds and $O(n \operatorname{poly}\log n/\log(1+\varepsilon))$ messages in total.*

Proof. We execute GC algorithm for each instance as it is described in Section 4.3 except that instead of having a single node v^* for all instances, we designate $\Theta(\log n)$ distinct nodes, one for each instance to avoid the congestion. It is easy to see that the rest of the steps can be executed in parallel either as it is or using the routing schemes (Theorem 4.1 and 4.2) as needed. □

Lemma 4.22. *Let M be a minimum spanning tree of G. Then, the tree T returned by* Approx-MST *has the property:*

$$\sum_{e\in M} wt(e) \le \sum_{e\in T} wt(e) \le (1+\varepsilon)\sum_{e\in M} wt(e).$$

Theorem 4.23 (Approximate MST)**.** *Algorithm* Approx-MST *computes a $(1+\varepsilon)$-approximate MST of G in $O(\log\log\log n)$ rounds using $O\left(n \operatorname{poly}\log n/\log(1+\varepsilon)\right)$ messages for any constant $\varepsilon > 0$ w.h.p.*

4.6 Recent Update

Recently Ghaffari and Parter [20] designed an $O(\log^* n)$-round algorithm, using the techniques presented in Chapter 3 but supplemented with the use of *sparsity-sensitive sketching*, which is useful for sparse graphs and *random edge sampling*, which is useful for dense graphs. Similar to the algorithm in Chapter 3 this algorithm has $\Theta(n^2)$ message complexity. Both the MST algorithms (the algorithm presented in Chapter 3 and Gaffari-Parter [20]) have similar communication patterns. Hence the techniques developed in this chapter to reduce the message complexity of our algorithm (Chapter 3) can also be applied to Ghaffari-Parter [20] algorithm to obtain a $O(\log^* n)$-round algorithm using $O(m \operatorname{poly}\log n)$ messages. We call this modified Ghaffari-Parter algorithm as LinearMessages-MST. Below we summarize this result

Theorem 4.24 (LinearMessages-MST)**.** *There exist a MST algorithm that computes a minimum spanning tree of an n-node m-edge input graph in $O(\log^* n)$ rounds using $O(m \operatorname{poly}\log n)$ messages w.h.p. in the Congested Clique.*

4.7 Conclusion

In this chapter, we presented $O(\log\log\log n)$-rounds algorithm for GC and MST that use $O(n \operatorname{poly}\log n)$ and $O(m \operatorname{poly}\log n)$ messages respectively on an m-edge n-node graph. We showed that if constant-factor approximation solution to MST is acceptable then we can solve this problem using only $O(n \operatorname{poly}\log n)$ messages in $O(\log\log\log n)$ rounds. Our results make crucial use of the low-message complexity

routing and sorting algorithm developed in this chapter which we believe will be of independent interest in developing low-message complexity fast algorithms on Congested Clique for variety of problems. We also showed that the recent $O(\log^* n)$-round algorithm [20] can be modified in similar way to reduce the message complexity to $O(m \operatorname{poly} \log n)$ from $\Theta(n^2)$.

We conclude this chapter with the following problem: Can we solve exact MST in sub-logarithmic rounds (ideally in $O(\log^* n)$ rounds) using $o(m)$ (ideally only $O(n \operatorname{poly} \log n)$) messages?

Algorithm 4.1 Phase k of Stage 1 of GC Algorithm executed by a node v which is in component F

Input: A set of edges $\mathcal{T}^{k-1}$ known to v^* ($\mathcal{T}^0 = \emptyset$). The set of connected components induced by $\mathcal{T}^{k-1}$ is $\mathcal{F}^{k-1}$ and for each $F \in \mathcal{F}^{k-1}, |F| \geq 2^{2^{k-2}}$. For each $F \in \mathcal{F}^{k-1}$, if $|F| \geq \lfloor \log^4 n \rfloor$ then F is inactive and all nodes in F are inactive, otherwise it is active (for $k = 0$, all nodes are active). At the start of phase $k \leq \lceil \log \log \log n \rceil + 3$ each node in a component $F \in \mathcal{F}^{k-1}$ knows its component label $\ell(F)$, which is the node with the minimum ID in F and knows whether it is active or inactive. $\ell(F)$ acts as the component leader of F. Every component leader also knows μ - the size of the smallest component.

Output: A set of edges $\mathcal{T}^k$ is known to v^* such that the size of any connected component induced by $\mathcal{T}^k$ is at least $2^{2^{k-1}}$. Each node knows the component label of its component induced by $\mathcal{T}^k$ and its status (active/inactive). Each component leader knows the size of the smallest component induced by $\mathcal{T}^k$ denoted by μ.
▷ v^* is the node in the graph with minimum ID.

1. **if** v is active **then**
2. Compute $t = O(\log^{13} n)$ sketches $\mathbf{s}_v^1, \mathbf{s}_v^2, \ldots, \mathbf{s}_v^t$ with respect to G. Send these sketches to $\ell(F)$ using RSG routing scheme (Theorem 4.1).
3. **end if**
4. **if** $v = \ell(F)$ **then**
5. (a) Using sketches received in Step 1, locally sample either μ edges connecting to μ different active components or an edge to an inactive component (Lemma 4.7 and Lemma 4.8).
(b) Appoint for each sampled edge e, a *guardian* node $g(e)$ in F, such that each node in F is assigned as guardian to at most one edge and send e to $g(e)$
6. **end if**
7. **if** $v = g(e)$ **then** send e to v^*.
8. **if** $v = v^*$ **then**
9. (i) Locally inspects set of received edges and $\mathcal{T}^{k-1}$ to construct a spanning forest $\mathcal{T}^k$.
(ii) Send message to each node notifying it of its (possibly) new component label. If the size of a component F is $|F| > \lfloor \log^4 n \rfloor$ then notify $\ell(F)$ to change its status to inactive.
10. **end if**

Algorithm 4.2 EXACT-MST Algorithm

Input: An edge-weighted n-node, m-edge graph $G = (V, E, w)$. Each node knows weights and end-points of incident edges. Every weight can be represented using $O(\log n)$ bits.

Output: An MST $\mathcal{T}$ of G. Each node in V knows which of its incident edges are part of T.

1. $(\mathcal{F}, \mathcal{T}_1) \leftarrow$ CC-MST$(G, \log\log\log n + 3)$
2. $G_1 \leftarrow$ BUILDCOMPONENTGRAPH$(G, \mathcal{T}_1)$
3. $H \leftarrow$ a subgraph of G_1 obtained by sampling each edge independently with probability $\sqrt{\frac{n}{m}}$
4. $F \leftarrow$ SQ-MST(H)
5. Compute F-light edges
6. $E_\ell \leftarrow \{\{u, v\} \in E(G_1) \mid \{u, v\}$ is F-light$\}$
7. $\mathcal{T}_2 \leftarrow$ SQ-MST(E_ℓ)
8. **return** $\mathcal{T}_1 \cup \mathcal{T}_2$

Algorithm 4.3 SQ-MST

Input: a weighted subgraph $G'(V', E', wt)$ with $O\left(\frac{n}{\log^4 n}\right)$ vertices and $O(\sqrt{mn})$ edges

Output: an MST of G'

1. $r(E') \leftarrow$ DistributedSort(E') in non-decreasing order of edge-weights (Theorem 4.4).
2. Partition edges in E' based on their ranks $r(e)$ into p partitions $E_1, E_2, \ldots E_p$ ($p = \left\lceil \sqrt{\frac{m}{n}} \right\rceil$), each partition having n edges (E_p might have less than n edges) such that E_1 contains edges with ranks $1, 2, \ldots, n$; E_2 contains edges with ranks $n+1, n+2, \ldots,$.
3. Let $g(i)$ be the node in G with ID i and assign $g(i)$ as the guardian of partition i. Gather partition E_i at $g(i)$ (Lemma 4.17).
4. **for** $i = 1$ to $i = p$ **in parallel do**
5. Let $G_i = (V', \cup_{j=1}^{i-1} E_j)$. Each vertex v in V' constructs sketches $\mathbf{s}_v^i$ of its neighborhood with respect to G_i.
6. Each node $v \in V'$ delivers $\mathbf{s}_v^i$ to $g(i)$ (Lemma 4.18).
7. $g(i)$ executes locally:
   (a). $\mathcal{T}_i \leftarrow$ SpanningForest(G_i) (based on sketches received)
   (b). $g(i)$ processes edges in E_i in rank-based order.
   For each edge $e_j = \{u, v\}$ in $e_1, e_2, \ldots$:
   if there is a path between u and v in $\mathcal{T}_i \cup \{e_\ell \mid \ell < j\}$
   discard e_j
   else add e_j to $\mathcal{M}_i$.
8. **end for**
9. **return** $\cup_{i=1}^{p} \mathcal{M}_i$

Algorithm 4.4 APPROX-MST: execution at a node v

Input: A weighted graph $G(V, E, wt)$. Each node only knows weights of incident edges and IDs of neighbors. All nodes know ε (the approximation factor).

Output: A spanning tree T such that $wt(T) = (1+\varepsilon)wt(MST(G))$ for a given $\varepsilon > 0$. Each node knows which of the incident edges are part of T.

1. Let E^v be the set of edges incident on v. Send $\max(wt(e) : e \in E^v)$ to v^*.
2. **if** $v = v^*$ **then**
3. Let W be the maximum of edge weight received in earlier step. Send W to all nodes.
4. **end if**
5. Let $r_0 = 0$ and $r_1 = \frac{W}{n^2}$. Define $r_i = (1+\varepsilon) \cdot r_{i-1}$ for $i = 2, \ldots, \left\lceil 2\frac{\log n}{\log(1+\varepsilon)} \right\rceil$. Let $E_i^v = \{e \mid e \in E_v \text{ and } wt(e) \in [r_i, r_{i+1})\}$.
   ▷ The edge sets E_i^v for all v define $\Theta(\log n)$ different graphs $G_i = (V, E_i)$.
6. **for** $i = 0$ to $i = \left\lceil 2\frac{\log n}{\log(1+\varepsilon)} \right\rceil$ **in parallel do**
7. $F_i \leftarrow$ GCFOREST(G_i) (Theorem 4.12).
8. **end for**▷ The union of F_i induces the graph $G' = (V, F)$, $F = \cup_i F_i$, $|F| = O(n \log n)$.
9. $T \leftarrow$ EXACT-MST(G') (Execute Algorithm 4.2).

CHAPTER 5
SUPER-FAST MINIMUM SPANNING TREE ALGORITHMS USING $O(M)$ MESSAGES

5.1 Introduction

In Chapter 3 we presented a MST algorithm that runs in $O(\log \log \log n)$ rounds but uses $\Theta(n^2)$ messages. Then, in Chapter 4 we showed how to reduce this complexity to $O(m \operatorname{poly} \log n)$ messages without affecting the running time. We concluded the earlier chapter by asking can MST be solved in sub-logarithmic rounds using only $o(m)$ messages. This chapter positively answers this question and presents the first "super-fast" MST algorithm with $o(m)$ message complexity for input graphs with m edges. Specifically, we present an algorithm running in $O(\log^* n)$ rounds, with message complexity $\widetilde{O}(\sqrt{m \cdot n})$ and then build on this algorithm to derive a family of algorithms, containing for any ε, $0 < \varepsilon \leq 1$, an algorithm running in $O(\log^* n/\varepsilon)$ rounds, using $O(n^{1+\varepsilon}/\varepsilon)$ messages. Setting $\varepsilon = \log \log n / \log n$ leads to the first sub-logarithmic round Congested Clique MST algorithm that uses only $\widetilde{O}(n)$ messages.

Our primary tools in achieving these results are (i) a component-wise bound on the number of candidates for MST edges, extending the sampling lemma of Karger, Klein, and Tarjan [28] and (ii) $\Theta(\log n)$-wise-independent linear graph sketches [9] for generating MST candidate edges.

5.1.1 Related Work

The earliest non-trivial example of a Congested Clique algorithm is the *deterministic* MST algorithm that runs in $O(\log \log n)$ rounds due to Lotker et al. [39]. Using *linear sketching* [1, 2, 26, 42, 9] and the *sampling* technique due to Karger, Klein, and Tarjan [28], Hegeman et al. [21] were able to design a substantially faster, *randomized* Congested Clique MST algorithm, running in $O(\log \log \log n)$ rounds. Soon afterwards, Ghaffari and Parter [20] designed an $O(\log^* n)$-round algorithm, using the techniques in Hegeman et al., but supplemented with the use of *sparsity-sensitive sketching*, which is useful for sparse graphs and *random edge sampling*, which is useful for dense graphs.

On the other hand, there are no non-trivial time complexity lower bounds in this model whatsoever and Drucker et al. [13] provide a partial explanation for this by showing that the Congested Clique model can simulate powerful classes of bounded-depth circuits, implying that even slightly super-constant lower bounds in this model would give new lower bounds in circuit complexity. In this context, our focus on time *and* message complexity may prove useful; while unconditional lower bounds on time complexity in the Congested Clique model may currently be out of reach, it may be possible to prove time complexity lower bounds while restricting message complexity. For example, one might hope to show a non-trivial time complexity lower bound for MST conditioned on the algorithm using $\widetilde{O}(n)$ messages.

There has been an increased focus on message complexity in the CONGEST model as well. For example, in PODC 2015 King et al. [30] presented a low

message-complexity MST algorithm in the CONGEST model that uses $\widetilde{O}(n)$ messages, contradicting long-believed "folklore" that MST construction in the CONGEST model would require $\Omega(m)$ messages. One can make a few changes to the King et al. [30] algorithm to run in the Congested Clique model, requiring $O(\log^2 n/\log\log n)$ rounds and $\widetilde{O}(n)$ messages.

5.1.2 Main Results

All of the MST algorithms mentioned above, essentially use the entire bandwidth of the Congested Clique model, i.e., they use $\Theta(n^2)$ messages. From these examples, one might (incorrectly!) conclude that "super-fast" Congested Clique algorithms are only possible when the entire bandwidth of the model is used. In this chapter, we focus on the design of MST algorithms in the Congested Clique model that have low *message complexity*, while still remaining "super-fast." Message complexity refers to the number and size of messages sent and received by all machines over the course of an algorithm; in many applications, this is the dominant cost as it plays a major role in determining the running time and auxiliary resources (e.g., energy) consumed by the algorithm. In our main result, we present an $O(\log^* n)$-round algorithm that uses $\widetilde{O}(\sqrt{m \cdot n})$ messages for an n-node, m-edge input graph. Two points are worth noting about this message complexity upper bound: (i) it is bounded above by $\widetilde{O}(n^{1.5})$ for all values of m and is thus substantially sub-quadratic, independent of m and (ii) it is bounded above by $o(m)$ for all values of m that are super-linear in n, i.e., when $m = \omega(n\,\mathrm{poly}(\log n))$. We then extend this result

to design a family of algorithms parameterized by ε, $0 < \varepsilon \leq 1$, and running in $O(\log^* n/\varepsilon)$ rounds and using $\widetilde{O}(n^{1+\varepsilon}/\varepsilon)$ messages. If we set $\varepsilon = \log\log n/\log n$, we get an algorithm running in $O(\log^* n \cdot \log n/\log\log n)$ rounds and using $\widetilde{O}(n)$ messages. Thus we demonstrate the existence of a sub-logarithmic round MST algorithm using only $O(n \cdot \text{poly}(\log n))$ messages, positively answering a question posed in Hegeman et al. [21]. We note that Hegeman et al. present an algorithm using $O(n \cdot \text{poly}(\log n))$ messages that runs in $O(\log^5 n)$ rounds. All of the round and message complexity bounds mentioned above hold with high probability (w.h.p.), i.e., with probability at least $1 - \frac{1}{n}$. Our results indicate that the power of the Congested Clique model lies not so much in its $\Theta(n^2)$ bandwidth as in the flexibility it provides – any communication link that is needed is present in the network, though most communication links may eventually not be needed.

5.1.3 Applications

Optimizing message complexity as well as time complexity for Congested Clique algorithms has direct applications to the performance of distributed algorithms in other models such as the Big Data (k-machine) model [31], which was recently introduced to study distributed computation on large-scale graphs. Via a Conversion Theorem in [31] one can obtain fast algorithms in the Big Data model from Congested Clique algorithms that have low time complexity *and* message complexity. Another related motivation comes from the connection between the Congested Clique model and the MapReduce model. In [22] it is shown that if a Congested Clique algorithm

runs in T rounds and, in addition, has moderate message complexity then it can be simulated in the MapReduce model in $O(T)$ rounds.

5.2 Technical Preliminaries

5.2.1 Linear Sketches

A key tool used by our algorithm is *linear sketches* [1, 2, 42]. We described this concept in-depth in Chapter 3 (see Section 3.2.1 therein). Since it is a key tool we summarize it below for convenience.

Let $\mathbf{a}_v$ denote a vector whose non-zero entries represent edges incident on v. A *linear sketch* of $\mathbf{a}_v$ is a low-dimensional random vector $\mathbf{s}_v$, typically of size $O(\text{poly}(\log n))$, with two properties: (i) sampling from the sketch $\mathbf{s}_v$ returns a non-zero entry of $\mathbf{a}_v$ with uniform probability (over all non-zero entries in $\mathbf{a}_v$) and (ii) when nodes in a connected component are merged, the sketch of the new "super node" is obtained by coordination-wise addition of the sketches of the nodes in the component. The first property is referred to as ℓ_0-*sampling* in the streaming literature [9, 42, 26] and the second property is *linearity*. The graph sketches used in [1, 2, 42] rely on the ℓ_0-sampling algorithm by Jowhari et al. [26]. Sketches constructed using the Jowhari et al. [26] approach use $\Theta(\log^2 n)$ bits per sketch, but require polynomially many mutually independent random bits to be shared among all nodes in the network. Sharing this volume of information is not feasible; it takes too many rounds and too many messages. So instead, we appeal to the ℓ_0-sampling algorithm of Cormode and Firmani [9] which requires a family of $\Theta(\log n)$-wise independent hash functions, as

opposed to hash functions with full-independence. In the earlier chapter (Chapter 3) we provided details of how the Cormode-Firmani approach can be used in the Congested Clique model to construct graph sketches. We summarize the result in the following theorem.

Theorem 5.1. *Given an input graph $G = (V, E)$, $n = |V|$, there is a Congested Clique algorithm running in $O(1)$ rounds and using $O(n \cdot \text{poly}(\log n))$ messages, at the end of which every node $v \in V$ has computed a linear sketch $\mathbf{s}_v$ of $\mathbf{a}_v$. The size of the computed sketch of a node is $O(\log^4 n)$ bits. The ℓ_0-sampling algorithm on this sketch succeeds with probability at least $1 - n^{-2}$ and, conditioned on success, returns an edge in $\mathbf{a}_v$ with probability in the range $[1/L_v - n^{-2}, 1/L_v + n^{-2}]$, where L_v is the number of non-zero entries in $\mathbf{a}_v$.*

5.2.2 Concentration Bounds for sums of k-wise-independent random variables

The use of k-wise-independent random variables, for $k = \Theta(\log n)$, plays a key role in keeping the time and message complexity of our algorithms low. The use of $\Theta(\log n)$-wise independent hash functions in the construction of linear sketches has been mentioned above. In the next subsection, we discuss the use of $\Theta(\log n)$-wise-independent edge sampling as a substitute for the fully-independent edge sampling of Karger, Klein, and Tarjan. For our analysis we use the following concentration bound on the sum of k-wise independent random variables, due to Schmidt et al. [53] and slightly simplified by Pettie and Ramachandran [52].

Theorem 5.2 (Schmidt et al. [53])**.** *Let $X_1, X_2, \ldots, X_n$ be a sequence of random*

k-wise independent 0-1 random variables with $X = \sum_{i=1}^{n} X_i$. If $k \geq 2$ is even and $C \geq \mathbf{E}[X]$ then:

$$Pr(|X - \mathbf{E}[X]| \geq T) \leq \left[\sqrt{2}\cosh\left(\sqrt{k^3/36C}\right)\right] \cdot \left(\frac{kC}{eT^2}\right)^{k/2}.$$

We use the above theorem for $k = \Theta(\log n)$ and $C = T = \mathbf{E}[X]$. Furthermore, in all instances in which we use this bound, $\mathbf{E}[X] > k^3$ and therefore the contribution of the $\cosh(\cdot)$ term is $O(1)$, whereas the contribution of the second term on the right hand side is smaller than $1/n^c$ for any constant c.

5.3 Algorithmic Overview

The high-level structure of our algorithm is simple. Suppose that the input is an n-node, m-edge graph $G = (V, E)$. We start by sparsifying G by sampling each edge with probability p and compute a *minimum spanning forest* F of the resulting sparse subgraph H. Thus H contains $O(m \cdot p)$ edges w.h.p. Now consider an edge $\{u, v\}$ in G and add it to F; if $F + \{u, v\}$ contains a cycle and $\{u, v\}$ is a heaviest edge in this cycle, then by Tarjan's "red rule" [55] the MST of G does not contain edge $\{u, v\}$. Ignoring all such edges leaves a set of edges that are candidates for being in the MST. We appeal to the well-known sampling lemma due to Karger, Klein, and Tarjan [28] that provides an estimate of the size of this set of candidates.

Definition (F-light edge [28])**.** *Let F be a forest in a graph G and let $F(u, v)$ denote the path (if any) connecting u and v in F. Let $w_F(u, v)$ denote the maximum weight of an edge on $F(u, v)$ (if there is no path then $w_F(u, v) = \infty$). We call an edge $\{u, v\}$ F-*heavy *if $w(u, v) > w_F(u, v)$, and F-*light *otherwise.*

Lemma 5.3 (KKT Sampling Lemma [28]). *Let H be a subgraph obtained from G by including each edge independently* [1] *with probability p and let F be the minimum spanning forest of H. The number of F-light edges in G is at most n/p, w.h.p.*

As our next step we compute the set of F-light edges and in our final step, we compute an MST of the subgraph induced by the F-light edges. Thus, at a high level our algorithm consists of two calls to an MST subroutine on sparse graphs, one with $O(m \cdot p)$ edges and the other with $O(n/p)$ edges. In between, these two calls is the computation of F-light edges. This overall algorithmic structure is clearly visible in Lines 5–7 in the pseudocode in Algorithm 5.1 MST-v1.

There are several obstacles to realizing this high-level idea in the Congested Clique model in order to obtain an algorithm that is "super-fast" and yet has low message complexity. The reason for sparsifying G and appealing to the KKT Sampling Lemma is the expectation that we would need to use fewer messages to compute an MST on a sparser input graph. However, all of the "super-fast" MST algorithms mentioned earlier in the chapter use $\Theta(n^2)$ messages and are insensitive to the number of edges in the input graph. We described how to modify the $O(\log \log \log n)$-round MST algorithm to use $\widetilde{O}(m)$ messages in Chapter 4. Similar modifications applies to the Ghaffari-Parter MST algorithm which allows us to complete the two calls to the MST subroutine in $O(\log^* n)$ rounds using $\max\{O(m \cdot p), O(n/p)\}$ messages. Setting

[1] For reasons that will become clear later, our goal of keeping the message complexity low, does not allow us to assume full independence in this sampling. Instead we use $\Theta(\log n)$-wise independent sampling and show that a slightly weaker version of the KKT Sampling Lemma holds even with limited independence sampling.

the sampling probability p in our algorithm to $\sqrt{\frac{n}{m}}$ balances the two terms in the $\max(\cdot,\cdot)$ and yields a message complexity of $O(\sqrt{m \cdot n})$.

Our *main* contribution (Section 5.5) is to show that the computation of F-light can be completed in $O(1)$ rounds, while still using $\widetilde{O}(\sqrt{m \cdot n})$ messages. To explain the challenge of this computation we present two simple algorithmic scenarios:

- Suppose that we want each node u to perform a local computation to determine which of its incident edges from G are F-light. To do this, node u needs to know $w_F(u,v)$ for all neighbors v. Thus u needs $\text{degree}_G(u)$ pieces of information and overall this approach seems to require the movement of $\Omega(m)$ pieces of information, i.e., $\Omega(m)$ messages.
- Alternately, we might want each node that knows F to be responsible for determining which edges in G are F-light. In this case, the obvious approach is to send queries of the type "Is edge $\{u,v\}$ F-light?" to nodes that know F. This approach also requires $\Omega(m)$ messages.

Various combinations of and more sophisticated versions of these ideas also require $\Omega(m)$ messages. So the fundamental question is how do we determine the status of m edges (i.e., F-light or F-heavy) while exchanging far fewer than m messages? Below we outline two techniques we have developed in order to positively answer this question.

Component-wise bound on number of F-light edges. As mentioned above, the KKT Sampling Lemma upper bounds the total number of F-light edges by $O(n/p)$, which is $O(\sqrt{m \cdot n})$ for $p = \sqrt{n/m}$. We show (in Corollary 5.12)

that a slightly weaker bound (weaker by a logarithmic factor) holds even if the edge-sampling is done using an $\Theta(\log n)$-wise-independent sampler. If we could ensure that the total volume of communication is proportional to the number of F-light edges, we would achieve our goal of $o(m)$ message complexity. To achieve this goal we show that the set of F-light edges has additional structure; they are "evenly distributed" over the components of F. To understand this imagine that F is constructed from H using Borůvka's algorithm. Let $\mathcal{C}^i = \{C_1^i, C_2^i, \ldots\}$ be the set of components at the beginning of a phase i of the algorithm. For each component $C_j^i \in \mathcal{C}^i$, the algorithm picks a *minimum weight outgoing edge (MWOE)* e_j^i from F. Components are merged using edges $e_j^i, j = 1, 2, \ldots$ and we get a new set of components $\mathcal{C}^{i+1}$. Let L_j^i be the set of edges in G leaving component C_j^i *with weight less than* $w(e_j^i)$. We show in Lemma 5.11 that the set of all F-light edges is just the union of the L_j^i's, over all phases i and components j within Phase i. Furthermore, we show in Lemma 5.9 that the size of L_j^i for any i, j is is bounded by $\widetilde{O}(1/p)$. This bound suggests that we could make each component C_j^i responsible for identifying the L_j^i-edges. (Note that we don't use Borůvka's algorithm to compute F because that would take $\Theta(\log n)$ rounds. We compute F in $O(\log^* n)$ rounds using the modified Ghaffari-Parter algorithm (see Chapter 4). Then F is gathered at a small number of nodes and each node who knows F completely simulates Borůvka's algorithm locally on F, thus identifying the components C_j^i and their MWOE's e_j^i.)

Component-wise generation of F-light edges using linear sketches. Linear s-

ketches play a key role in helping nodes in each component C_j^i collectively compute all edges in L_j^i. For any node v and number x, let $N_x(v)$ denote the set of neighbors of v that are connected to v via edges of weight less than x. Each node $v \in C_j^i$ computes a $w(e_j^i)$-*restricted sketch* $\mathbf{s}_v$, i.e., a sketch of its neighborhood $N_{w(e_j^i)}$, and sends it to the component leader of C_j^i who aggregates these sketches to compute a single component sketch. Sampling this sketch yields a single light edge in L_j^i. Since L_j^i has $\widetilde{O}(1/p)$ edges, each node $v \in C_j^i$ can send $\widetilde{O}(1/p)$ separate $w(e_j^i)$-restricted sketches to the component leader of C_j^i and the Coupon Collector argument ensures that this volume of sketches is enough to generate *all* edges incident in L_j^i w.h.p.

The sampling approach of Karger, Klein, and Tarjan is used in a somewhat minor way in earlier Congested Clique MST algorithms [20, 21] and in fact in [33] it is shown that this sampling approach can be replaced by a simple, deterministic sparsification. However, the $\Theta(\log n)$-wise independent sampling we use in the current algorithm seems crucial for ensuring low message complexity, while keeping the algorithms fast.

5.4 MST Algorithms

In this section we describe two "super-fast" MST algorithms, the first runs in $O(\log^* n)$ rounds, using $\widetilde{O}(\sqrt{m \cdot n})$ messages and the second algorithm running in $O(\log^* n/\varepsilon)$ rounds, using $\widetilde{O}(n^{1+\varepsilon}/\varepsilon)$ messages, for any $0 < \varepsilon \leq 1$.

5.4.1 A super-fast algorithm using $\widetilde{O}(\sqrt{mn})$ messages

Our first algorithm MST-V1, shown in Algorithm 5.1 has already been outlined in Section 5.3. The correctness, time complexity, and message complexity of this algorithm depends mainly on two subroutines: LINEARMESSAGES-MST$(\cdot)$ and COMPUTE-F-LIGHT$(\cdot)$. For the purpose of this section, we assume that LINEARMESSAGES-MST(H) computes an MST on an n-node m-edge input graph H in $O(\log^* n)$ rounds using $\widetilde{O}(m)$ messages. This is shown in Chapter 4. We also show that COMPUTE-F-LIGHT(G, F, p) terminates in $O(1)$ rounds using $\widetilde{O}(n/p)$ messages w.h.p. This is the main result in our chapter and is shown in Section 5.5.

Lemma 5.4. *For some constants $c_1, c_2 > 1$, (i) $\Pr(|E(H)| > c_1 \cdot \sqrt{mn}) < \frac{1}{n}$ and (ii) $\Pr(|E_\ell| > c_2 \cdot \sqrt{mn}\,\mathrm{poly}(\log n)) < \frac{1}{n}$.*

Proof. For $0 < i \leq m$, let $X_i = 1$ if edge i is sampled. Hence $|E(H)| = \sum_i X_i$ and $\mathbf{E}[|E(H)|] = \sqrt{mn}$. Note that X_i's are $\Theta(\log n)$-wise independent. Therefore, by Theorem 5.2 we have, $\Pr(|E(H)| > c_1\sqrt{mn}) < \frac{1}{n}$ for some suitable constant $c_1 > 1$. Claim (ii) follows from Corollary 5.12. □

The following theorem summarizes the properties of Algorithm MST-V1. The running time and message complexity bounds follow from Table 5.1.

Theorem 5.5. *Algorithm* MST-V1 *computes an MST of an edge-weighted n-node, m-edge graph G when it terminates. Moreover, it terminates in $O(\log^* n)$ rounds and requires $\widetilde{O}(\sqrt{mn})$ messages w.h.p.*

Algorithm 5.1 MST-v1

Input: An edge-weighted n-node, m-edge graph $G = (V, E, w)$.

▷ Each node knows weights and end-points of incident edges. Every weight can be represented using $O(\log n)$ bits.

Output: An MST $\mathcal{T}$ of G.

▷ Each node in V knows which of its incident edges are part of T.

▷ Let v^* denote the node with lowest ID in V, known to all nodes.

1. v^* generates a sequence π of $\Theta(\log^2 n)$ bits independently and uniformly at random and shares with all nodes in V.
2. $p \leftarrow \sqrt{\frac{n}{m}}$
3. Each node constructs an $\Theta(\log n)$-wise-independent sampler from π and uses this to sample each incident edge in G with probability p
4. $H \leftarrow$ the spanning subgraph of G induced by the sampled edges
5. $F \leftarrow$ LinearMessages-MST(H)
6. $E_\ell \leftarrow$ Compute-F-Light(G, F, p)
7. $\mathcal{T} \leftarrow$ LinearMessages-MST$((V, E_\ell, w))$
8. **return** $\mathcal{T}$

5.4.2 Trading messages and time

The MST-v2 algorithm (shown in Algorithm 5.2) is a recursive version of MST-v1 algorithm yielding a time-message trade-off. The algorithm recurses until the number of edges in the subproblem becomes "low" enough to solve it via a call to the LinearMessages-MST subroutine. Specifically, we treat a n-node graph with $m = O(n^{1+\varepsilon})$ edges as a base case. For graphs with more edges we use a sampling probability of $p = 1/n^\varepsilon$, leading to a sparse graph H with $O(m/n^\varepsilon)$ edges w.h.p.,

which is recursively processed. The use of limited independence sampling is critical here. One simple approach to sampling an edge would be to let the endpoint with higher ID sample the edge and inform the other endpoint *if the outcome is positive*. Unfortunately, this would lead to the use of $\widetilde{O}(m/n^{\varepsilon})$ messages w.h.p., exceeding our target of $\widetilde{O}(n^{1+\varepsilon})$ messages when m is large[2]. Using $\Theta(\log n)$-wise-independent sampling allows us to complete the sampling step using $\widetilde{O}(n)$ messages.

Theorem 5.6. *Algorithm* MST-V2 *outputs an MST of an edge-weighted n-node, m-edge graph when terminates. Moreover, for any $\varepsilon > 0$, it terminates after $O(\log^* n/\varepsilon)$ rounds and uses $\widetilde{O}(n^{1+\varepsilon}/\varepsilon)$ messages, w.h.p.*

Proof. If $m = O(n^{1+\varepsilon})$ then the claim follows from Theorem 4.24. Let $T(m)$ denote the time required for Algorithm 5.2 to compute an MST of a n-node, m-edge graph. Since COMPUTE-F-LIGHT$(\cdot)$ runs in $O(1)$ time and LINEARMESSAGES-MST$(\cdot)$ runs in $O(\log^* n)$ time, we see that, $T(m) = T(m/n^{\varepsilon}) + O(\log^* n)$, for all large m. The first quantity is the result of a recursive call on the sampled graph H, where each edge is sampled with probability $p = 1/n^{\varepsilon}$. Solving this recursion with base case $m = O(n^{1+\varepsilon})$, we get $T(m) = O(\log^* n/\varepsilon)$. The message complexity bound is obtained by similar arguments. □

Setting $\varepsilon = \log\log n/\log n$, we get the following result.

Corollary 5.7. *There exists an algorithm that computes an MST of an n-node, m-*

[2]This approach would have worked fine for MST-V1, but to keep the two algorithms consistent to the extent possible, we use the $\Theta(\log n)$-wise independent sampler there as well.

Algorithm 5.2 MST-v2

Input: An edge-weighted n-node, m-edge graph $G = (V, E, w)$

▷ Each node knows weights and end-points of incident edges in G. Every weight can be represented using $O(\log n)$ bits. There is a parameter $\varepsilon > 0$, known to all nodes.

Output: An MST $\mathcal{T}$ of G.

▷ Each node in V knows which of its incident edges are part of T.

▷ Let v^* denote the node with lowest ID in V and $c \geq 1$ is a constant.

1. **if** $m < c \cdot n^{1+\varepsilon}$ **then**
2. $\quad \mathcal{T} \leftarrow$ LinearMessages-MST(G)
3. $\quad$ **return** $\mathcal{T}$
4. **else**
5. $\quad v^*$ generates a sequence π of $\Theta(\log^2 n)$ bits independently and uniformly at random and shares with all nodes in V
6. $\quad p \leftarrow 1/n^{\varepsilon}$
7. $\quad$ Each node constructs an $\Theta(\log n)$-wise-independent sampler from π and uses this to sample each incident edge in G with probability p
8. $\quad H \leftarrow$ the spanning subgraph of G induced by the sampled edges
9. $\quad F \leftarrow$ MST-v2(H)
10. $\quad E_\ell \leftarrow$ Compute-F-Light(G, F, p)
11. $\quad \mathcal{T} \leftarrow$ LinearMessages-MST$((V, E_\ell, w))$
12. $\quad$ **return** $\mathcal{T}$
13. **end if**

edge input graph and w.h.p. terminates in $O(\log n \cdot \log^* n / \log\log n)$ *rounds and* $\widetilde{O}(n)$ *messages.*

5.5 Efficient Computation of F-light Edges

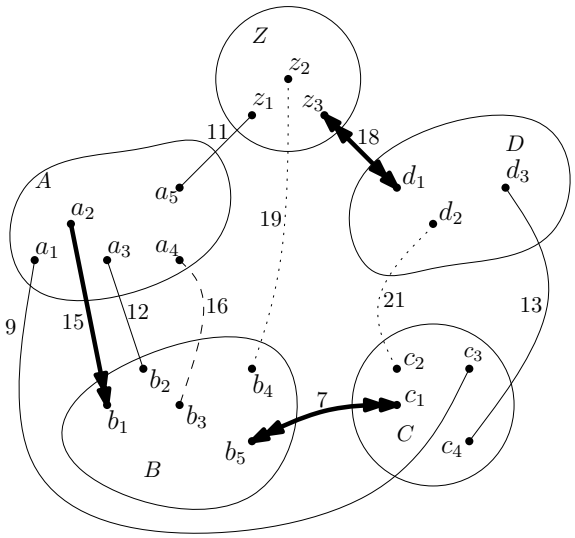
In this section we describe the Compute-F-Light algorithm and prove its correctness and analyze its time and message complexity. The inputs to this algorithm are the graph G, a spanning forest F of G, and a probability p. Recall that F is a minimum spanning forest of the subgraph H obtained by sampling edge in G with probability p, using a $\Theta(\log n)$-wise-independent sampler. The main ideas in Compute-F-Light have been informally described in Section 5.3. The Compute-F-Light algorithm is described below in Algorithm 5.3.

5.5.1 Analysis

Let $\mathcal{C}^i = \{C_1^i, C_2^i, \ldots\}$ be the set of components at the beginning of Phase i of Borůvka's algorithm being simulated on F. Consider the set of edges from G with exactly one endpoint in C_j^i with weight at most $w(e_j^i)$: $L_j^i = \{e = \{u, v\} \in E \mid u \in C_j^i, v \notin C_j^i \text{ and } w(e) \leq w(e_j^i)\}$. For example, see Figure 5.1. Our first task is to bound the size of L_j^i and for this we appeal to the following lemma from Pettie and Ramachandran [52] on sampling from an ordered set.

Lemma 5.8 (Pettie & Ramachandran [52])**.** *Let χ be a set of n totally ordered elements and χ_p be a subset of χ, derived by sampling each element with probability p using a k-wise-independent sampler. Let Z be the number of unsampled elements less than the smallest element in χ_p. Then $\mathbf{E}[Z] \leq p^{-1}(8(\pi/e)^2 + 1)$ for $k \geq 4$.*

Observe that a straight-forward application of the above lemma gives us $\mathbf{E}[|L_j^i|] = O(1/p)$. In the next lemma, we modify the proof of Lemma 5.8 in Pettie &



$\mathcal{C}^i = \{A, B, C, D, Z\}$
$e_1^i = \{a_2, b_1\}$
$e_2^i = \{b_5, c_1\}$
$e_3^i = \{c_1, b_5\}$
$e_4^i = \{d_1, z_3\}$
$e_5^i = \{z_3, d_1\}$
$L_1^i = \{\{a_2, b_1\}, \{a_1, c_3\}, \{a_3, b_2\}, \{a_5, z_1\}\}$
$L_2^i = \{\{b_5, c_1\}\}$
$L_3^i = \{\{c_1, b_5\}\}$
$L_4^i = \{\{d_1, z_3\}, \{d_3, c_4\}\}$
$L_5^i = \{\{z_3, d_1\}, \{z_1, a_5\}\}$

Figure 5.1: Illustration of notation and terminology used in Algorithm 5.3 Compute-F-Light. At the beginning of Phase i of Borůvka's algorithm, there are 5 components $\{A, B, C, D, Z\}$. Each component's MWOE in F is shown as thick directed arc. Solid arcs show edges in G that are in respective L_j^i's and hence identified as being F-light. Dashed arcs (e.g., a_4b_3) represent edges that the algorithm ignores; these edge are not F-light. Dotted arcs (e.g., b_4z_2, c_2d_2) represent edges in G whose status has not yet been resolved by the algorithm. After the merging of components is completed, we end up with two components $\{ABC, DZ\}$.

Ramachandran [52] to obtain a bound on size of L_j^i that holds w.h.p.

Lemma 5.9. $\Pr\left(\textit{There exist } i \textit{ and } j\textit{:}|L_j^i| > c \cdot \log^3 n/p\right) < \frac{1}{n}$ *for some constant* $c > 1$.

Proof. Fix a Phase i and a component C_j^i in that phase. Let X be the set of all edges from G having exactly one endpoint in C_j^i. Let X_t be an indicator random variable

defined as $X_t = 1$ if the t^{th} smallest edge in X is sampled, and 0 otherwise. For any integer ℓ, $1 \leq \ell \leq |X|$, let $S_\ell = \sum_{t=1}^{\ell} X_t$ count the number of ones in $X_1, \ldots, X_\ell$. Note that $L_j^i \subseteq X$ is a set of all edges with weight at most e_j^i, the MWOE from C_j^i in F. This implies that the lightest edge in X that is sampled is e_j^i, otherwise Borůvka's algorithm would have chosen a different MWOE. In other words, $X_k = 0$ for all $k \leq \ell$ if the rank of e_j^i in the ordered set X is $\ell + 1$ or more. Therefore, $\Pr\left(|L_j^i| > \ell\right) = Pr(S_\ell = 0)$.

Observe that, S_ℓ is a sum of 0-1 random variables which are $\Theta(\log n)$-wise-independent and $\mathbf{E}[S_\ell] = p\ell$. By Theorem 5.2, we have $\Pr(S_\ell = 0) < \frac{1}{n^3}$ for $\ell > c \cdot \log^3 n/p$ for some constant $c > 1$. The lemma follows by applying union bound over all phases and components. □

Lemma 5.10. *For any Phase i and any component-MWOE pair (C_j^i, e_j^i), w.h.p. $O\left(\log^5 n/p\right)$ $w(e_j^i)$-restricted sketches of C_j^i are sufficient to find all edges in L_j^i.*

Proof. Consider an oracle which when queried returns an edge in L_j^i independently and uniformly at random. Let T_s denote the number of the oracle queries required to obtain $s = |L_j^i|$ distinct edges (i.e., all edges in L_j^i). Then by the Coupon Collector argument [44], $Pr(T_s > \beta s \log s) < s^{-\beta+1}$ for any $\beta > 1$. Also, if the oracle is not uniform, but is "almost uniform," returning an edge in L_j^i with probability $\frac{1}{s} \pm s^{-\alpha}$ for a constant $\alpha > 2$, then we get $Pr(T_s > \beta s \log s + o(1)) < s^{-\beta+1}$.

Now, to simulate a t^{th} oracle query ($t \in [1, T_s]$) mentioned above, we sample an unused sketch of C_j^i until we get an edge. Since sampling from a sketch fails with probability at most n^{-2}, w.h.p., $O(1)$ sketches are sufficient to simulate one

oracle query. Hence w.h.p., $O(T_s)$ sketches are sufficient to simulate T_s oracle queries. Therefore, with probability at least $1 - s^{-\beta+1}$, $O(\beta s \log s)$ sketches are sufficient to get s distinct edges from L_j^i.

By Lemma 5.9, we have w.h.p., $s = |L_j^i| = O\left(\log^3 n/p\right)$. Therefore by letting $s = \Theta\left(\log^3 n/p\right)$ and $\beta = O(\log n)$ in the above argument, w.h.p., $O\left(\log^5 n/p\right)$ sketches are sufficient to find all edges in L_j^i. □

Lemma 5.11. *Let E_ℓ be the set of F-light edges in G. Let $L = \cup_i \cup_j L_j^i$. Then, $E_\ell = L$.*

Proof. We first show that $L \subseteq E_\ell$. Consider a Phase i and a component-MWOE pair (C_j^i, e_j^i). Consider any edge $e = \{u, v\} \in L_j^i$ with $u \in C_j^i, v \notin C_j^i$. Since e_j^i is the MWOE from C_j^i and $u \in C_j^i$, any path in F connecting u to any node $x \notin C_j^i$ has to go through edge e_j^i. Therefore, for any $x \notin C_j^i, w_F(u, x) \geq w(e_j^i)$. Since $v \notin C_j^i$ we have $w_F(u, v) \geq w(e_j^i)$. Moreover, since $e \in L_j^i$, we have $w(e) \leq w(e_j^i)$ implies $w(e) \leq w_F(u, v)$. Hence, e is F-light. Since this is true for any $e \in L_j^i$, we have $L_j^i \subseteq E_\ell$. Hence, $L \subseteq E_\ell$.

Now, we show that $E_\ell \subseteq L$. For any node $u \in V$, let $C^q(u)$ denote the component containing u just before Phase q of Borůvka's algorithm (Step 2 in Algorithm Compute-F-Light). For the sake of contradiction, let there be an edge $e = \{u, v\} \in E_\ell \setminus L$. Let i be the index of the phase in which component of u and component of v is merged together[3] (that is, for any $q < i + 1$, $C^q(u) \neq C^q(v)$ and

[3]If u and v are never merged into one component, i.e., they are in different components in F then $\{u, v\} \in L_j^i$ where i is the phase in which u's component becomes maximal with

$C^{i+1}(u) = C^{i+1}(v)$). Consider the path $F(u,v)$ and note that since $C^{i+1}(u) = C^{i+1}(v)$, the entire path $F(u,v)$ is in $C^{i+1}(u)$. Now consider the Phase i components $C_1^i, \ldots, C_t^i$, $t \geq 2$ along this path $F(u,v)$ (see Figure 5.2). WLOG, let $u \in C_1^i$ and $v \in C_t^i$ and suppose that the path $F(u,v)$ visits the components in the order $u \in C_1^i, C_2^i, \ldots, C_{t-1}^i, v \in C_t^i$. For example, in Figure 5.2 the path $F(u,v)$ starts in C_1^i then goes through C_2^i, then to C_3^i, and finally to C_4^i. Let $F'(u,v)$ denote the subset of edges in $F(u,v)$ that have endpoints in two distinct Phase i components.

Now consider the MWOE's of these components: e_j^i is the MWOE for C_j^i for $j = 1, 2, \ldots, t$. There are three cases depending on how the MWOEs e_j^i relate to the path $F(u,v)$.

- e_j^i connects C_j^i to C_{j+1}^i for $j = 1, 2, \ldots, t-1$. Since e has exactly one endpoint in C_1^i and $e \notin L_1^i$ (since $e \notin L$), we have $w(e) > w(e_1^i)$. Furthermore, due to the structure of the MWOEs: $w(e_1^i) > w(e_2^i) > \cdots > w(e_{t-1}^i)$. This implies that $w(e)$ is larger than the weights of all edges in $F'(u,v)$.
- e_j^i connects C_j^i to C_{j-1}^i for $j = 2, \ldots, t$. Since e has exactly one endpoint in C_t^i and $e \notin L_t^i$ (since $e \notin L$), we have $w(e) > w(e_t^i)$. Furthermore, due to the structure of the MWOEs: $w(e_t^i) > w(e_{t-1}^i) > \cdots > w(e_2^i)$. This implies that $w(e)$ is larger than the weights of all edges in $F'(u,v)$.
- There is some ℓ, $1 \leq \ell < t$ such that e_j^i connects C_j^i to C_{j+1}^i for $j = 1, 2, \ldots, \ell$ and e_j^i connects C_j^i to C_{j-1}^i for $j = \ell + 1, \ldots, t$. This case is illustrated in

---

respect to F and j is such that u belongs to C_j^i. This follows from the fact that $e_j^i = \perp$ and $w(e_j^i) = \infty$.

Figure 5.2 with $\ell = 2$. In this case, $w(e) > w(e_1^i)$ and $w(e) > w(e_t^i)$ for reasons mentioned in the previous two cases. Furthermore, due to the structure of the MWOEs: $w(e_1^i) > w(e_2^i) > \cdots > w(e_\ell^i)$ and $w(e_t^i) > w(e_{t-1}^i) > \cdots > w(e_{\ell+1}^i)$. This implies that $w(e)$ is larger than the weights of all edges in $F'(u, v)$.

Thus in all three cases, $w(e)$ is larger than the weights of all edges in $F'(u, v)$. Now let $e_F = \{u', v'\} \in F$ be the maximum weight edge in $F(u, v)$. Since e is F-light, we have $w(e) < w(e_F)$. This inequality combined with the fact that $w(e)$ is larger than the weights of all edges in $F'(u, v)$ implies that u' and v' belong to the same Phase i component, i.e., $C^i(u') = C^i(v')$. For example, in Figure 5.2, u' and v' are in C_2^i.

Let $C^i(u') = C^i(v') = C_\ell^i$ for some $\ell \leq t$. Let $F(u, v) = F(u, u') \cup \{u', v'\} \cup F(v', v)$. Since e_F is the heaviest edge in $F(u, v)$, all the edges in $F(u, u')$ are lighter than e_F. Hence at any Phase $i' < i$, Borůvka's algorithm considers edges in $F(u, u')$ for component $C^{i'}(u')$ and edges in $F(v', v)$ for component $C^{i'}(v')$ before considering e_F. The implication of this is, $C^i(u) = C^i(u')$ and $C^i(v) = C^i(v')$. But, $C^i(u) \neq C^i(v)$ therefore, $C^i(u') \neq C^i(v')$ – a contradiction. □

From Lemma 5.9 and Lemma 5.11 we get the following bound on the number of F-light edges in G.

Corollary 5.12. *W.h.p., the number of F-light edges in G is $\widetilde{O}(n/p)$.*

A naive implementation of Step 5 may require super-constant number of rounds because of receiver-side bottlenecks, but we describe here a more sophisticated implementation which runs in $O(1)$ rounds, using $\widetilde{O}(n/p)$ messages.

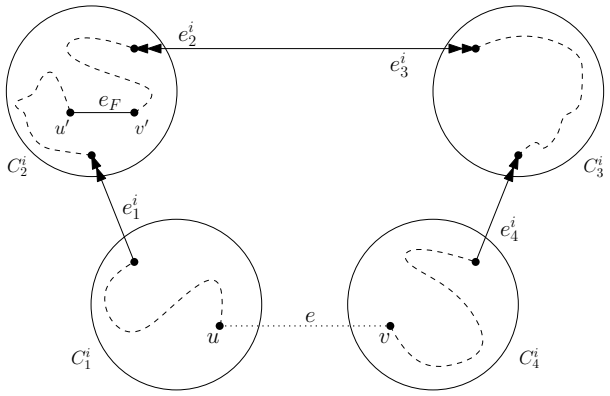


Figure 5.2: Illustration of proof of Lemma 5.11. After Phase i, components $C_1^i, C_2^i, C_3^i, C_4^i$ are merged together using edges $e_1^i, e_2^i, e_3^i, e_4^i$ in F. Dashed curves represent paths in F between the respective end-points. e is an F-light edge. e_F is the heaviest edge on path from u to v in F.

Lemma 5.13. *Step 5 of Algorithm 5.3 can be implemented in $O(1)$ rounds using $\widetilde{O}(n/p)$ messages.*

Proof. A component C_j^i can be quite large and as a result, the volume of sketches of all nodes in C_j^i can be much larger than can be received by C_j^i's component leader in $O(1)$ rounds. So before we can gather sketches at component leaders, we perform two tasks:

(i) each commander v_i sets up a simple *rooted tree* communication structure for each component C_j^i, $j = 1, 2, \ldots$ and

(ii) v_i informs each node in each component C_j^i, $j = 1, 2, \ldots$, the identity of that node's parent in the rooted tree communication structure.

We will show that once these two tasks are completed, then all requisite sketches can

then be gathered at component leaders in $O(1)$ rounds. Of course, we will also need to show that these two tasks can be completed in $O(1)$ rounds.

Recall that each commander v_i knows F and locally simulates Borůvka's algorithm on F and therefore knows the components C_j^i for all j. We will now describe how v_i sets up the rooted tree communication structure for a particular component C_j^i. Let $s := \frac{n^{2/3} \cdot p}{\log^9 n}$ and let $S_0 := C_j^i$. Since $p = \sqrt{n/m}$, we know that p is bounded below by $1/\sqrt{n}$ and therefore $s \geq \frac{n^{1/6}}{\log^9 n}$. This shows that s is asymptotically greater than 1 and for the rest of the proof we assume that $s > 1$. Now commander v_i partitions S_0 into $\lceil |S_0|/s \rceil$ subsets, each of size at most s. For each of the $\lceil |S_0|/s \rceil$ parts, node v_i appoints a *part leader* (e.g., node with smallest ID in that part). Let S_1 be the set of part leaders. Note that $|S_1| = \lceil |S_0|/s \rceil$. Next, commander v_i appoints each part leader as the *parent* of all other nodes in that part.

Now v_i repeats this process on S_1 to construct the set S_2. In other words, v_i partitions S_1 in $\lceil |S_1|/s \rceil$ subsets, each of size at most s, picks part leaders for each of the parts of S_1 (S_2 is the set of these part leaders), and appoints each part leader the parent of all other nodes in its part. Commander v_i continues in this manner until it generates a set S_t such that $|S_t| \leq s$. Commander v_i then picks a leader for S_t and makes it the parent of all other nodes in S_t. We let S_{t+1} denote the singleton set containing this final leader. It it easy to see that the choices of part leaders can be made such that the single node in S_{t+1} is the component leader of C_j^i.

Now note that $|S_{i+1}| = \lceil |S_i|/s \rceil$ for $i = 0, 1, \ldots, t$. Since $|S_0| \leq n$ and $s \geq \frac{n^{1/6}}{\log^9 n}$, it follows that $t = O(1)$ and therefore the rooted tree communication structure we

create for each component has $O(1)$ depth.

Since each part has size at most s, each node in the rooted tree has at most s children. Now consider how the sketches are sent up this rooted tree to the component leader of C_j^i. First, nodes in S_0 are required to send $\Theta\left(\frac{\log^5 n}{p}\right)$ sketches each to their parents, i.e., nodes in S_1. Thus each node in S_1 needs to receive a total of at most

$$s \times \Theta\left(\frac{\log^5 n}{p}\right) \times \Theta(\log^4 n) = \frac{n^{2/3} \cdot p}{\log^9 n} \times \Theta\left(\frac{\log^5 n}{p}\right) \times \Theta(\log^4 n) = \Theta(n^{2/3})$$

bits. This means that we can use the RSG scheme (Theorem 4.1) to deliver all sketches from nodes in S_0 to nodes in S_1 in $O(1)$ rounds, while keeping the number of messages bounded above by $O\left(|C_j^i| \cdot \frac{\log^5 n}{p}\right)$. Once sketches are delivered to nodes in S_1, these nodes will aggregate the sketches. More specifically, suppose that each node v in S_0 organizes the $\Theta\left(\frac{\log^5 n}{p}\right)$ sketches that it sends to its parent, as a vector $(s_1(v), s_2(v), \ldots, s_\beta(v))$ where $\beta = \Theta\left(\frac{\log^5 n}{p}\right)$. Each node w in S_1, on receiving sketch-vectors from children, computes the following size-β vector:

$$\left(\sum_v s_1(v), \sum_v s_2(v), \ldots, \sum_v s_\beta(v)\right).$$

Each of the sums above are over all children v of w (in the rooted tree). Note that the linearity property of the sketches permits this type of aggregation. At the end of this step, nodes in S_1 have a size-β vectors to send to their parents (i.e., nodes in S_2). The above-described process that delivers information from S_0 to S_1 can be used to deliver information from S_1 to S_2, also in $O(1)$ rounds, using $O\left(|C_j^i| \cdot \frac{\log^5 n}{p}\right)$ messages. Thus, this scheme delivers $\beta = \Theta\left(\frac{\log^5 n}{p}\right)$ component sketches to the component leader of C_j^i in $O(1)$ rounds while using $O\left(|C_j^i| \cdot \frac{\log^5 n}{p}\right)$ messages.

The routing scheme we have described above can be executed in parallel for all components in a particular phase, i.e., for C_j^i for a fixed i and all possible j. We now point out that a stronger claim is true: the above-mentioned routing can be accomplished in parallel for all phases as well. This is because there are $O(\log n)$ phases and thus each node has $O(\log n)$ times as much information to send and receive as before (when we were talking about just one phase) and the constraints of the RSG scheme are still met. Thus this routing scheme delivers information needed by each component leader to compute $\Theta\left(\frac{\log^5 n}{p}\right)$ component sketches, in $O(1)$ rounds using $O\left(n \cdot \frac{\log^5 n}{p}\right) = \tilde{O}(n/p)$ messages.

Finally, we point out that the information on the routing tree communication structure can be communicated by the commanders to all nodes in 1 communication round. This is because each commander v_i needs to tell each node v the ID of v's parent in the routing tree, of C_j^i, where v belongs to C_j^i. Thus each commander needs to send n messages to n distinct nodes. Also note that there are $O(\log n)$ phases in Borůvka's algorithm and therefore each node needs to receive messages from $O(\log n)$ distinct nodes (commanders). All this can be done by direct communication in 1 round using $O(n \log n)$ messages. □

Table 5.2 summarizes the time and message complexity of each step of Algorithm Compute-F-Light. From Lemma 5.11 and Table 5.2 we get the following result.

Theorem 5.14. *Algorithm* Compute-F-Light *computes all F-light edges for given graph G and a minimum spanning forest F of H where H is obtained by sampling each*

edge in G with probability p using a $\Theta(\log n)$-wise-independent sampler. Moreover, the computation takes $O(1)$ rounds and uses $\widetilde{O}(n/p)$ messages.

5.6 Conclusion

In this chapter, we presented an algorithm running in $O(\log^* n)$ rounds, with message complexity $\widetilde{O}(\sqrt{m \cdot n})$ and then build on this algorithm to derive a family of algorithms, containing for any ε, $0 < \varepsilon \leq 1$, an algorithm running in $O(\log^* n/\varepsilon)$ rounds, using $O(n^{1+\varepsilon}/\varepsilon)$ messages. Setting $\varepsilon = \log\log n/\log n$ leads to the first sub-logarithmic round Congested Clique MST algorithm that uses only $\widetilde{O}(n)$ messages. We believe that the tools and techniques used to achieve these results can be extended to more general setting – the CONGEST model. The fastest MST algorithm in the CONGEST model requires $O(\sqrt{n} + D)$ rounds but uses $\Omega(m)$ messages. On the other hand, the low-message-complexity algorithm [30] uses $O(n \operatorname{poly}\log n)$ messages but requires $O(n \operatorname{poly}\log n)$ rounds. We believe we can use the tools and techniques developed in this chapter to obtain the first $O(\sqrt{n} + D)$-round $o(m)$-messages MST algorithm in the CONGEST model. We conclude this chapter with the following open problem: Is there a $\widetilde{O}(\sqrt{n}+D)$-round $o(m)$-message MST algorithm in the CONGEST model?

Table 5.1: Time and message complexity for steps in Algorithm 5.1 MST-v1

Step	**Time**	**Messages**	**Analysis**
1	$O(1)$	$\widetilde{O}(n)$	Theorem 4.3
2-4	-	-	Local computation
5	$O(\log^* n)$	$\widetilde{O}(\lvert E(H)\rvert)$	Chapter 4, Theorem 4.24
6	$O(1)$	$\widetilde{O}\left(\sqrt{mn}\right)$	Theorem 5.14 with $p = \sqrt{\frac{n}{m}}$
7	$O(\log^* n)$	$\widetilde{O}(\lvert E_\ell\rvert)$	Chapter 4, Theorem 4.24

Table 5.2: Time and message complexity for steps in Algorithm 5.3 Compute-F-Light

Step	**Time**	**Messages**	**Analysis**
1	$O(1)$	$\widetilde{O}(n)$	Theorem 4.2
2	-	-	Local computation
3	$O(1)$	$\widetilde{O}(n)$	Trivial direct communication
4	$O(1)$	$\widetilde{O}(n/p)$	Theorem 5.1
5	$O(1)$	$\widetilde{O}(n/p)$	Lemma 5.13

Algorithm 5.3 COMPUTE-F-LIGHT

Input: (i) An edge-weighted n-node, m-edge graph $G = (V, E, w)$, (ii) A spanning forest F of G, and (iii) a number p, $0 < p < 1$.

▷ F is a minimum spanning forest of a subgraph H of G, where H is a spanning subgraph of G obtained by sampling each edge in G with probability p using a $\Theta(\log n)$-wise-independent sampler. Each node knows weights and end-points of incident edges from G and F. Every weight can be represented using $O(\log n)$ bits.

Output: F-light edges of G.

▷ Each node in V knows which of its incident edges from G are F-light.

---

1. Let $\{v_1, v_2, \ldots, v_c\}$ be set of *commander nodes* (or in short, *commanders*) where $c = \Theta(\log n)$. Gather F at each of these commanders.
2. Each commander simulates Borůvka's algorithm locally on input graph F. Let $\mathcal{C}^i = \{C_1^i, C_2^i, \ldots\}$ be the set of components at the beginning of Phase i. The node with smallest ID in a component C_j^i is the *leader* of component C_j^i and the ID of the leader serves as the label of each component. For each component $C_j^i \in \mathcal{C}^i$, the algorithm picks a MWOE e_j^i from F. Components are merged and we get a new set of components $\mathcal{C}^{i+1}$. If there is no incident edge on a component C_j^i in F then commander sets $e_j^i = \perp$ with the understanding that $w(\perp) = \infty$.
3. For each component C_j^i, commander v_i sends the following 3-tuple to each node in C_j^i:

   (a) Phase number i, (b) label of C_j^i, and (c) $w(e_j^i)$.
4. A node v having received a 3-tuple (i, ℓ, w') associated with component C_j^i for some i and j computes $\Theta\left(\frac{\log^5 n}{p}\right)$ different graph sketches with respect to its w'-restricted neighborhood $N_{w'}(v)$.
5. The component leader of C_j^i for each i and j, gathers $\Theta\left(\frac{\log^5 n}{p}\right)$ $w(e_j^i)$-restricted sketches from all the nodes in C_j^i and computes $w(e_j^i)$-restricted sketches of C_j^i. Then it samples an edge from each sketch computed and notifies the end-points of all sampled edges.
6. **return** Union of sampled edges over all i over all j.

CHAPTER 6
FUTURE WORK

In this chapter we conclude our report with few open problems related to our work. We plan to use our expertise developed during this work to tackle these open problems in the future.

We provide two sets of problems that we plan to work in the future. The first set of problems is on the more general model – the CONGEST model. The second set of problems is on more specific related distributed models – MapReduce model [29] and the k-machine model a.k.a. the Big Data model [31, 48].

6.1 MST in the Congest Model

Earlier in Chapter 1 we defined the CONGEST model. Here we review the existing work on the MST problem in this model and conclude the section with the open problem.

6.1.1 Related Work

The MST problem in the CONGEST is well studied in the past [17, 4, 35, 51]. The research on the MST problem was initiated by the seminal work of Gallager, Humblet, and Spira [17]. They presented a $O(n \log n)$-round MST algorithm which uses $O(m \log n)$ messages. Later, this result was improved by Awerbuch [4] and the round and message complexity was reduced to $O(n)$ rounds and $O(m + n \log n)$ messages.

Garay et al. [18] initiated the analysis of the time complexity of the MST problem with additional parameters and presented the first sublinear time algorithm, requiring $O(D + n^{0.614})$, where D is the diameter of the input graph. This was further improved to $O(D + \sqrt{n}\log^* n)$ by Kutten and Peleg [35]. In fact, Peleg and Rubinovich [51] proved that this is optimal with respect to time by proving a lower-bound of $\widetilde{\Omega}(\sqrt{n})$ rounds on graphs with $D = \Omega(\log n)$. Recently, Lotker et al. [40] proved lower bound of $\widetilde{\Omega}(n^{1/3})$ rounds on graphs with constant diameters ($D > 2$).

6.1.2 Open Problems

All of the above mentioned sublinear time algorithms requires $\Theta(m + n^{3/2})$ messages. On the other hand, the low-message-complexity algorithm [30] uses $O(n \operatorname{poly}\log n)$ messages but requires $O(n \operatorname{poly}\log n)$ rounds. This leads us to the question whether there exists an MST algorithm that achieves *simultaneously* $\widetilde{O}(m)$ messages and $\widetilde{O}(D + \sqrt{n})$ rounds:

Open Problem 1 (MST in the CONGEST Model)**.** *Is there a $\widetilde{O}(\sqrt{n} + D)$-round $\widetilde{O}(m)$-message (ideally $o(m)$) MST algorithm in the* CONGEST *model?*

We believe we can use the tools and techniques developed in this report (especially, Chapter 5) to obtain the first $O(\sqrt{n} + D)$-round $o(m)$-messages MST algorithm in the CONGEST model.

6.2 MST in the MapReduce Model

First we briefly describe the MapReduce model. Then we review existing work in this model and conclude this section with the open problems.

6.2.1 MapReduce Model

Our description of the MapReduce model follows the work of Karloff et al. [29] and Hegeman et al. [22].

The basic unit of information is a $\langle key, value \rangle$-pair. The input, and all intermediate data, is stored in these $\langle key, value \rangle$-pairs and the computation proceeds in rounds. At a high level, each round is composed of three phases: map, shuffle, and reduce. In the map phase, $\langle key, value \rangle$-pairs are processed individually and the output of this phase is a collection of $\langle key, value \rangle$-pairs. In the shuffle phase, all the $\langle key, value \rangle$-pairs with the same key are aggregated and sent to the same machine. In the reduce phase, each key and all associated values are processed together. Below we elaborate each phase in some more details.

- The map phase of a round is a collection of functions $\{\mu_1, \mu_2, \ldots \mu_m\}$ called *mappers*, one per $\langle key, value \rangle$-pair. Each mapper takes a $\langle key, value \rangle$-pair and outputs a collection of $\langle key, value \rangle$-pairs. Since each mapper is independent of other mappers, the mappers can be arbitrarily distributed among machines.
- The shuffle phase is entirely implemented by the underlying MapReduce framework and is responsible for the data movement after the map phase. Specifically, in the shuffle phase, a $\langle key, value \rangle$-pair $\langle k, v \rangle$ emitted by a mapper is physically moved to the machine which will run the reducer responsible for the key k. We ignore the shuffle phase and consider data movement from one machine to another as a part of the map phase.
- The reduce phase of a round is a collection of functions $\{\rho_1, \rho_2, \ldots, \rho_r\}$ called

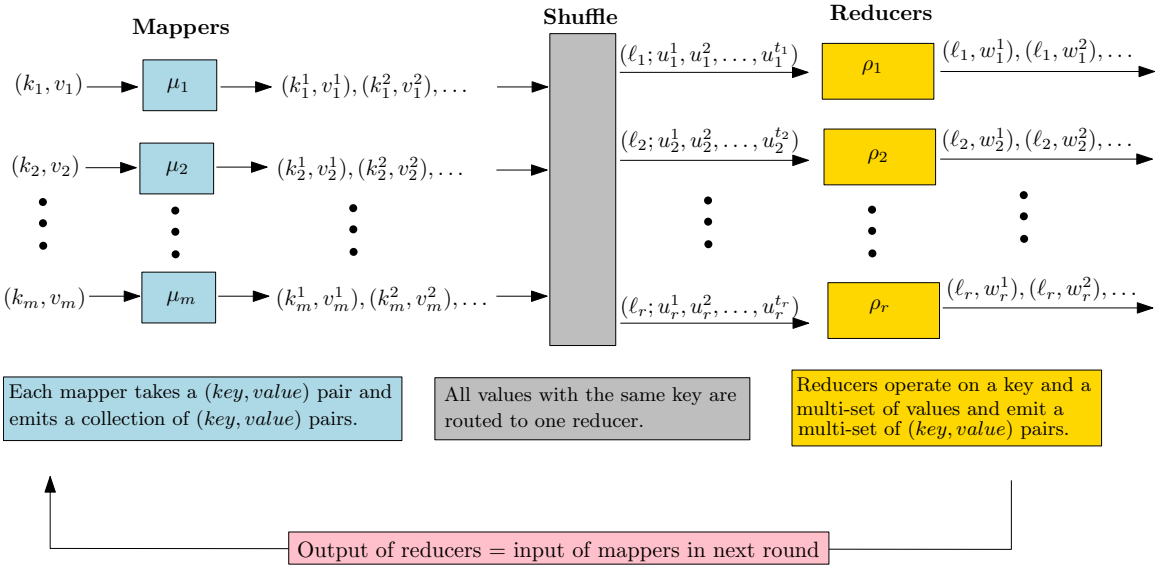


Figure 6.1: This figure describes the execution of the map phase, the shuffle phase, and the reduce phase of a single round of MapReduce program.

reducers. Each reducer operates on an input a pair $\langle k, \{v_{k,i}\}_i \rangle$, where the first element is a key k and the second is a multiset of values which consists of all the values contained in the $\langle key, value \rangle$-pairs emitted by mappers in this round and having key k. The output of each reducer is a multiset $\{\langle k, v_{k,\ell} \rangle\}_\ell$, where the key k in each pair is the same as for the key k of the input.

For further illustration of each of these phases, see Figure 6.1.

Karloff et al. [29] restricted the following resources in the MapReduce model (see [29] for the justification of these constraints). Let n be the size of the input (note that this is total size and not the size of a single mapper's input). We assume, as do Karloff et al. [29] and Lattanzi et al. [36], that memory is measured in $O(\log n)$-bit-sized words. Let $0 < \varepsilon < 1$ be a constant.

Memory: Memory per machine is restricted to $O(n^{1-\varepsilon})$. Moreover, the total space available for any mapper or reducer is constrained to $O(n^{1-\varepsilon})$. This also implies that the size of any $\langle key, value \rangle$-pair is no larger than $O(n^{1-\varepsilon})$.

Machines: The total number of machines is restricted to $O(n^{1-\varepsilon})$, hence the total memory available is $O(n^{2-2\varepsilon})$. This implies that, the total space taken by all $\langle key, value \rangle$-pairs emitted by all the mappers in a round is restricted to $O(n^{2-2\varepsilon})$. It is worth noting that there is no space restriction on the $\langle key, value \rangle$-pairs emitted by reducers.

Time: Each mapper and reducer run in time polynomial in the original input length.

Since the shuffle step is time consuming, we want to minimize the number of MapReduce rounds. The *round complexity* of a MapReduce algorithm is measured in terms of number of MapReduce rounds required to terminate the algorithm.

6.2.1.1 Graph problems in the MapReduce

As discussed earlier the input to the MapReduce framework is made of a list of $\langle key, value \rangle$-pairs distributed among machines. An input graph with m edges is represented as m $\langle key, value \rangle$-pairs where each edge $\{u, v\}$ with weight w is represented as $\langle \{u, v\}; w \rangle$. If the input graph is unweighted then $w = 1$ is used. The input is distributed among all available machines (may not be randomly). For the MST problem, we require that when the algorithm ends each machine knows which of its edges (input $\langle key, value \rangle$-pairs) belong to the output MST; for verification problems such as GC, we require that when the algorithm ends at least one machine

knows the output value.

6.2.2 Related Work

The existing work on MST in the MapReduce differentiate sparse graph instances and dense graph instances. We call a graph G, *c-dense*, if the number of edges m in G is at least n^{1+c} for some $0 < c \leq 1$. Karloff et al. [29] presented a $O(1)$ round algorithm to compute MST of c-dense graphs. Later Lattanzi et al. [36] showed that MST of a c-dense graph can be computed in $\left\lceil \frac{c}{\varepsilon} \right\rceil$ rounds using $O(n^{c-\varepsilon})$ machines with each machine having $O(n^{1+\varepsilon})$ memory for some constant $0 < \varepsilon < c$. For sparse graphs, Karloff et al. [29] presented a $O(\log n)$-round MST algorithm.

6.2.3 Open Problems

In the case of sparse graphs, the question of whether $o(\log n)$ round algorithm exist or not is still open, even in the case of GC problem.

Open Problem 2 (GC/MST problem in the MapReduce model)**.** *Design a $o(\log n)$-round MapReduce algorithm to solve the GC problem or the MST problem for graphs with $m = \Theta(n)$ edges.*

It is worth mentioning that, even for graph problems like *maximal matching*, *min cut*, etc. there are efficient algorithms in the case of dense graphs [36] but when the input graph is sparse then these problems have comparatively high round complexity. Furthermore, consider the following simple problem of counting number of cycles in a 2-degree regular graph: given a n-node graph such that degree of each node is exactly 2 (hence $m = n$), then find the number of cycles in this graph. Observe that there is

a single cycle if and only if the graph is connected. Hence this cycle counting problem can be solved in $O(\log n)$ rounds by using the $O(\log n)$-round GC algorithm. Also, one can solve this problem in $O(\log n)$ rounds using the standard pointer-jumping technique [25]. But the question whether this can be solved in $o(\log n)$ rounds is open.

Open Problem 3. *Given an n-node input graph G such that the degree of each node is 2 (hence $m = n$). Determine whether G has one cycle or multiple cycles in $o(\log n)$ rounds in the MapReduce model.*

Notice that, the above problem is a special case of the GC problem: If this 2-degree regular graph is connected, then it has exactly one cycle, if it is not connected, then there are multiple cycles. There is no known MapReduce algorithm to answer this besides the $O(\log n)$-round GC algorithm and the $O(\log n)$-round pointer-jumping algorithm [25].

6.3 MST in the k-machine Model

First we briefly describe the k-machine model. Then we review existing work in this model and conclude this section with the open problems.

6.3.1 k-machine Model

The k-machine model was first introduced by Klauck et al. [31] to abstract key constraints of large-scale graph processing frameworks such as Pregel [41] and Apache Giraph [15]. Our description of the k-machine model follows the work of Klauck et al. [31] and Pandurangan et al. [48].

The k-machine model consists of a network of $k > 1$ (distinct) machines $N = \{p_1, \cdots, p_k\}$ that are pairwise interconnected by bidirectional point-to-point communication links. Each machine executes an instance of a distributed algorithm. The computation advances in synchronous rounds where, in each round, machines can exchange messages over their communication links. Each link is assumed to have a bandwidth of $O(\log n)$ bits where n is the number of nodes in the input graph. Local computation within a machine is considered free, while communicating messages between the machines is the costly operation. It is assumed that both the computing entities and the communication links are fault-free. The *time (or round) complexity* of a computation is the number of rounds required to complete the computation.

6.3.1.1 Graph problems in the k-machine model

We are interested in solving the minimum spanning tree (MST) problem and the graph connectivity (GC) problem in the k-machine model. The following description is applicable to any graph problem. We are given an input graph G of n vertices and m edges. We assume that $n >> k$. Initially, the entire input graph is not known to a single machine, rather it is "distributed" among the k machines in a "balanced" way. This partition can be done in several ways, but we assume *random vertex-centric partition*, i.e, the n vertices and their incident edges are assigned randomly to machines: each vertex of G is assigned independently and randomly to one of the k machines (see Figure 6.2 for an illustration). If the vertex v is assigned to machine p_i, we call p_i the *home* machine of v. The home machine of

v knows the end-points of all the incident edges and also knows the home machines of these end-points. If G is weighted, then the home machine of v also knows the weights of incident edges on v. We call edge $e = \{u, v\}$ is assigned to machine p_i if p_i is home of either u or v. For the GC problem, at the end of computation, each machine knows whether G is connected or not. For the MST problem, depending upon the requirement at the end of the computation, there are the following variants: (i) for every MST edge $\{u, v\}$ the home machine of u and the home machine of v *must both* eventually know $\{u, v\}$ as being part of the MST, (ii) for every MST edge $e = \{u, v\}$ at least *one* machine knows e as being part of MST, but not necessarily any of the home machines of e.

6.3.2 Related Work

Klauck et al. [31] designed a $O\left(\frac{n}{k} \operatorname{poly} \log n\right)$-round algorithm in the k-machine model such that at the end of the algorithm every machine knows which of its assigned edges are part of the computed MST. Furthermore, they proved a matching lower bound of $\Omega\left(\frac{n}{k}\right)$ rounds for this variant of the MST problem. Recently, Pandurangan et al. [48] broke this $\Omega\left(\frac{n}{k}\right)$ barrier, under the slightly less stringent requirement that at least *one* machine outputs each spanning tree edge e, but not necessarily any of the home machines of e. For this variant of the MST problem, Pandurangan et al. [48] designed a $O\left(\frac{n}{k^2} \operatorname{poly} \log n\right)$-round k-machine algorithm. Moreover, it is shown that there is a $\Omega\left(\frac{n}{k^2}\right)$ lower bound for this variant of the MST problem [48].

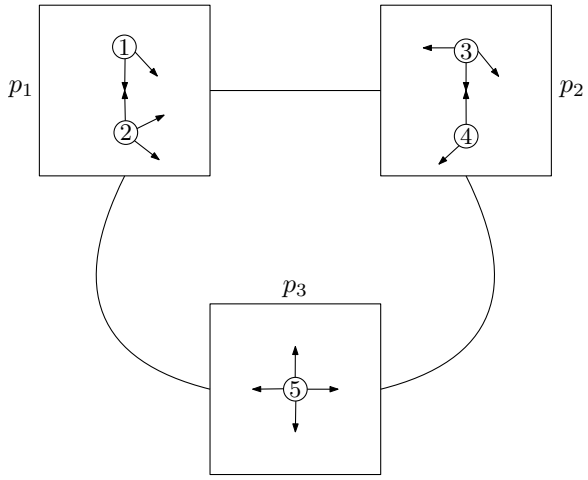


Figure 6.2: This figure illustrates how a 5-node input graph is distributed across $k = 3$ machines. The machines p_1, p_2, and p_3 are shown as rectangles and the communication links between these machines are shown in solid lines. Each machine knows the ID of the assigned nodes, ID neighbors of assigned nodes, and the ID of machines of these neighbors.

6.3.3 Open Problems

Though any Congested Clique algorithm can be simulated in the k-machine model using the Conversion Theorem [31], the obtained k-machine algorithm may not be optimal. Pandurangan et al. [48] obtained a $O\left(\left(\frac{n}{k^2}\right) \operatorname{poly}\log n\right)$ round k-machine algorithm for the MST problem by addressing the problem directly in the k-machine model. But, if k is large, say $k > \sqrt{n}$, then we have a $O(\operatorname{poly}\log n)$-round k-machine algorithm for the MST problem. On the other hand, the lower-bound for the MST problem is $\Omega(\frac{n}{k^2})$, that is, for $k > \sqrt{n}$ it translates to $\Omega(1)$. Hence for large values of k, $O(\operatorname{poly}\log n)$-round algorithm may not be optimal. This motivates us to further study this and investigate the following problem in particular:

Open Problem 4 (GC/MST in the k-machine)**.** *Design a $o(\log n)$-round algorithm for the GC or the MST problem in the k-machine model for some value of $k = \Omega(\sqrt{n})$.*

REFERENCES

[1] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the 23rd annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 459–467, 2012.

[2] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st Symposium on Principles of Database Systems (PODS)*, pages 5–14, 2012.

[3] Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-efficient local computation algorithms. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1132–1139. SIAM, 2012.

[4] Baruch Awerbuch. Optimal Distributed Algorithms for Minimum Weight Spanning Tree, Counting, Leader Election and Related Problems (Detailed Summary). In Alfred V. Aho, editor, *STOC*, pages 230–240. ACM, 1987.

[5] Baruch Awerbuch, Oded Goldreich, Ronen Vainish, and David Peleg. A Trade-off Between Information and Communication in Broadcast Protocols. *J. ACM*, 37(2):238–256, 1990.

[6] Andrew Berns, James Hegeman, and Sriram V. Pemmaraju. Super-Fast Distributed Algorithms for Metric Facility Location. In *Proccedings of the 39th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 428–439, 2012.

[7] Andrew Berns, James Hegeman, and Sriram V. Pemmaraju. Super-Fast Distributed Algorithms for Metric Facility Location. *CoRR*, abs/1308.2473, August 2013.

[8] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic Methods in the Congested Clique. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 143–152. ACM, 2015.

[9] Graham Cormode and Donatella Firmani. A unifying framework for ℓ_0-sampling algorithms. *Distributed and Parallel Databases*, 32(3):315–335, 2014.

[10] Mirela Damian, Saurav Pandit, and Sriram V. Pemmaraju. Distributed Spanner Construction in Doubling Metric Spaces. In *International Conference on Principles of Distributed Systems*, volume 4305 of *OPODIS*, pages 157–171. Springer, 2006.

[11] Danny Dolev, Christoph Lenzen, and Shir Peled. "Tri, Tri Again": Finding Triangles and Small Subgraphs in a Distributed Setting. In *Proceedings of the 26th International Symposium on Distributed Computing (DISC)*, pages 195–209, 2012.

[12] Andrew Drucker, Fabian Kuhn, and Rotem Oshman. The communication complexity of distributed task allocation. In *Proceedings of the 30st ACM Symposium on Principles of Distributed Computing (PODC)*, pages 67–76, 2012.

[13] Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the Power of the Congested Clique Model. In *Proceedings of the 32nd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 367–376, 2014.

[14] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

[15] Apache Software Foundation. Apache Giraph. http://giraph.apache.org/, Accessed on: Jan 2016.

[16] C. Frank. *Algorithms for Sensor and Ad Hoc Networks*. Springer, 2007.

[17] Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A Distributed Algorithm for Minimum-Weight Spanning Trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.

[18] J.A. Garay, S. Kutten, and D. Peleg. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM J. Comput.*, 27(1):302–316, 1998.

[19] Joachim Gehweiler, Christiane Lammersen, and Christian Sohler. A Distributed $O(1)$-approximation Algorithm for the Uniform Facility Location Problem. In *Proceedings of the 18th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 237–243, 2006.

[20] Mohsen Ghaffari and Merav Parter. MST in Log-Star Rounds of Congested Clique. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, 2016.

[21] James W. Hegeman, Gopal Pandurangan, Sriram V. Pemmaraju, Vivek B. Sardeshmukh, and Michele Scquizzato. Toward Optimal Bounds in the Congested Clique: Graph Connectivity and MST. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, PODC '15, pages 91–100. ACM, 2015.

[22] James W. Hegeman and Sriram V. Pemmaraju. Lessons from the Congested Clique Applied to MapReduce. In *Proceedings of the 21th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 149–164, 2014.

[23] James W. Hegeman, Sriram V. Pemmaraju, and Vivek Sardeshmukh. Near-Constant-Time Distributed Algorithms on a Congested Clique. In Fabian Kuhn, editor, *Distributed Computing - 28th International Symposium, DISC 2014*, volume 8784 of *Lecture Notes in Computer Science*, pages 514–530. Springer, 2014.

[24] Stephan Holzer and Nathan Pinsker. Approximation of Distances and Shortest Paths in the Broadcast Congest Clique. *CoRR*, abs/1412.3445, 2014.

[25] Ja Ja Joseph. *An introduction to parallel algorithms*. Addison-Wesley, 1992.

[26] Hossein Jowhari, Mert Sağlam, and Gábor Tardos. Tight Bounds for Lp Samplers, Finding Duplicates in Streams, and Related Problems. In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '11, pages 49–58. ACM, 2011.

[27] Bruce M Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1131–1142, 2013.

[28] David R. Karger, Philip N. Klein, and Robert E. Tarjan. A Randomized Linear-time Algorithm to Find Minimum Spanning Trees. *J. ACM*, 42(2):321–328, March 1995.

[29] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A Model of Computation for MapReduce. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 938–948, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

[30] Valerie King, Shay Kutten, and Mikkel Thorup. Construction and Impromptu Repair of an MST in a Distributed Network with $o(m)$ Communication. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, PODC '15, pages 71–80, New York, NY, USA, 2015. ACM.

[31] Hartmut Klauck, Danupon Nanongkai, Gopal Pandurangan, and Peter Robinson. Distributed Computation of Large-scale Graph Problems. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 391–410. SIAM, 2015.

[32] Ephraim Korach, Shlomo Moran, and Shmuel Zaks. The Optimality of Distributive Constructions of Minimum Weight and Degree Restricted Spanning Trees in a Complete Network of Processors. *SIAM J. Comput.*, 16(2):231–236, 1987.

[33] Janne H. Korhonen. Deterministic MST Sparsification in the Congested Clique. *CoRR*, abs/1605.02022, 2016.

[34] Fabian Kuhn, Thomas Moscibroda, and Rogert Wattenhofer. On the Locality of Bounded Growth. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Principles of Distributed Computing*, PODC '05, pages 60–68. ACM, 2005.

[35] Shay Kutten and David Peleg. Fast Distributed Construction of Small k-Dominating Sets and Applications. *J. Algorithms*, 28(1):40–66, 1998.

[36] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*, pages 85–94. ACM, 2011.

[37] Christoph Lenzen. Optimal Deterministic Routing and Sorting on the Congested Clique. In *Proceedings of the 31st ACM Symposium on Principles of Distributed Computing (PODC)*, pages 42–50. ACM, 2013.

[38] Christoph Lenzen and Roger Wattenhofer. Brief announcement: exponential speed-up of local algorithms using non-local communication. In *Proceedings of the 2010 ACM Symposium on Principles of Distributed Computing*, PODC '10, pages 295–296, 2010.

[39] Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. Minimum-Weight Spanning Tree Construction in $O(\log \log n)$ Communication Rounds. *SIAM Journal on Computing*, 35(1):120–131, 2005.

[40] Zvi Lotker, Boaz Patt-Shamir, and David Peleg. Distributed MST for Constant Diameter Graphs. *Distributed Computing*, 18(6):453–460, 2006.

[41] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A System for Large-scale Graph Processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 135–146. ACM, 2010.

[42] Andrew McGregor. Graph stream algorithms: A survey. *ACM SIGMOD Record*, 43(1):9–20, 2014.

[43] Thomas Moscibroda and Roger Wattenhofer. Facility location: distributed approximation. In *PODC '05: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 108–117. ACM, 2005.

[44] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995.

[45] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proceedings of the 46th ACM Symposium on Theory of Computing (STOC)*, pages 565–573, 2014.

[46] S. Pandit and S. V. Pemmaraju. Finding facilities fast. *Distributed Computing and Networking*, pages 11——24, 2009.

[47] Saurav Pandit and Sriram V. Pemmaraju. Rapid randomized pruning for fast greedy distributed algorithms. In *PODC*, pages 325–334, 2010.

[48] Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. Fast Distributed Algorithms for Connectivity and MST in Large Graphs. In Christian Scheideler and Seth Gilbert, editors, *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*, pages 429–438. ACM, 2016.

[49] Boaz Patt-Shamir and Marat Teplitsky. The Round Complexity of Distributed Sorting. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 249–256, 2011.

[50] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial Mathematics, 2000.

[51] David Peleg and Vitaly Rubinovich. A Near-Tight Lower Bound on the Time Complexity of Distributed Minimum-Weight Spanning Tree Construction. *SIAM J. Comput.*, 30(5):1427–1442, 2000.

[52] Seth Pettie and Vijaya Ramachandran. Randomized minimum spanning tree algorithms using exponentially fewer random bits. *ACM Trans. Algorithms*, 4(1), 2008.

[53] Jeanette P. Schmidt, Alan Siegel, and Srinivasan Aravind. Chernoff-Hoeffding Bounds for Applications with Limited Independence. *SIAM J. Discrete Math.*, 8(2):223–250, 1995.

[54] Johannes Schneider and Roger Wattenhofer. A Log-Star Distributed Maximal Independent Set Algorithm for Growth-Bounded Graphs. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pages 35–44. ACM, 2008.

[55] Robert Endre Tarjan. *CBMS-NSF Regional Conference Series in Applied Mathematics: Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics, New York, NY, USA, 1983.

[56] Leslie G. Valiant. A Bridging Model for Parallel Computation. *Commun. ACM*, 33(8):103–111, August 1990.