Super-fast MST Algorithms in the Congested Clique using $o(m)$ Messages[*]

Sriram V. Pemmaraju Vivek B. Sardeshmukh
Department of Computer Science, The University of Iowa, Iowa City, IA 52242
{sriram-pemmaraju, vivek-sardeshmukh}@uiowa.edu

October 14, 2016

Abstract

In a sequence of recent results (PODC 2015 and PODC 2016), the running time of the fastest algorithm for the *minimum spanning tree (MST)* problem in the *Congested Clique* model was first improved to $O(\log\log\log n)$ from $O(\log\log n)$ (Hegeman et al., PODC 2015) and then to $O(\log^* n)$ (Ghaffari and Parter, PODC 2016). All of these algorithms use $\Theta(n^2)$ messages independent of the number of edges in the input graph.

This paper positively answers a question raised in Hegeman et al., 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 $\tilde{O}(\sqrt{m \cdot n})$ 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 $\tilde{O}(n^{1+\varepsilon}/\varepsilon)$ messages. Setting $\varepsilon = \log\log n/\log n$ leads to the first sub-logarithmic round Congested Clique MST algorithm that uses only $\tilde{O}(n)$ 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 (Karger, Klein, and Tarjan, JACM 1995) and (ii) $\Theta(\log n)$-wise-independent linear graph sketches (Cormode and Firmani, Dist. Par. Databases, 2014) for generating MST candidate edges.

1 Introduction

The *Congested Clique* is a synchronous, message-passing model of distributed computing in which the underlying network is a clique and in each round, a message of size $O(\log n)$ bits can be sent in each direction across each communication link. The Congested Clique is a simple, clean model for studying the obstacles imposed by congestion – all relevant information is nearby in the network (at most 1 hop away), but may not be able to travel to an intended node due to the $O(\log n)$-bit bandwidth restriction on the communication links. There has been a lot of recent work in studying various fundamental problems in the Congested Clique model, including facility location [10, 3], *minimum spanning tree (MST)* [24, 14, 12, 11], shortest paths and distances [4, 15, 27], triangle finding [7, 6], subgraph detection [7], ruling sets [3, 14], sorting [29, 23], and routing [23]. The modeling assumption in solving these problems is that the input graph $G = (V, E)$ is "embedded"

[*]This work is supported in part by National Science Foundation grant CCF 1318166.

in the Congested Clique – 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 (see Section 1.1).

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. [24]. Using *linear sketching* [1, 2, 16, 25, 5] and the *sampling* technique due to Karger, Klein, and Tarjan [17], Hegeman et al. [12] 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 [11] 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.

Our Contributions. 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 paper, 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 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 $\widetilde{O}(\sqrt{m \cdot n})$ [1] 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 $m = \omega(n \operatorname{poly}(\log n))$. 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 Hegeman et al. [12]. We note that Hegeman et al. present an algorithm using $\widetilde{O}(n)$ 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.

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 [19], which was recently introduced to study distributed computation on large-scale graphs. Via a Conversion Theorem in [19] 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 [13] it is shown that if a Congested Clique algorithm runs in

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

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

1.1 Technical Preliminaries

Congested Clique model. The *Congested Clique* is a set of n computing entities (nodes) connected through a complete network that provides point-to-point communication. Each node in the network has a distinct identifier of $O(\log n)$ bits. At the beginning of the computation, each node knows the identities of all n nodes in the network and the part of the input assigned to it. 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 towards 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).

The input graph is assumed to be a spanning subgraph of the underlying communication network. Before the algorithm starts, each node knows the edges of the input graph incident on it and their (respective) weights. We assume that every edge weight can be represented with $O(\log n)$ bits. For ease of exposition, we assume that edge weights are distinct; otherwise, without loss of generality (WLOG) we can "pad" each edge weight with the IDs of the two end points of the edge so as to distinguish the edges by weight while respecting their weight-based ordering. We require that when the algorithm ends, each node knows which of its incident edges belong to the output MST.

Linear Sketches. A key tool used by our algorithm is *linear sketches* [1, 2, 25]. 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 [5, 25, 16] and the second property is *linearity*. The graph sketches used in [1, 2, 25] rely on the ℓ_0-sampling algorithm by Jowhari et al. [16]. Sketches constructed using the Jowhari et al. [16] 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 [5] which requires a family of $\Theta(\log n)$-wise independent hash functions, as opposed to hash functions with full-independence. Hegeman et al. [12] provide details of how the Cormode-Firmani approach can be used in the Congested Clique model to construct graph sketches. We summarize their result in the following theorem.

Theorem 1.1 (Hegeman et al. [12])**.** *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 sketch $\mathbf{s}_v$ 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 $\mathbf{a}_v$.

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. [33] and slightly simplified by Pettie and Ramachandran [31].

Theorem 1.2 (Schmidt et al. [33])**.** *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.

MST with Linear Message Complexity. The "super-fast" MST algorithms mentioned so far [24, 12, 11] use $\Theta(n^2)$ messages, independent of the number of edges in the input graph. One reason for this is that these algorithms rely on deterministic constant-round Congested Clique algorithms for routing and sorting due to Lenzen [23]. Lenzen's algorithms do not attempt to explicitly conserve messages and need $\Omega(n^{1.5})$ messages independent of the number of messages being routed or the number of keys being sorted. However, the above-mentioned MST algorithms do not need the full power of Lenzen's algorithms. We design sorting and routing protocols that work in slightly restricted settings, but use only a linear number of messages (i.e., linear in the total number messages to be routed or keys to be sorted). Details of these protocols appear in Section 4 We use these protocols (instead of Lenzen's protocols) as subroutines in the Ghaffari-Parter MST algorithm [11] to derive a version that uses only linear (up to a polylogarithmic factor) number of messages.

1.2 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 *maximal minimum weight 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 the heaviest edge in this cycle, then by Tarjan's "red rule" [34] 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 [17] (KKT sampling) that provides an estimate of the size of this set of candidates.

Definition (F-light edge [17]). *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 1.3 (KKT Sampling Lemma [17]). *Let H be a subgraph obtained from G by including each edge independently [2] with probability p and let F be the maximal minimum weight 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 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 paper use $\Theta(n^2)$ messages and are insensitive to the number of edges in the input graph. In our first contribution, we develop a collection of simple, low-message-complexity distributed routing and sorting subroutines that we can use in any of the "super-fast" MST algorithms mentioned above [24, 12, 11] (see Section 4) in order to reduce their message complexity to $O(m)$, without increasing their time complexity. Specifically, modifying the Ghaffari-Parter MST algorithm to use these routing and sorting subroutines 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 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 $O(\sqrt{m \cdot n})$. We describe this in Section 4.

Our second and *main* contribution (Section 3) 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.

[2] 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.

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 (i.e., F-light or F-heavy) of m edges while exchanging far fewer than m messages? Below we outline two techniques we have developed in order to 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 3.5) 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 Borŭvka'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 at most* $w(e_j^i)$. We show in Lemma 3.4 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 3.2 that the size of L_j^i for any i, j is is bounded by $\widetilde{O}(1/p)$ w.h.p. This "even distribution" of F-light edges suggests that we could make each component C_j^i responsible for identifying the L_j^i-edges. Note that we don't use distributed Borŭvka's algorithm to compute F because that would take $\Theta(\log n)$ rounds. We compute F in $O(\log^* n)$ rounds using LinearMessages-MST, the modified Ghaffari-Parter algorithm (see Section 4). F is then gathered at each of a small number of nodes and each node who knows F completely simulates Borŭvka'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 sketches 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 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.

Remark: The sampling approach of Karger, Klein, and Tarjan is used in a somewhat minor way in earlier Congested Clique MST algorithms [11, 12] and in fact in [20] it is shown that this sampling approach can be replaced by a simple, deterministic sparsification. However, KKT sampling and specifically its $\Theta(\log n)$-wise independent version that we use in the current algorithm seems crucial for ensuring low message complexity, while keeping the algorithms fast.

1.3 Related Work

It is important to point out that our algorithms are designed for the so-called KT1 [30] model, where every node initially knows the IDs of all its neighbors, in addition to its own ID. (In the Congested Clique model, this means that each node knows the IDs of all n nodes in the network.) If we drop this assumption and work in the so-called KT0 model [30], in which nodes are unaware of IDs of neighbors, then it has been shown in [12] that $\Omega(m)$ messages are needed by any Congested Clique MST algorithm (including randomized Monte Carlo algorithms, and regardless of the number of rounds) on an m-edge input graph. In fact, this lower bound is shown for the simpler graph connectivity problem.

There have also been some recent developments on simultaneously optimizing message complexity and round complexity for the MST problem in the CONGEST model. For example, in [28] it is shown that there exists a randomized (Las Vegas) algorithm that runs in $\widetilde{O}(\sqrt{n} + \text{diameter}(G))$ rounds and uses $\widetilde{O}(m)$ messages (both w.h.p.). This improves the message complexity of the well-known Kutten-Peleg algorithm [22], without sacrificing round complexity (upto polylogarithmic factors). The Kutten-Peleg algorithm runs in $O(\sqrt{n}\log^* n + \text{diameter}(G))$ rounds, while using $O(m + n^{1.5})$ messages. Note that the algorithm in [28] simultaneously matches the round complexity lower bound [9, 32] and the message complexity lower bound [21] for the MST problem.

The above-mentioned upper and lower bound results assume the KT0 model. In the KT1 model, the message complexity lower bound of Kutten et al. [21] does not hold and King et al. [18] were able to design an MST algorithm in the KT1 CONGEST model that uses $\widetilde{O}(n)$ messages, though this algorithm has significantly higher round complexity than $\widetilde{O}(\sqrt{n} + \text{diameter}(G))$ rounds.

As mentioned earlier, Hegeman et al. [12] present a Congested Clique MST algorithm using $\widetilde{O}(n)$ messages, but running in $O(\log^5 n)$ rounds. One can make a few changes to the King et al. [18] CONGEST-model algorithm to implement it in the Congested Clique model, requiring $\widetilde{O}(n)$ messages, but running in $O(\log^2 n/\log\log n)$ rounds.

2 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 $0 < \varepsilon \leq 1$.

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

Our first algorithm MST-v1, shown in Algorithm 1 has already been outlined in Section 1.2. 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 Section 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 paper and is shown in Section 3.

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

Lemma 2.1. *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 1.2 we have, $\Pr(|E(H)| > c_1\sqrt{mn}) < \frac{1}{n}$ for some suitable constant $c_1 > 1$. Claim (ii) follows from Corollary 3.5. □

The following theorem summarizes the properties of Algorithm MST-v1. The running time and message complexity bounds follow from Table 1.

Theorem 2.2. *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.*

2.2 Trading messages and time

The MST-v2 algorithm (shown in Algorithm 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 $m =$

Table 1: Time and message complexity for steps in Algorithm 1 MST-v1

Step	Time	Messages	Analysis
1	$O(1)$	$\widetilde{O}(n)$	Theorem 4.3
2-4	-	-	Local computation
5	$O(\log^* n)$	$\widetilde{O}(\|E(H)\|)$	Theorem 4.5
6	$O(1)$	$\widetilde{O}\left(\sqrt{mn}\right)$	Theorem 3.7 with $p = \sqrt{\frac{n}{m}}$
7	$O(\log^* n)$	$\widetilde{O}(\|E_\ell\|)$	Theorem 4.5

$O(n^{1+\varepsilon})$ 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[3]. Using $\Theta(\log n)$-wise-independent sampling allows us to complete the sampling step using $\widetilde{O}(n)$ messages.

Algorithm 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 $0 < \varepsilon \leq 1$,
          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.
if $m < c \cdot n^{1+\varepsilon}$ then
    $\mathcal{T} \leftarrow$ LinearMessages-MST($G$)
    return $\mathcal{T}$
else
    $v^*$ generates a sequence $\pi$ of $\Theta(\log^2 n)$ bits independently and uniformly at ran-
        dom and shares with all nodes in $V$
    $p \leftarrow 1/n^{\varepsilon}$
    Each node constructs an $\Theta(\log n)$-wise-independent sampler from $\pi$ and uses this
        to sample each incident edge in $G$ with probability $p$
    $H \leftarrow$ the spanning subgraph of $G$ induced by the sampled edges
    $F \leftarrow$ MST-v2($H$)
    $E_\ell \leftarrow$ Compute-F-Light($G, F, p$)
    $\mathcal{T} \leftarrow$ LinearMessages-MST($(V, E_\ell, w)$)
    return $\mathcal{T}$
```

Theorem 2.3. *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 $O\left(\log^* n/\varepsilon\right)$ rounds and uses $\widetilde{O}\left(n^{1+\varepsilon}/\varepsilon\right)$ messages, w.h.p.*

Proof. If $m = O(n^{1+\varepsilon})$ then the claim follows from Theorem 4.5. Let $T(m)$ denote the time required for Algorithm 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.

[3] 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.

Corollary 2.4. *There exists an algorithm that computes an MST of an n-node, m-edge input graph and w.h.p. terminates in $O(\log n \cdot \log^* n / \log\log n)$ rounds and $\widetilde{O}(n)$ messages.*

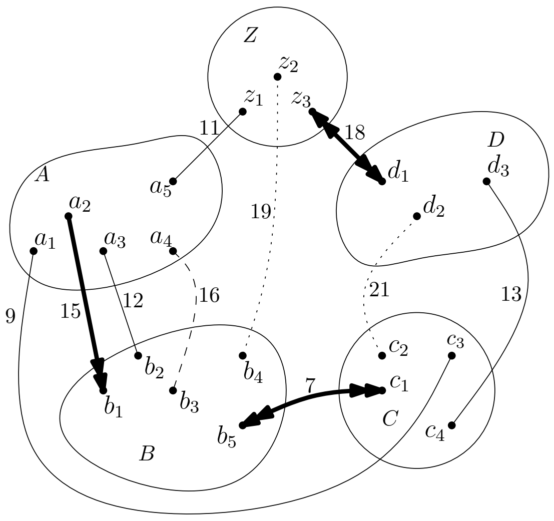
3 Efficient Computation of F-light Edges

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 the maximal minimum weight spanning forest of the subgraph H obtained by sampling edges 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 1.2. The Compute-F-Light algorithm is described below in Algorithm 3.

Algorithm 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 maximal minimum weight 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.

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.
Each commander simulates Borůvka's algorithm locally on input graph $F$. Let
    $\mathcal{C}^i = \{C^i_1, C^i_2, \ldots\}$ be the set of components at the beginning of Phase $i$. The
    node with smallest ID in a component $C^i_j$ is the leader of component $C^i_j$ and
    the ID of the leader serves as the label of each component. For each component
    $C^i_j \in \mathcal{C}^i$, the algorithm picks a MWOE $e^i_j$ 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
    $C^i_j$ in $F$ then commander sets $e^i_j = \perp$ with the understanding that $w(\perp) = \infty$.
For each component $C^i_j$, commander $v_i$ sends the following 3-tuple to each node in
    $C^i_j$:
      (a) Phase number $i$, (b) label of $C^i_j$, and (c) $w(e^i_j)$.
A node $v$ having received a 3-tuple $(i, \ell, w')$ associated with component $C^i_j$ for some $i$
    and $j$ computes $\Theta\left(\frac{\log^5 n}{p}\right)$ different graph sketches with respect to its $w'$-restricted
    neighborhood $N_{w'}(v)$.
The component leader of $C^i_j$ for each $i$ and $j$, gathers $\Theta\left(\frac{\log^5 n}{p}\right)$ $w(e^i_j)$-restricted
    sketches from all the nodes in $C^i_j$ and computes $w(e^i_j)$-restricted sketches of $C^i_j$.
    Then it samples an edge from each sketch computed and notifies the end-points
    of all sampled edges.
return Union of sampled edges over all $i$ over all $j$.
```



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

Figure 1: Illustration of notation and terminology used in Algorithm 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 $\{ABC, DZ\}$.

3.1 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 locally 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 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 [31] on sampling from an ordered set.

Lemma 3.1 (Pettie & Ramachandran [31]). *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 3.1 in Pettie & Ramachandran [31] to obtain a bound on size of L_j^i that holds w.h.p.

Lemma 3.2. $\Pr\left(\textit{There exist } i \textit{ and } j{:}|L_j^i| > c \cdot \log^3 n/p\right) < \frac{1}{n}$ *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^i_j in the ordered set X is $\ell + 1$ or more. Therefore, $\Pr\left(|L^i_j| > \ell\right) = Pr(S_\ell = 0)$.

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 1.2, 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 3.3. *For any Phase i and any component-MWOE pair (C^i_j, e^i_j), w.h.p. $O\left(\log^5 n/p\right)$ $w(e^i_j)$-restricted sketches of C^i_j are sufficient to find all edges in L^i_j.*

Proof. Consider an oracle which when queried returns an edge in L^i_j independently and uniformly at random. Let T_s denote the number of the oracle queries required to obtain $s = |L^i_j|$ distinct edges (i.e., all edges in L^i_j). Then by the Coupon Collector argument [26], $Pr(T_s > \beta s \log s) < s^{-\beta+1}$ for any $\beta > 1$. Also, if the oracle is not uniform, but is "almost uniform," returning an edge in L^i_j 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^i_j 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^i_j.

By Lemma 3.2, we have w.h.p., $s = |L^i_j| = O\left(\log^3 n/p\right)$. 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^i_j. □

Lemma 3.4. *Let E_ℓ be the set of F-light edges in G. Let $L = \cup_i \cup_j L^i_j$. Then, $E_\ell = L$.*

Proof. We first show that $L \subseteq E_\ell$. Consider a Phase i and a component-MWOE pair (C^i_j, e^i_j). Consider any edge $e = \{u, v\} \in L^i_j$ with $u \in C^i_j, v \notin C^i_j$. Since e^i_j is the MWOE from C^i_j and $u \in C^i_j$, any path in F connecting u to any node $x \notin C^i_j$ has to go through edge e^i_j. Therefore, for any $x \notin C^i_j, w_F(u, x) \geq w(e^i_j)$. Since $v \notin C^i_j$ we have $w_F(u, v) \geq w(e^i_j)$. Moreover, since $e \in L^i_j$, we have $w(e) \leq w(e^i_j)$ implies $w(e) \leq w_F(u, v)$. Hence, e is F-light. Since this is true for any $e \in L^i_j$, we have $L^i_j \subseteq E_\ell$. 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[4] (that is, for any $q < i + 1$, $C^q(u) \neq C^q(v)$ and $C^{i+1}(u) = C^{i+1}(v)$). 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 $C^i_1, \ldots, C^i_t$, $t \geq 2$ along this path $F(u, v)$ (see Figure 2). WLOG, let $u \in C^i_1$ and $v \in C^i_t$ and suppose that the path $F(u, v)$ visits the components in the order $u \in C^i_1, C^i_2, \ldots, C^i_{t-1}, v \in C^i_t$. For example, in Figure 2 the path $F(u, v)$ starts in C^i_1 then goes through C^i_2, then to C^i_3, and finally to C^i_4. Let

[4] If u and v are never merged into one component, i.e., they are in different components in F then $\{u, v\} \in L^i_j$ where i is the phase in which u's component becomes maximal with respect to F and j is such that u belongs to C^i_j. This follows from the fact that $e^i_j = \perp$ and $w(e^i_j) = \infty$.

Table 2: Time and message complexity for steps in Algorithm 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 1.1
5	$O(1)$	$\widetilde{O}(n/p)$	Lemma 3.6

$F'(u,v)$ 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^i_j is the MWOE for C^i_j for $j = 1, 2, \ldots, t$. There are three cases depending on how the MWOEs e^i_j relate to the path $F(u,v)$.

- e^i_j connects C^i_j to C^i_{j+1} for $j = 1, 2, \ldots, t-1$. Since e has exactly one endpoint in C^i_1 and $e \notin L^i_1$ (since $e \notin L$), we have $w(e) > w(e^i_1)$. Furthermore, due to the structure of the MWOEs: $w(e^i_1) > w(e^i_2) > \cdots > w(e^i_{t-1})$. This implies that $w(e)$ is larger than the weights of all edges in $F'(u,v)$.

- e^i_j connects C^i_j to C^i_{j-1} for $j = 2, \ldots, t$. Since e has exactly one endpoint in C^i_t and $e \notin L^i_t$ (since $e \notin L$), we have $w(e) > w(e^i_t)$. Furthermore, due to the structure of the MWOEs: $w(e^i_t) > w(e^i_{t-1}) > \cdots > w(e^i_2)$. 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^i_j connects C^i_j to C^i_{j+1} for $j = 1, 2, \ldots, \ell$ and e^i_j connects C^i_j to C^i_{j-1} for $j = \ell+1, \ldots, t$. This case is illustrated in Figure 2 with $\ell = 2$. In this case, $w(e) > w(e^i_1)$ and $w(e) > w(e^i_t)$ for reasons mentioned in the previous two cases. Furthermore, due to the structure of the MWOEs: $w(e^i_1) > w(e^i_2) > \cdots > w(e^i_\ell)$ and $w(e^i_t) > w(e^i_{t-1}) > \cdots > w(e^i_{\ell+1})$. 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 2, u' and v' are in C^i_2.

Let $C^i(u') = C^i(v') = C^i_\ell$ 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 3.2 and Lemma 3.4 we get the following bound on the number of F-light edges in G.

Corollary 3.5. *W.h.p., the number of F-light edges in G is $\widetilde{O}(n/p)$.*

Table 2 summarizes the time and message complexity of each step of Algorithm Compute-F-Light. A naive implementation of Step 5 may require super-constant

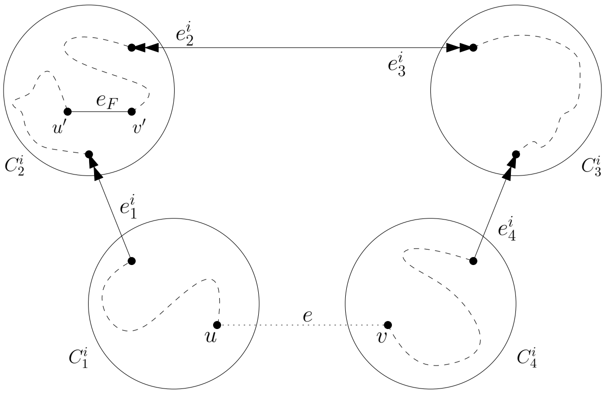


Figure 2: Illustration of proof of Lemma 3.4. After Phase i, components $C_1^i, C_2^i, C_3^i, C_4^i$ are merged together using edges $e_1^i, e_2^i, e_3^i, e_4^i$ in F. Dashed curves represent paths in F between the respective end-points. e is an F-light edge. e_F is the heaviest edge on path from u to v in F.

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.

Lemma 3.6. *Step 5 of Algorithm 3 can be implemented in $O(1)$ rounds using $\widetilde{O}(n/p)$ messages.*

From Lemma 3.4 and Table 2 we get the following result.

Theorem 3.7. *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 $\widetilde{O}\left(n/p\right)$ messages w.h.p.*

4 "Super-Fast" Linear-Message-Complexity MST Algorithms

In this section we first describe three low-message-complexity routing subroutines and then we describe a low-message-complexity sorting subroutine. We apply these subroutines to any of three known "super-fast" Congested Clique MST algorithms [24, 12, 11] to reduce their message complexity to $\widetilde{O}(m)$ while leaving their time complexity unchanged. Specifically, we apply these subroutines to the algorithm of Ghaffari and Parter [11] to obtain an algorithm, we call LinearMessages-MST, that computes an MST of m-edge n-node input graph in $O(\log^* n)$ rounds and using $\widetilde{O}(m)$ messages.

4.1 Routing Subroutines

Many recent Congested Clique algorithms have relied on the deterministic routing protocol due to Lenzen [23] that runs in constant rounds on the Congested Clique. The specific routing problem, called an *Information Distribution Task*, solved by Lenzen's protocol [23] 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. 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 algorithms does not require the full power of Lenzen's routing protocol and our routing primitives suffice for all the routing needs of our algorithms (including LinearMessages-MST which is described later).

Theorem 4.1 (Randomized Scatter-Gather (RSG scheme))**.** *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 $c \cdot n^{1-\epsilon}$ messages, where $\epsilon > 0$ 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 $\lceil 3c/\epsilon \rceil$ 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 $\lceil 3c/\epsilon \rceil$ messages intended for the same destination and hence every intermediate node can deliver all messages to destinations in $\lceil 3c/\epsilon \rceil$ rounds.
Let M_w be the set of messages from all senders intended for w and let $r_w = |M_w| \leq c{\cdot}n^{1-\epsilon}$ 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 $\mathbf{E}[X_w(u)] \leq \frac{cn^{1-\epsilon}}{n} = c{\cdot}n^{-\epsilon}$. 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 [8]. Therefore by Chernoff's bound [8] we have, $\Pr(X_w(u) > c') \leq n^{-2}$ where $c' \leq \lceil 3c/\epsilon \rceil$. By the union bound, with probability at least $1 - n^{-1}$, each intermediate node will receive at most $\lceil 3c/\epsilon \rceil$ messages intended for each node and hence can be delivered in less than $\lceil 3c/\epsilon \rceil$ rounds. □

By using techniques from [3, 6], we obtain the following result for a particular case of the routing problem.

Theorem 4.2 (Deterministic Scatter-Gather (DSG scheme))**.** *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 $2\lceil k/n \rceil + 2$ 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 $2\lceil k/n \rceil + 2$ rounds using $(2k + 2)|V^*|$ messages.*

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 Sorting Subroutine

The Ghaffari and Parter MST algorithm (GP-MST) is partly based on techniques of Hegeman et al. [12] and one of the key ideas there is to sort edges in the input graph based on weights. GP-MST and Hegeman et al. [12] both rely on the $O(1)$-round deterministic sorting routine by Lenzen [23] which requires $\Omega(n^{1.5})$ messages regardless of the number of keys to sort. In addition to the low-message-complexity routing primitive mentioned above, we develop a low-message-complexity sorting primitive (based on the Congested Clique sorting algorithm of [23]).

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 [29] 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 [23]. But, both the algorithms [29, 23] 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 $k = O(n^{2-\epsilon}), \epsilon > 0$ 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 [23] 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 scheme and DSG scheme). 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 $\left(i \mod \lfloor\sqrt{k}\rfloor\right)$. 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 [23] 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 $k = O(n^{2-\epsilon})$ comparable keys of size $O(\log n)$ each such that each node has up to n keys for some constant $\epsilon > 0$. Then, Algorithm* DistributedSort *requires $O(1/\epsilon)$ 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 $\lceil\sqrt{k}\rceil = O(n^{1-\epsilon})$ keys, the keys can be routed using the RSG scheme (Theorem 4.1) in $O(1)$ rounds and $O(k)$ messages. Nodes in V_μ can now execute Lenzen's sorting algorithm [23] 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. □

We obtain the following result by replacing the routing and sorting routines due to Lenzen [23] used in GP-MST with our routing and sorting routines developed above.

Theorem 4.5 (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 $\widetilde{O}(m)$ messages w.h.p. in the Congested Clique.*

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] 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.

[4] 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. URL: http://doi.acm.org/10.1145/2767386.2767414.

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

[6] 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.

[7] 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. URL: http://doi.acm.org/10.1145/2332432.2332443.

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

[9] Michael Elkin. An Unconditional Lower Bound on the Time-Approximation Tradeoff for the Distributed Minimum Spanning Tree Problem. *SIAM J. Comput.*, 36(2):433–456, 2006. URL: http://dx.doi.org/10.1137/S0097539704441058, doi:10.1137/S0097539704441058.

[10] 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 Ar-*

chitectures (SPAA), pages 237–243, 2006. URL: http://doi.acm.org/10.1145/1148109.1148152.

[11] 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.

[12] 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. URL: http://doi.acm.org/10.1145/2767386.2767434.

[13] 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.

[14] James W. Hegeman, Sriram V. Pemmaraju, and Vivek B. Sardeshmukh. Near-Constant-Time Distributed Algorithms on a Congested Clique. In *Proceedings of the 28th International Symposium on Distributed Computing (DISC)*, pages 514–530, 2014.

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

[16] 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. URL: http://doi.acm.org/10.1145/1989284.1989289, doi:10.1145/1989284.1989289.

[17] 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. URL: http://doi.acm.org/10.1145/201019.201022, doi:10.1145/201019.201022.

[18] 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. URL: http://doi.acm.org/10.1145/2767386.2767405, doi:10.1145/2767386.2767405.

[19] Hartmut Klauck, Danupon Nanongkai, Gopal Pandurangan, and Peter Robinson. Distributed Computation of Large-Scale Graph Problems. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 391–410, 2015.

[20] Janne H. Korhonen. Deterministic MST sparsification in the congested clique. *CoRR*, abs/1605.02022, 2016. URL: http://arxiv.org/abs/1605.02022.

[21] Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. On the complexity of universal leader election. *J. ACM*, 62(1):7:1–7:27, March 2015. URL: http://doi.acm.org/10.1145/2699440, doi:10.1145/2699440.

[22] Shay Kutten and David Peleg. Fast Distributed Construction of Small k-Dominating Sets and Applications. *J. Algorithms*, 28(1):40–66, 1998. URL: http://dx.doi.org/10.1006/jagm.1998.0929, doi:10.1006/jagm.1998.0929.

[23] 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. URL: http://doi.acm.org/10.1145/2484239.2501983, doi:10.1145/2484239.2501983.

[24] 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.

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

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

[27] 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.

[28] Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. A time- and message-optimal distributed algorithm for minimum spanning trees. *CoRR*, abs/1607.06883, 2016. URL: http://arxiv.org/abs/1607.06883.

[29] 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. URL: http://doi.acm.org/10.1145/1993806.1993851, doi:10.1145/1993806.1993851.

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

[31] Seth Pettie and Vijaya Ramachandran. Randomized minimum spanning tree algorithms using exponentially fewer random bits. *ACM Trans. Algorithms*, 4(1), 2008. URL: http://doi.acm.org/10.1145/1328911.1328916, doi:10.1145/1328911.1328916.

[32] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed Verification and Hardness of Distributed Approximation. *SIAM J. Comput.*, 41(5):1235–1265, 2012.

[33] 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. URL: http://dx.doi.org/10.1137/S089548019223872X, doi:10.1137/S089548019223872X.

[34] 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.