Toward Optimal Bounds in the Congested Clique: Graph Connectivity and MST

James W. Hegeman[†]
The University of Iowa
Iowa City, IA 52242, USA
james-hegeman
@uiowa.edu

Gopal Pandurangan[*]
University of Houston
Houston, TX 77204, USA
gopalpandurangan
@gmail.com

Sriram V. Pemmaraju[†]
The University of Iowa
Iowa City, IA 52242, USA
sriram-pemmaraju
@uiowa.edu

Vivek B. Sardeshmukh[†]
The University of Iowa
Iowa City, IA 52242, USA
vivek-sardeshmukh
@uiowa.edu

Michele Scquizzato[*]
University of Houston
Houston, TX 77204, USA
michele@cs.uh.edu

ABSTRACT

We study two fundamental graph problems, Graph Connectivity (GC) and Minimum Spanning Tree (MST), in the well-studied *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). 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.

The second contribution of this paper is to present several almost-tight bounds on the *message* complexity of these problems. Specifically, we show that $\Omega(n^2)$ messages are needed by any algorithm (including randomized Monte Carlo algorithms, and regardless of the number of rounds) that solves the GC (and hence also the MST) problem if each machine in the Congested Clique has initial knowledge only of itself (the so-called *KT0* model). In contrast, if the machines have initial knowledge of their neighbors' IDs (the so-called *KT1* model), we present a randomized Monte Carlo algorithm for MST that uses $O(n\,\mathrm{polylog}\,n)$ messages and runs in $O(\mathrm{polylog}\,n)$ rounds. To complement this, we also present a lower bound in the *KT1* model that shows that $\Omega(n)$ messages are required by any algorithm that solves GC, regardless of the number of rounds used. Our results are a step toward understanding the power of randomization in the Congested Clique with respect to both time and message complexity.

Categories and Subject Descriptors

F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General; C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Algorithms, Theory

Keywords

Congested Clique; Graph connectivity; Minimum spanning tree; Graph sketches; Message complexity; Randomization

1. INTRODUCTION

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. Indeed, there has been a lot of recent work in studying various fundamental problems in the Congested Clique model, including facility location [12, 6], minimum spanning tree (MST) [22, 14], shortest paths and distances [7, 15, 25], triangle finding [10, 9], subgraph detection [10], ruling sets [6, 14], sorting [28, 21], and routing [21]. 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 1.2).

Research on the Congested Clique has focused mostly on the *time complexity* (i.e., the number of synchronous *rounds*) of these

[*]Supported, in part, by US-Israel Binational Science Foundation grant 2008348.
[†]Supported, in part, by National Science Foundation grant CCF-1318166.

PODC'15, July 21–23, 2015, Donostia-San Sebastián, Spain.

DOI: http://dx.doi.org/10.1145/2767386.2767434.

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. [22], 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 [29]). 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 [11] 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 [19]. 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 [19], which was recently introduced to study distributed computation on large-scale graphs. The above work shows how to "convert" algorithms (cf. Conversion theorem of [19]) 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. 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 T rounds and, in addition, has moderate message complexity, then it can be simulated in the MapReduce model in $O(T)$ rounds.

1.1 Our Contributions

In this paper 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.

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 2.3).[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. [22]. 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* [3, 2, 23] 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 [3, 2, 23] 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 $O(\log 1/\epsilon)$ 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.

Message Complexity Bounds. Our second contribution is a collection of almost-tight lower bounds on the *message complexity* of GC and MST. 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

[1] This is the case with respect to the standard *unicast/multicast* version of the Congested Clique—the model assumed in this paper—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 [15] and the subgraph detection problem [10].

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

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

complexity of $\Theta(n^2)$ (as does the algorithm of Lotker et al. [22]). 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, in Section 3 we 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 1.2). This result improves on the $\Omega(n^2)$ MST lower bound of Korach et al. [20], 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 1.2), 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. We show that this is optimal in the *KT1* model by showing that $\Omega(n)$ messages are needed by any algorithm, regardless of the running time of the algorithm. These two results hold for all algorithms, including randomized Monte Carlo algorithms, thus extending previously known lower bounds. 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 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 [2] that combines the use of linear sketches with a standard Borůvka-type MST algorithm.

1.2 The Model

The *Congested Clique* is a set of n computing entities connected through a complete network. Specifically, the model consists of a point-to-point network described by a graph $N = (V, E)$ where the vertices V represent independent computing entities pairwise connected through a complete network of $|E| = \binom{n}{2}$ bidirectional communication channels, represented by the edges E of N.

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.

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). It is assumed that both the computing entities and the communication links are fault-free.

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 paper, we consider both complexity measures.

Graph problems in the Congested Clique. In the Congested Clique, the input graph of a graph problem is assumed to be a spanning subgraph of the underlying machine network. Specifically, the input is a spanning subgraph $G = (V, E')$, $E' \subseteq E$, of the underlying clique communication network $N = (V, E)$. 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, and indicates whether or not in E' there is an edge between node v and the endpoint of its i-th channel; when the *KT1* variant of the model is assumed, then the i-th bit 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. Recall that in the *KT1* model v knows the IDs of all its neighbors and can associate these with its communication channels. This definition can be easily generalized to deal with directed and/or weighted input graphs (in the latter case we shall also assume that edge weights are integers that can be represented with $O(\log n)$ bits).

For the MST problem, we require that when the algorithm ends each machine knows which of its incident edges 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.

Input distribution. Notice that in the Congested Clique model the input graph G is tightly coupled with the communication network N and the graph is distributed among the machines via a vertex partition. This is not the case in other related models for distributed graph processing, such as [1, 30, 19]. In these papers the input graph can be much larger than the machine network and the distribution of the graph among machines is via an edge partition. In [19] this edge partition is assumed to be random (initially), in [30] the edge partition can be worst case, whereas in [1] the edge partition is worst case, but with the requirement that each processor has the same number of edges. It is worth noting that [30] does prove message complexity lower bound for problems such as GC, but these lower bounds make crucial use of the worst case distribution of edges and do not apply in our model. Similarly, the lower bounds in the setting of [1] do not seem to directly apply in the Congested Clique model.

2. MST CONSTRUCTION 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. Given this message complexity, the *KT0* and *KT1* model are equivalent since a *KT0* algorithm can start with each node broadcasting its ID to all $n-1$ other nodes. Therefore, for convenience we describe our algorithms in the *KT1* model. 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.

2.1 Linear Sketches of a Graph

A key tool used by our algorithm is *linear sketches* [2, 3, 23]. An important aspect of using sketches for connectivity is working with an appropriate graph representation. As described in [23], 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 [8, 23, 16] and the second property is referred as linearity (hence, the name linear sketches). The graph sketches used in [2, 3, 23] rely on the ℓ_0-sampling algorithm by Jowhari et al. [16]. Sketches constructed using the Jowhari et al. [16] 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. [16] 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 [8] 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 [4]. 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 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 $1/$(non-zero entries in $\mathbf{a_v}$) $+ n^{-2}$.*

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. [22] 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 algorithm [21], 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 2 (LOTKER ET AL. [22]). 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 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. *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 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 part of $\mathcal{T}_\infty$; and in Step 3 we discard these edges. By Theorem 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

Algorithm 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** $(\mathcal{T}_1, G_1)$

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 algorithm. 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 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 betwee 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 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 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. [2].

2.3 Using Linear Sketches to Solve MST

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) [18].

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.

2.3.1 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 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 [18] 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 1 (F-LIGHT EDGE [18]). *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 6 (KKT SAMPLING LEMMA [18]). *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 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. Algorithmn SQ-MST comes with the guarantee that at the end of its execution, all nodes know the MST computed by it.

Algorithm 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** $\mathcal{T}_1 \cup \mathcal{T}_2$

2.3.2 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 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 [21]. 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 $e \in E_i$.

Algorithm 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. **return** $\cup_{i=1}^{p} \mathcal{M}_i$

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 algorithm, 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 algorithm, and thus we obtain the following theorem.

THEOREM 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. MESSAGE LOWER BOUNDS IN THE *KT0* MODEL

In this section we show that, at least in the *KT0* model, it is impossible to avoid the $\Omega(n^2)$ message complexity of the algorithms of the previous section. We show that any algorithm that solves GC, even a randomized Monte Carlo algorithm, needs to send $\Omega(n^2)$ messages. In the following, we use GC(m) to denote the problem of determining, in the *KT0* model, whether or not an input graph with m edges is connected.

3.1 Construction

To show a lower bound on the message complexity of Monte Carlo algorithms, we construct a hard distribution H on the inputs and use Yao's Minimax Principle (Prop. 2.6 in [24]).

We assume without (much) loss of generality that n is even. Consider any m, $n \leq m \leq \frac{n}{2} \cdot (\frac{n}{2} - 1)$. Then, for a given n, partition the n nodes into two subsets U and V of size $n/2$. Next, we construct biconnected (2-connected) graphs G_U and G_V on U and V, respectively. Assume w.l.o.g. that the nodes of U are $u_0, u_1, \ldots, u_{n/2-1}$ and the nodes of V are $v_0, v_1, \ldots, v_{n/2-1}$. We next use m edges to construct our graph $G = G_U \cup G_V$.

Since m should be at least as large as n, with the first n edges of G we connect u_j to u_{j+1} and v_j to v_{j+1} for each j (where indices are interpreted $\bmod n/2$). If $m \geq 2n$, we connect u_j to u_{j+2} and v_j to v_{j+2} for each j; and in general, if $m \geq k \cdot n$ we connect u_j to u_{j+k} and v_j to v_{j+k} for each j. This procedure is performed for each $k = 1, 2, \ldots$ until $k = k'$ such that $m < k' \cdot n$. With the final $m - (k'-1) \cdot n$ "leftover" edges, we connect u_0 to $u_{k'}$, u_1 to $u_{1+k'}$, u_2 to $u_{2+k'}$, etc. in that order until we run out of edges to add to the graph. Let G_U be the graph on the nodes U and G_V be the graph on the nodes V constructed in this manner. Then $G = G_U \cup G_V$ is a nearly-regular disconnected graph composed of two biconnected components G_U and G_V. Assuming n is even, every vertex has degree $\lfloor \frac{2m}{n} \rfloor$ or $\lceil \frac{2m}{n} \rceil$.

We next construct the input distribution H. One-half of the probability mass of H will be placed on the (disconnected) graph $G = G_U \cup G_V$. The other $1/2$ of the probability mass will be distributed uniformly among the members of the following set S_G. An element of S_G is formed by choosing two edges of G – one from G_U and one from G_V – and "swapping" the endpoints. That is, suppose $e_1 = (u_1, u_2) \in G_U$ and $e_2 = (v_1, v_2) \in G_V$ are chosen in G. For each such pair e_1, e_2, add the following graphs to S_G: $G - e_1 - e_2 + (u_1, v_1) + (u_2, v_2)$ and $G - e_1 - e_2 + (u_1, v_2) + (u_2, v_1)$. Since G_U and G_V are biconnected, each member of S_G is connected (in contrast to $G = G_U \cup G_V$, which is disconnected). The half of the probability mass of H assigned to S_G is distributed uniformly over the elements of S_G.

3.2 Lower Bounds

We first show how to extend the lower bound argument of Korach et al. [20] for deterministic MST algorithms in *KT0* to GC.

THEOREM 8. *Let $\mathcal{A}_D$ be a deterministic algorithm solving* GC*(m) in the KT0 model, where $n \leq m \leq \frac{n}{2} \cdot (\frac{n}{2} - 1)$. Then the worst-case number of messages sent during the execution of $\mathcal{A}_D$ is $\Omega(m)$.*

PROOF. As in [20], consider the execution of $\mathcal{A}_D$ on G, and suppose that there exists a "square" u_1, v_1, v_2, u_2, with (u_1, u_2) an edge in G_U and (v_1, v_2) an edge in G_V, such that during the execution of $\mathcal{A}_D$, no messages are sent along the communication links (u_1, u_2), (u_1, v_1), (v_1, v_2), or (u_2, v_2). As Korach et al. do in [20], we claim that if this is the case, then the execution of $\mathcal{A}_D$ proceeds *identically* on the input $G' = G - (u_1, u_2) - (v_1, v_2) + (u_1, v_1) + (u_2, v_2)$ (which is an element of S_G). To see this point, note that in the *KT0* model, no node (including u_1) can distinguish between the situation where an edge leaving u_1 at a certain "port" goes to v_1 and the situation where the edge leaving that "port" goes to u_2. This is true for the other edges and nodes involved in the "square." Therefore, since the output of $\mathcal{A}_D$ must be different on G (disconnected) and G' (connected), it cannot be the case that there exists such a "square" in the execution of $\mathcal{A}_D$ on G. To complete the proof, as in [20], one can show that there exists a set of at least $\Omega(m)$ edge-disjoint "squares" and hence there is no set of edges of size less than $\Omega(m)$ that can intersect all such G_U-G_V squares. In summary, then, in order to be correct on G and all members of S_G, $\mathcal{A}_D$ must use a set of links, during its execution on G, of size $\Omega(m)$, and hence must have worst-case message complexity $\Omega(m)$. □

We now combine the above argument with the hard distribution H (defined in previous subsection) to obtain a lower bound on deterministic algorithms that err with (at most) a certain constant probability. Then, by applying Yao's Minimax Principle we obtain a lower bound on the message complexity of Monte Carlo randomized algorithms.

THEOREM 9. *Let $\mathcal{A}_{MC}$ be a Monte Carlo algorithm solving* GC*(m) with probability at least $4/5$ in the KT0 model, where $n \leq m \leq \frac{n}{2} \cdot (\frac{n}{2} - 1)$. Then the expected number of messages sent during the execution of $\mathcal{A}_{MC}$ (on a worst-case input) is $\Omega(m)$.*

PROOF. To show a lower bound on the (expected) message complexity of $\mathcal{A}_{MC}$, by Yao's Minimax Principle (Prop. 2.6 in [24]), it suffices to consider deterministic algorithms that err with probability at most $2/5$ (i.e., twice the error probability of $\mathcal{A}_{MC}$) on the hard distribution on the input. Consider any such deterministic algorithm $\mathcal{A}_D$. Since $\mathcal{A}_D$ is correct with probability at least $3/5$, we conclude that (i) $\mathcal{A}_D$ is correct on at least one-tenth of the mass assigned to G (implying $\mathcal{A}_D$ is correct on G, since $\mathcal{A}_D$ is deterministic); and (ii) $\mathcal{A}_D$ is correct on at least one-tenth of the input instances in S_G.

The correctness of $\mathcal{A}_D$ on G together with at least one-tenth of the instances of S_G allows a re-use of the analysis for deterministic case (see proof of Theorem 8). The result follows. □

4. MESSAGE COMPLEXITY IN THE *KT1* MODEL

We start with the simple, but important observation that n bits of communication suffice to solve MST (or any problem, for that matter) in the *KT1* model. The idea is that each node u can view the initial information it possesses as number r_u and simply send a bit to a "leader" (e.g., node with smallest ID) in round r_u. The leader can solve the problem locally and communicate the solution back in a similar manner using an additional n bits. Hence, all the information is encoded using the synchronized clock. This sets up a significant contrast with the *KT0* model, where we have shown an $\Omega(n^2)$ lower bound on the number of messages needed to solve GC, even assuming a synchronous setting. We now show an unconditional, matching lower bound in the *KT1* model, namely that any algorithm (even randomized Monte Carlo) for solving GC (and hence MST) needs to communicate $\Omega(n)$ bits of information. In fact, what we show is stronger: we show that at least $\Omega(n)$ distinct messages are sent, in the worst case, during the course any algorithm that solves GC.

4.1 Message Lower Bounds in the *KT1* Model

Definitions. We define the $GC(x, y)$ problem, in the *KT1* variant of the Congested Clique model, to be the problem of determining the connectivity of an input graph (with the usual assumption that nodes have initial knowledge of their incident edges and neighbors in the input graph—as GC has been posed previously), with the extra condition that, during the last round of any algorithm solving $GC(x, y)$, node x must send a message to node y containing the final output ("connected" or "disconnected", as 1 or 0). The identities of x and y are known to all nodes as part of the input to $GC(x, y)$.

We now define a set of forests, $\{G_{i,j}\}$, for $i \geq 1$ and $0 \leq j \leq i + 1$. In $G_{i,j}$, there are $n = 2i+2$ nodes $\{u_0, u_1, \ldots, u_i, v_0, v_1, \ldots, v_i\}$. The edges of $G_{i,0}$ are as follows: u_0 is connected to v_0, and v_0 is connected to u_k for each $k \in \{1, \ldots, i\}$. Then, for each $k \in \{1, \ldots, i\}$, u_k is connected to v_k. See Figure 1.

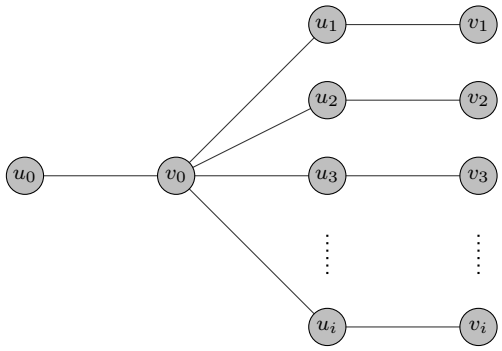

Figure 1: The graph $G_{i,0}$.

Next, for $j \in \{1, \ldots, i\}$, $G_{i,j}$ is constructed from $G_{i,0}$ by deleting edge (u_j, v_j). (Thus each such $G_{i,j}$ has two components.) Lastly, let $G_{i,i+1}$ be the intersection $\cap_{j=1}^{i} G_{i,j}$, which is the graph formed from $G_{i,0}$ in which *all* of the edges $\{(u_j, v_j)\}_{j=1}^{i}$ have been deleted (and thus $G_{i,i+1}$ has $i + 1$ components).

THEOREM 10. *Let A_{i,u_0,v_0} be a deterministic algorithm, running in the KT1 variant, that solves $GC(u_0, v_0)$ on the family of graphs $\{G_{i,j}\}_j$ for $j \in \{0, 1, \ldots, i, i + 1\}$. Then the worst-case message-complexity of A_{i,u_0,v_0} is $\Omega(n)$. In particular, there exists a $j \in \{0, i + 1\}$ such that A_{i,j_0,v_0} uses (sends) $\Omega(i) = \Omega(n)$ messages during its execution on $G_{i,j}$.*

PROOF. We partition the $n = 2i + 2$ nodes of each $G_{i,\cdot}$ as follows: For each $j \in \{1, \ldots, i\}$, let $P_{i,j}^{(1)}$ be the set of nodes $\{u_j, v_j\}$; let $P_{i,j}^{(2)} = \overline{P_{i,j}^{(1)}}$ be the complement of $P_{i,j}^{(1)}$. We say that partition $P_{i,j} = \{P_{i,j}^{(1)}, P_{i,j}^{(2)}\}$ is *crossed* during the running of an algorithm (on some $G_{i,\cdot}$) if a message is sent (i) from a node in $P_{i,j}^{(1)}$ to a node in $P_{i,j}^{(2)}$; or (ii) from a node in $P_{i,j}^{(2)}$ to a node in $P_{i,j}^{(1)}$.

We claim that each partition $P_{i,j}$ is crossed *either* during the running of A_{i,u_0,v_0} on $G_{i,0}$ *or* during the running of A_{i,u_0,v_0} on $G_{i,i+1}$. Indeed, suppose not, and let j' be one such j. Since the inputs to the $GC(u_0, v_0)$ problem for graphs $G_{i,0}$ and $G_{i,j'}$ are the same for each node of $P^{(2)}_{i,j'}$, we can first conclude that $P_{i,j'}$ is crossed during the running of A_{i,u_0,v_0} on $G_{i,j'}$. If not, then the operation of A_{i,u_0,v_0} would be identical on the nodes of $P^{(2)}_{i,j'}$ for both of the graphs $G_{i,0}$ and $G_{i,j'}$, which is impossible since u_0 must eventually decide the connectivity of the graph and send the result to v_0—and unlike $G_{i,0}$, $G_{i,j'}$ is disconnected. So a message is sent across $P_{i,j'}$ on the input graph $G_{i,j'}$. Furthermore, we can also conclude that, during the first round r' in which a message is sent across $P_{i,j'}$ (on input $G_{i,j'}$), a message is sent from a node in $P^{(1)}_{i,j'}$ to a node in $P^{(2)}_{i,j'}$. The reason for this is precisely that it is the only possibility—during round r', no node in $P^{(2)}_{i,j'}$ can send a message across $P_{i,j'}$, for otherwise that same message would be sent during the running of A_{i,u_0,v_0} on input $G_{i,0}$ (because the decision to send such a message is made prior to the receipt of any messages from $P^{(1)}_{i,j'}$).

Therefore, we conclude that during the (again, deterministic) running of A_{i,u_0,v_0} on $G_{i,j'}$, one of the nodes of $P^{(1)}_{i,j'}$ (either $u_{j'}$ or $v_{j'}$) makes the decision to send a message across $P_{i,j'}$ prior to the receipt of any messages from $P^{(2)}_{i,j'}$. The consequence of this conclusion, however, is that exactly the same behavior must occur during the running of A_{i,u_0,v_0} on $G_{i,i+1}$, because the information at $u_{j'}$ and $v_{j'}$ is identical for both of the inputs $G_{i,j'}$ and $G_{i,i+1}$. Thus a message must be sent from $P^{(1)}_{i,j'}$ to $P^{(2)}_{i,j'}$ during the running of A_{i,u_0,v_0} on $G_{i,i+1}$, which is a contradiction.

Any single message sent between two of the n nodes can cross only two of the partitions $P_{i,j}$ (because the sets $P^{(1)}_{i,j}$ are pairwise disjoint), so, in total, at least $i = \frac{n-2}{2}$ messages are sent during the runnings of A_{i,u_0,v_0} on both $G_{i,0}$ and $G_{i,i+1}$. Therefore the number of messages sent by A_{i,u_0,v_0} is at least $\frac{i}{2} = \frac{n-2}{4} = \Omega(n)$ on at least one of $G_{i,0}, G_{i,i+1}$. □

COROLLARY 11. *Let A_n be a deterministic algorithm, running in the KT1 variant of the Congested Clique model, that solves GC on inputs having n nodes. Then the worst-case message-complexity of A_n is $\Omega(n)$.*

PROOF. It is straightforward to reduce $GC(u, v)$, on an input graph G', to GC using two additional messages: (i) Run A_n on G'; (ii) At the conclusion of A_n, whichever node knows the result sends that result to u (note that *each* node of the input graph is assumed to be provided the identities of u and v as part of the input); and (iii) u sends the result to v. It follows that no deterministic algorithm for GC in the *KT1* Congested Clique model can have a worst-case message-complexity less than the smallest worst-case message-complexity for an algorithm solving a problem $GC(u, v)$ (in *KT1*) minus 2. The result follows. □

4.1.1 Lower Bound for Monte Carlo Algorithms

The above lower bound can be extended to randomized Monte Carlo algorithms via a straightforward application of Yao's Minimax Principle (Prop. 2.6 in [24]). (A lower bound of the same form for Monte Carlo algorithms has recently been derived using a different technique [27].)

COROLLARY 12. *Let A_n be a randomized algorithm for solving GC on n-node graphs, running in the KT1 variant of the Congested Clique model with error probability of at most 1/10. Then the worst-case message-complexity of A_n is $\Omega(n)$.*

PROOF. Consider the family of graphs $\{G_{i,j}\}_j$, for $0 \leq j \leq i+1$, and a "hard distribution" on this family that assigns probability 1/4 to $G_{i,0}$, 1/4 to $G_{i,i+1}$ and distributes the remaining 1/2 uniformly to the graphs $G_{i,j}$, $j = 1, 2, \ldots, i$. Consider any deterministic algorithm solving $GC(u_0, v_0)$, that errs with probability at most 1/5 over this distribution. Such an algorithm would have to be correct on $G_{i,0}$ and $G_{i,i+1}$ and correct on at least $3i/10$ of the instances $G_{i,j}$. The argument in the proof of Theorem 10 can be then repeated for $G_{i,0}$, $G_{i,i+1}$, and the $\Omega(i)$ instances $G_{i,j}$ for which the algorithm is correct. As in the deterministic case, this yields the result that at least one of the executions, on $G_{i,0}$ or $G_{i,i+1}$ uses at least $3i/40$ messages for $GC(u_0, v_0)$. This leads to an $\Omega(n)$ lower bound on message complexity of any Monte Carlo algorithm solving $GC(u_0, v_0)$ that errs with probability at most 1/10. The reduction in the proof of Corollary 11 then leads to the result for GC. □

4.2 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 *KT1* 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 [26, 2, 17].

THEOREM 13. *An MST can be computed in the KT1 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 2.1 that the Cormode-Firmani [8] 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. □

5. CONCLUSIONS

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?

6. REFERENCES

[1] M. Adler, W. Dittrich, B. H. H. Juurlink, M. Kutylowski, and I. Rieping. Communication-optimal parallel minimum spanning tree algorithms. In *Proceedings of the 10th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 27–36, 1998.

[2] K. J. Ahn, S. Guha, and A. McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the 23rd annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 459–467, 2012.

[3] K. J. Ahn, S. Guha, and A. McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st Symposium on Principles of Database Systems (PODS)*, pages 5–14, 2012.

[4] N. Alon, R. Rubinfeld, S. Vardi, and N. Xie. Space-efficient local computation algorithms. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1132–1139. SIAM, 2012.

[5] B. Awerbuch, O. Goldreich, R. Vainish, and D. Peleg. A trade-off between information and communication in broadcast protocols. *J. ACM*, 37(2):238–256, 1990.

[6] A. Berns, J. Hegeman, and S. 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] K. Censor-Hillel, P. Kaski, J. H. Korhonen, C. Lenzen, A. Paz, and J. Suomela. Algebraic methods in the congested clique. *arXiv preprint, arXiv:1503.04963*, 2015.

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

[9] D. Dolev, C. Lenzen, and S. 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.

[10] A. Drucker, F. Kuhn, and R. 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.

[11] A. Drucker, F. Kuhn, and R. 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.

[12] J. Gehweiler, C. Lammersen, and C. 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.

[13] J. W. Hegeman and S. 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] J. W. Hegeman, S. V. Pemmaraju, and V. 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] S. Holzer and N. Pinsker. Approximation of Distances and Shortest Paths in the Broadcast Congest Clique. *arXiv preprint, arXiv:1412.3445*, 2014.

[16] H. Jowhari, M. Sağlam, and G. Tardos. Tight bounds for L_p samplers, finding duplicates in streams, and related problems. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 49–58, 2011.

[17] B. M. Kapron, V. King, and B. 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.

[18] D. R. Karger, P. N. Klein, and R. E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *J. ACM*, 42(2):321–328, Mar. 1995.

[19] H. Klauck, D. Nanongkai, G. Pandurangan, and P. 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] E. Korach, S. Moran, and S. 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.

[21] C. 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, 2013.

[22] Z. Lotker, B. Patt-Shamir, E. Pavlov, and D. Peleg. Minimum-weight spanning tree construction in $O(\log\log n)$ communication rounds. *SIAM Journal on Computing*, 35(1):120–131, 2005.

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

[24] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995.

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

[26] G. Pandurangan, P. Robinson, and M. Scquizzato. Almost optimal distributed algorithms for large-scale graph problems. *arXiv preprint, arXiv:1503.02353*, 2015.

[27] G. Pandurangan and M. Scquizzato. Message lower bounds in synchronous distributed models via communication complexity. *Manuscript*, 2015.

[28] B. Patt-Shamir and M. 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.

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

[30] D. P. Woodruff and Q. Zhang. When distributed computation is communication expensive. *Distrib. Comput.*, to appear.