MST in Log-Star Rounds of Congested Clique

Mohsen Ghaffari
MIT
ghaffari@mit.edu

Merav Parter
MIT
parter@mit.edu

ABSTRACT

We present a randomized algorithm that computes a Minimum Spanning Tree (MST) in $O(\log^* n)$ rounds, with high probability, in the *Congested Clique* model of distributed computing. In this model, the input is a graph on n nodes, initially each node knows only its incident edges, and per round each two nodes can exchange $O(\log n)$ bits.

Our key technical novelty is an $O(\log^* n)$ *Graph Connectivity* algorithm, the heart of which is a (recursive) forest growth method, based on a combination of two ideas: a *sparsity-sensitive sketching* aimed at sparse graphs and a *random edge sampling* aimed at dense graphs.

Our result improves significantly over the $O(\log \log \log n)$ algorithm of Hegeman et al. [PODC 2015] and the $O(\log \log n)$ algorithm of Lotker et al. [SPAA 2003; SICOMP 2005].

1. INTRODUCTION AND RELATED WORK

Congested Clique is a basic model of distributed computing that was introduced by Lotker, Patt-Shamir, Pavlov, and Peleg [LPPSP03]. Since its introduction, this model has received extensive attention and it is receiving increasingly more, recently [PST11,DLP12,BHP12,Len13a,DKO14,Nan14, HPS14, HP14, CHKK+15, HPP+15, BFARR15, Gha16]. In this model, initially each of the n nodes knows only its incident edges in the graph $G = (V, E)$, communications happen in synchronous rounds, and per round, each two nodes exchange $O(\log n)$ bits. Note that unlike the classical CONGEST model [Pel00], here even non-adjacent nodes of G can communicate. While at first glance this seems unorthodox, there are at least two strong motives for studying this *complete communication* model:

The Practical Motivation, *Overlay Networks* and *Large-Scale Graph Processing*: The traditional viewpoint in distributed network algorithms was that processors running the algorithm reside in network nodes and two processors can communicate only if their nodes are adjacent. This implicitly assumes that the whole network, or a connected subgraph of it, runs the algorithm. In many practical cases, this would not be realistic, e.g., most high-level protocols running on the Internet do not control such a subgraph. A more common scenario is that of *overlay networks*, where a number of processors spread over the whole network are running the distributed algorithm. These processors are not necessarily adjacent in the base network but are instead connected via an overlay that provides point-to-point communications between any two of them.

Congested Clique also relates to two models of processing large-scale graphs, the k-machine model [KNPR15] and the MapReduce model [DG08, KSV10, HP15].

The Theoretical Motivation, *Taking Locality Out of the Picture*: Two central challenges in distributed computing are *congestion* and *locality* [Pel00]. Studying these challenges together, as is necessary in the CONGEST model, creates a complex mix. From a theoretician's perspective, a natural step towards understanding these two issues is to decouple them, i.e., to first understand each of them separately. The LOCAL model [Lin92,Pel00], which is by now quite well-studied, is a perfect model in taking the congestion out of the picture and focusing on *locality*. The Congested Clique model can be viewed as the complement, which instead takes locality out and focuses on *congestion*. For instance, the usual locality-based $\Omega(D)$ lower bound of the CONGEST model—where D denotes the network diameter—disappears in the Congested Clique model.

MST in Congested Clique: At the center of the study of the Congested Clique model is the problem of computing a *Minimum Spanning Tree* (MST); in fact the model started with this problem [LPPSP03]. It is easy to see that the 1926 approach of Boruvka [NMN01] can be adapted to solve MST in $O(\log n)$ rounds of this model. While introducing the model, the main result of Lotker et al. [LPPSP03] was to show that MST can be solved faster, particularly in $O(\log \log n)$ rounds. Progress was scarce afterward, for more than a decade, until recently when Hegeman et al. [HPP+15] presented an $O(\log \log \log n)$ algorithm. Considering these gradually shrinking bounds, there has been a growing anticipation in the community that the optimal bound should be much smaller. Though, the question of improving the bound remained *"tantalizing"* [HPP+15] and open.

Our Result: In this paper, we show that MST can indeed be solved significantly faster:

This work was partially funded by the following grants: AFOSR FA9550-13-1-0042, NSF CCF-1461559, CCF-1217506, CCF-0939370.

PODC'16, July 25-28, 2016, Chicago, IL, USA

DOI: http://dx.doi.org/10.1145/2933057.2933103

Theorem 1.1. *There is a randomized algorithm in the Congested Clique model that computes a minimum spanning tree in $O(\log^* n)$ rounds, with high probability.*

The time complexity of our algorithm improves gracefully when larger messages are allowed. If the message size is $\Theta(b \log n)$, then the round complexity becomes $O(\log^* n - \log^* b)$. Thus, the round complexity is $O(1)$ rounds for $b = \log\log\ldots\log n$ for c time iterated logarithm, for any constant $c \geq 1$. Contrasting this with a result of Hegeman et al. [HPP+15], it can be phrased as improving the message size of $O(1)$-round algorithms from $O(\log^4 n)$ to almost $O(\log n)$. Our connectivity algorithm also leads to improvements in several other problems, using standard connections:

Corollary 1.2. *In the congested clique model, (1) there are randomized algorithms that solve the following verification problems in $O(\log^* n)$ rounds with high probability*[1]*: Bipartiteness verification, cut verification, s-t connectivity and cycle containment. (2) There is an $O(\log \Delta + \log^* n)$ round randomized MIS algorithm.*

1.1 Our Approach, in a Nutshell

The main technical novelty in our result is an $O(\log^* n)$ *Graph Connectivity* algorithm. The extension from this to MST is based on a mostly known method and will be described in Section 3.

The heart of our connectivity algorithm is growing a maximal forest. To explain this forest growth, suppose that the graph is connected; the method generalizes to disconnected graphs as we describe later. The key in our forest growth is an $O(1)$-round algorithm with the following guarantee:

Forest Growth: *The forest grows from $O(\frac{n}{\log^2 x})$ to $\frac{n}{x}$ components, in $O(1)$ rounds, w.h.p.*

To get a maximal forest, we apply this forest growth for $O(\log^* n)$ iterations, where in the i^{th} iteration we set $x = 2 \uparrow\uparrow \Theta(i)$. Here, $2 \uparrow\uparrow k$ denotes a power tower with base 2 and height k [Knu76].

The Algorithm's Intuition: To explain the intuition behind this $O(1)$-round forest growth algorithm, we next discuss two opposite extremes of the scenario we can face: a *dense* and a *sparse* scenario. In reality, we will not be in either of these extremes *per se*. Nonetheless, the algorithm is an interpolation of the two solutions and we will prove that it always enjoys the fast growth. (I) Dense Scenario: Suppose that each of the current components is incident on at least x^6 outgoing edges. We make each component select a uniformly random outgoing edge, to be added to the forest. One can prove that due to the high density, once these randomly selected edges are added to the forest, w.h.p., the number of connected components is at most n/x. Roughly speaking, the reason is that each new component will (be expected to) have at least x nodes.

(II) Sparse Scenario: Suppose that each of the current components is incident on at most x^6 outgoing edges, and assume that this remains true even as we merge components. For this case, our solution makes use of a simple linear sketching that we designed for this particular case. We note that this is in part inspired by the work of Hegeman et al. [HPP+15] whose approach was, in a nutshell, as follows: grow forest components using Lotker et al's [LPPSP03] algorithm till there are $O(n/\operatorname{poly}(\log n))$ components, and then use $O(\operatorname{poly}(\log n))$-bit ℓ_0-sampler sketches, following Ahn et al. [AGM12], to finish the problem in $O(1)$ extra rounds[2].

In our solution, a key point is to design a *sparsity-sensitive linear sketching* which allows us to reduce the sketch size when the graph is sparse. Particularly, considering that each component is assumed to have at most x^6 outgoing edges, we compute $L = \Theta(\log x)$ many special *linear sketches* in each component, where each sketch has size $O(\log x \log n)$ bits. We discuss these sketches shortly. Then, we gather all these sketches—which is a total of $O(\frac{n}{\log^2 x}) \cdot O(\log x) \cdot O(\log x \log n) = O(n \log n)$ bits—in a leader node, in $O(1)$ rounds. The leader then simulates $\Theta(\log x)$ iterations of Boruvka's component-merging algorithm locally using these sketches. At the end of these locally-computed merges, the forest has at most n/x components.

Our sketches are very simple. Each sketch is composed of $10 \log x$ rows, and each row has $\Theta(\log n)$ bits. To compute the i^{th} row, we do as follows: mark each edge with probability $1/2^i$. Then, the i^{th} row in the sketch of each node v is the bit-wise XOR of the identifiers[3] of the marked edges incident on v. The sketch of a component is defined to be the bit-wise XOR of the sketches of its nodes.

The intuition for this sketch design is as follows: We argue that for each component C, with constant probability, a sketch of C provides an edge going out of C. Notice that the edges internal to C do not affect the sketch of C, because even if such an edge is marked, it gets added twice to the XOR, once from each endpoint, and thus gets canceled out. Let k be the number of edges with exactly one endpoint in C. Note that due to the sparsity assumption, we have $k \in [1, x^6]$. In the i^{th} row of the sketch of C, where $i = \lceil \log k \rceil$, the probability that exactly one outgoing edge of C is marked for this row is at least a constant. If that happens, the i^{th} row is simply the identifier of one outgoing edge, i.e., the sketch of C provides an edge going out of C. Now this happens for each component with constant probability. Thus, if the leader adds all the outgoing edges extracted from the first sketch of all components, w.h.p., the number of components decreases by a constant factor.

The leader can continue this process locally for all L sketches. When two components merge, the sketch of the resulting merged component is simply the XOR of their sketches. Hence, the leader can locally compute the new component sketches and continue. Repeating this for L iterations, once for each sketch, reduces the number of connected components to $n/2^{\Theta(L)} = n/x$.

2. CONNECTIVITY IN $O(\log^* N)$ ROUNDS

In this section, we describe our graph connectivity algorithm, which leads to the following result:

[1] As standard, the term *high probability* indicates a probability exceeding $1 - 1/n^c$, for an arbitrary constant $c \geq 2$.

[2] Sketching based solutions for the connectivity problems were also employed for the centralized dynamic setting [KKM13] and in the k-machine model [PRS15].

[3] We will assign certain $\Theta(\log n)$-bit random identifiers to the edges. These ids will let us determine w.h.p. if a row is exactly only one identifier, or the XOR of two or more identifiers. This will be discussed in Section 2.2.3.

Theorem 2.1. *There exists a randomized algorithm in the congest clique model that identifies the connected components in the graph G in $O(\log^* n)$ communication rounds, with high probability*[4].

This algorithm computes a *maximal forest* F of G, by starting with the trivial forest $F = (V, \emptyset)$ with no edges, and gradually adding edges of G to F—which thus merges some connected components of F with each other—until F is maximal. For the sake of analysis, we call each connected component C of F *growable* if it is not equal to the component of G that contains C, i.e., if C has at least one outgoing edge to vertices in other components and hence it can still grow. In the algorithm, each component C of F is either *active* or *inactive*, and the algorithm ensures that a component becomes inactive only if it is *not* growable anymore.

The key in our method is an algorithm called $\mathsf{ReduceCC}(x)$, which reduces the number of active connected components of F from $O(n/\log^2 x)$ to n/x, in $O(1)$ rounds. The connectivity algorithm is simply made of $O(\log^* n)$ calls to $\mathsf{ReduceCC}(x)$, where in the i^{th} call we set $x = 2 \uparrow\uparrow \Theta(i)$. In the sequel, we describe and analyze the Algorithm $\mathsf{ReduceCC}(x)$.

2.1 The Algorithm's Outline

Our approach for growing the forest is, to some extent, in the spirit of Boruvka's 1926 algorithm. Let us briefly recall Boruvka's algorithm. This algorithm works in $O(\log n)$ iterations, where in each iteration, from each growable component, we pick an outgoing edge. We then add all these outgoing edges to the forest, while ignoring cycle-creating edges. Each such iteration reduces the number of growable components by a 2 factor. Hence, after $O(\log n)$ iterations, we reach to a maximal forest.

As mentioned above, to reduce the number of active components from n to 0 in $O(\log^* n)$ rounds, our task boils down to reducing the number of active components from $O(n/\log^2 x)$ to n/x within $O(1)$ rounds. To achieve this, our approach has two main forest growth steps, as well as a simple clean up step. First, we reduce the number of active components whose number of outgoing edges is at most x^6. We call these components *low-degree* components. Then, the second forest growth step is aimed mainly at *high-degree* components, those (new components) with at least x^6 outgoing edges. After this, we perform a clean up step that deactivates some components that are not growable. At the end, we get that the number of active connected components decreases to at most n/x. We next briefly explain these three steps. A more detailed description follows.

To reduce the number of growable low-degree components to at most n/x, we would like to locally simulate $\Theta(\log x)$ iterations of Boruvka's algorithm at a given leader node u^*. The main challenge in doing so is that *a priori* a node v cannot predict the sequence of the components to which it might belong throughout the process of component merging. Thus v does not know which of its edges will be outgoing for these components. To overcome this, we will design and use a special *linear sketching* procedure that handles all components with at most x^6 outgoing edges.

Our key observation is as follows: an outgoing edge of a *low-degree* component can be computed by letting each node sketch the identifiers of its incident edges into $O(\log x \log n)$ bits. Since there are $O(n/\log^2 x)$ active components, we in fact have sufficient capacity to send $\Theta(\log x)$ such sketches per component to the leader u^*; this will be proven formally. The leader then will use these sketches to locally simulate $\Theta(\log x)$ iterations of Boruvka's algorithm. We will show that the number of growable low-degree components at the end of this step, in the output decomposition $\mathcal{C}'$, is at most $n/(4x)$.

Next, we focus on high-degree components in $\mathcal{C}'$, where we use the following simple yet powerful observation: a single randomly sampled edge of a high-degree component of cardinality at most $(8x)$ is with a "good" probability an outgoing edge. Thus, such an edge is likely to reduce the number of connected components. We will show that by letting each node send one random edge to the leader, and the leader adding them to the forest (while ignoring cycles), the number of high-degree components gets reduced, such that we have at most $n/(2x)$ growable components.

The third step then identifies the non-growable components that have less than $(8x)$ nodes and deactivates them. Note that there are at $n/(8x)$ non-growable components with more than $8x$ nodes. Hence, at the end of this clean up step, what remains will be at most n/x active components. We next provide a compact and high level description of the algorithm, after which we discuss each of the steps in detail, each in one of the subsequent subsections.

Algorithm $\mathsf{ReduceCC}(x)$

Input: a forest F with a set $\mathcal{C}$ of at most $O(n/\log^2 x)$ active components.
Output: a forest F' with a set $\mathcal{C}_{out}$ of at most n/x active components.

- **(S1): Handling low-degree components**
  - Each node v computes $L = \Theta(\log x)$ many linear sketches $\mathsf{Sketch}_1(v)$, $\mathsf{Sketch}_2(v)$, ..., $\mathsf{Sketch}_L(v)$, where each of these sketches has $O(\log x \log n)$ bits.
  - For every active connected component $C \in \mathcal{C}$, let $\mathsf{Sketch}_i(C) = \sum_{v \in C} \mathsf{Sketch}_i(v)$.
  - Route the sketches $\mathsf{Sketch}_i(C)$ of all active components $C \in \mathcal{C}$ to the leader node u^*.
  - Let u^* locally simulates $\Theta(\log x)$ rounds of the basic connectivity algorithm, resulting in the new component decomposition $\mathcal{C}' = \{C'_1, \ldots, C'_k\}$.
- **(S2): Handling high-degree components**
  - Every node sends a random edge to the leader.
  - The leader u^* merges components using these edges, resulting in the output decomposition $\mathcal{C}_{out} = \{C''_1, \ldots, C''_{k'}\}$.
- **(S3): Clean Up**
  - The leader identifies and deactivates components of $\mathcal{C}_{out}$ that are not *growable* and are *small*—having less than $8x$ nodes.

[4]We remark that, by slight adjustments, one can make the message complexity of this algorithm be $O(n \,\mathrm{poly}(\log n))$. Details are deferred to the full version.

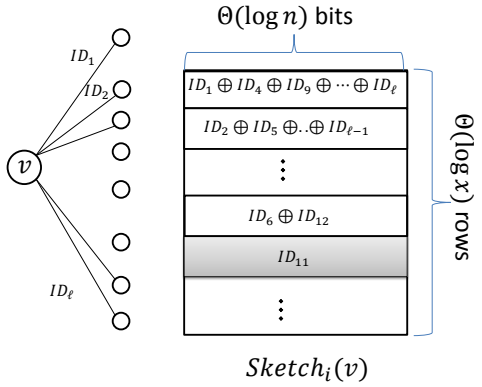

Figure 1: An illustration of a single copy $\mathsf{Sketch}_i(v)$ of the sketch for vertex v. The degree of vertex v is $\ell \leq x$. The sketch contains $\Theta(\log x)$ rows, each contains $\Theta(\log n)$ bits. For each row j, the edges are sampled with probability 2^{-j}, and the row contains the bitwise XOR of these sampled edges. The row $\ell' = \lceil \log \ell \rceil$ contains the ID of a single edge with constant probability.

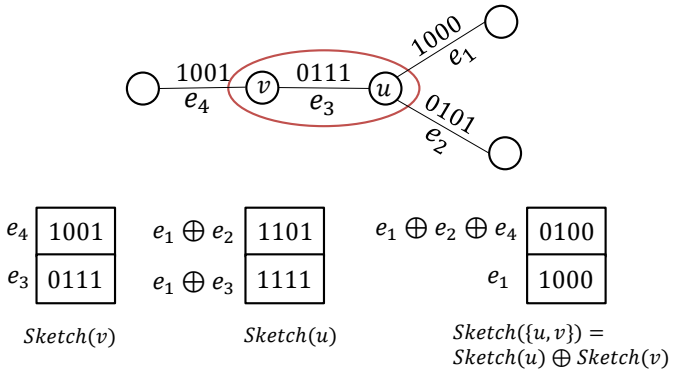

Figure 2: A simple illustration of the sketches and the effect after a merge.

2.2 Step (S1): Handling the Low-Degree Components

2.2.1 The Algorithm of Step (S1)

Let $\mathcal{I} = \{\mathrm{ID}(e_1), \ldots, \mathrm{ID}(e_m)\}$ be a set of random edge identifiers, each having $\Theta(\log n)$ bits. The construction of these edge identifiers is described in Section 2.2.3. For a subset of edges $S \subseteq E$, let $\mathsf{XOR}(S)$ be the bitwise XOR of the IDs of edges in S, i.e., $\mathsf{XOR}(S) = \oplus_{e \in S} \mathrm{ID}(e)$.

Sketch Description: Each sketch of node v is composed of $10 \log x$ rows, and each row has $\Theta(\log n)$ bits. Let $E_j \subseteq E$ be a subset of the graph edges, where each $e \in E$ is included in E_j independently with probability 2^{-j}. Let $E_j(v)$ be the subset of E_j edges that are incident on v. The j^{th} row of the sketch of v is the bit-wise XOR of the edge identifiers in $E_j(v)$. The node v computes $\Theta(\log x)$ independent copies of the sketch, where each $\mathsf{Sketch}_i(v)$ is a matrix consists of $10 \log x$ rows as follows

$$\mathsf{Sketch}_i(v) = [\mathsf{XOR}(E_1(v)), \ldots, \mathsf{XOR}(E_{10 \log x}(v))]. \quad (1)$$

See Fig. 1 for an illustration.

We remark that we will ensure that for each edge $e = (u, v)$, nodes u and v know consistently whether $e \in E_i$ or not. Thus, $e = (u, v)$ either appears in both $E_i(v)$ and $E_i(u)$ or in neither. This property is critical, cf. Observation 2.2. In Section 2.2.3, we explain how to achieve this property for all sketches using merely $O(\log n)$ bits of communication between u and v.

Local simulation of Boruvka's algorithm: Let $\mathcal{C} = \{C_1, \ldots, C_k\}$ be the initial collection of $k = O(n/\log^2 x)$ active connected components. For each i, the i^{th} sketch of a component C is defined to be the bit-wise XOR of the i^{th} sketches of its nodes. In Lemma 2.6, we explain the routing that in $O(1)$ rounds, delivers to the leader $\Theta(\log x)$ sketches $\mathsf{Sketch}_i(C)$ per each component C in $\mathcal{C}$.

Each of the $\Theta(\log x)$ sketches of a given component would be used to obtain *one* outgoing edge. The i^{th} sketches $\mathsf{Sketch}_i(C)$ are used in the i^{th} iteration of simulation of Boruvka's algorithm, where $\mathsf{Sketch}_i(C)$ will provide (at most) one edge going out of C. By Lemma 2.4, the random identifiers are constructed in a way that allows the leader to determine w.h.p. whether a row has exactly only one identifier or it is a XOR of two or more identifiers. The leader then looks for a row (among the $10 \log x$ rows) in $\mathsf{Sketch}_i(C)$ that contains one sampled edge. If such an outgoing edge is found, then this edge is added to the forest and the components of the corresponding two endpoint nodes are merged into a single component. For each $j \geq i$, the j^{th} sketch of the new component is simply the bitwise XOR of the two components' j^{th} sketches. At the end of these $\Theta(\log x)$ merge iterations of Boruvka's algorithm, let $\mathcal{C}'$ be the resulting intermediate component decomposition.

2.2.2 Analysis of Step (S1)

The following simple observation is useful in this analysis.

Observation 2.2. *Let C be a connected component and let E_{out} be the set of edges with exactly one endpoint in C. Then for each $i \in [1, 10 \log x]$, the i^{th} row of $\mathsf{Sketch}(C)$ is simply the XOR of a subset S of edges E_{out}, where each edge $e \in E_{out}$ appears in S independently with probability $1/2^i$. Particularly, the set of edges with zero or with two endpoints in C do not affect this sketch.*

We say that a component is *large* if it has at least $8x$ nodes, and *small* otherwise. Also recall that a component is called *high-degree* if the number of edges going out of it is at least x^6, and *low-degree* otherwise. Let $\mathcal{C}'$ be the component decomposition at the end of step (S1).

Lemma 2.3. *W.h.p., the number of growable low-degree components in $\mathcal{C}'$ is at most $n/(4x)$.*

Proof. Given any component decomposition, let us call a component (and thus also all of its nodes) *blue* if its outgoing degree is at least $2x^6$, or if it contains a node/component that was colored blue in the past component decompositions. Otherwise, the component is called *red*.

We first argue that the total number of blue low-degree components in $\mathcal{C}'$ is at most $n/(8x)$. Consider a blue component $C' \in \mathcal{C}'$. There are two cases. Case 1: either C' is large, in which case there are at most $n/(8x)$ such components. Case 2: C' is small, in which case since C' contains a blue node, its outgoing degree is at least $2x^6 - (8x)^2 \geq x^6$ and hence C' is not low-degree. Therefore, the total number of blue low-degree components in $\mathcal{C}'$ is at most $n/(8x)$.

Clearly, this also means the number of such components that are also growable is at most $n/(8x)$.

To complete the proof, we next show that, w.h.p., the number of the growable red low-degree components of $\mathcal{C}'$ is at most $n/(8x)$. To prove this, we consider $\Theta(\log x)$ iterations of Boruvka's algorithm as simulated locally by the leader u^*. For each iteration i, let $\mathcal{C}_i$ be the component decomposition at the beginning of this iteration and let y_i be the number of red growable components in $\mathcal{C}_i$. The argument will be provided in two similar parts, first when $\log x \leq O(\log n)$ and second when $\log x \geq \Omega(\log n)$. In the first case, we argue that w.h.p. $y_{i+1} \leq 39y_i/40$. Thus, we would get that $y_k \leq n/(8x)$ for $k = c \log x$ for large enough constants c. In the second case, we argue that $\mathbb{E}[y_{i+1}] \leq 39y_i/40$, and then use the large value of x to show that after $k = c \log x$ merges, w.h.p., $y_k \leq n/(8x)$.

Let us focus on the case where $\log x \leq O(\log n)$ and consider the i^{th} iteration; here we want to prove that w.h.p. $y_{i+1} \leq 39y_i/40$. Let us first examine the expected number of outgoings edges that are decoded successfully using the i^{th} sketch. Consider a red growable component C in $\mathcal{C}_i$ and let t be the number of its outgoing edges $E_{OUT}(C)$. Note that since this is growable component and since it is red, we have $t \in [1, 2x^6]$, which implies $\ell = \lceil \log t \rceil \in [1, 10 \log x]$. Consider the ℓ^{th} row of the sketch $\mathsf{Sketch}_i(C)$. Recall that this row is the bitwise XOR of a sampling of edges of $E_{OUT}(C)$, where each edge is sampled independently with probability $2^{-\ell}$. Therefore, the probability that this sampling contains exactly one edge is at least $\frac{t}{2^\ell}(1-1/2^\ell)^{t-1} \geq 1/10$. That is, with probability at least $1/10$, this row provides the identifier of one edge going out of C.

From above, we can conclude that in expectation, the number of outgoing edges that were successfully sampled from the sketches of growable components with only red nodes is at least $y_i/10$. Since $y_i \geq \log n$, by Chernoff bound, we get that w.h.p. the number of outgoing edges is at least $y_i/20$. Therefore, once we add these edges to the forest, the number of red growable components in $\mathcal{C}_{i+1}$ is at most $39y_i/40$. Notice that it is crucial in this argument that no new red component gets introduced, meaning that each component that is red in $\mathcal{C}_{i+1}$ is made solely of red components of $\mathcal{C}_i$, which holds because of the way we defined colors blue and red. Therefore, we have that w.h.p. $y_{i+1} \leq 39/40 \cdot y_i$, which completes the proof for the $\log x \leq O(\log n)$ case.

Let us consider the case where $\log x \geq \Omega(\log n)$. Using arguments as above, we get that $\mathbb{E}[y_{i+1}] \leq 39y_i/40$. Invoking this expectation for $k = c \log x$ iterations, and noting that $\log x \geq \Omega(\log n)$, we get that $\mathbb{E}[y_k] \leq 1/n^5$. Hence, by Markov's inequality, the probability that $y_k \geq 1$ is at most $1/n^5$. That is, w.h.p, $y_k = 0$. This completes the proof. □

2.2.3 Smaller Algorithmic Details of Step (S1)

Assigning random edge identifiers: Our edge identifier construction is based on the notion of *ϵ-bias* sets [NN93]. The construction is randomized and guarantees that, w.h.p., the XOR of each given subset of edges $S \subseteq E$, for $|S| \geq 2$, is not a legal edge identifier (i.e., is not equal to correspond to an ID of an edge that exists). The construction of the edge IDs is randomized and uses a seed of only $O(\log n)$ random bits, which can be chosen by the leader and delivered to all nodes of the network in $O(1)$ rounds. This allows all nodes to know the their corresponding edge IDs. Furthermore, the leader will be able to detect whether a given $O(\log n)$ bits corresponding to an $\mathsf{XOR}(S)$ is a legal ID (in which case, S contains a single edge) or not, w.h.p.

Lemma 2.4. *There is an algorithm that creates a collection $\mathcal{I} = \{\mathrm{ID}(e_1), \ldots, \mathrm{ID}(e_M)\}$ of $M = \binom{n}{2}$ random identifiers for edges, each of $O(\log n)$-bits, and where each node learns the identifiers of its incident edges, within $O(1)$ rounds. These identifiers are such that for each subset $E' \subseteq E$, where $|E'| \neq 1$, we have $\Pr[\mathsf{XOR}(E') \in \mathcal{I}] \leq 1/n^{10}$.*

Proof. Let $\ell = O(\log n)$ be the length of the edge identifier. To compute $\mathcal{I}$, we use an *ϵ-bias* set $S_j \in \{0,1\}^M$ for each of the $j \in [\ell]$ coordinates of the edge identifiers. That is, for each $j \in [\ell]$, the set $S_j = [b_1, \ldots, b_M]$ specifies the j^{th} bit for the identifier of each of the possible $M = \binom{n}{2}$ graph edges. Next, we first define the ϵ-bias property of these sets, and then explain how they can be constructed using $O(\log^2 n)$ bits of communication. Finally, we argue why the ϵ-bias property guarantees that $\Pr[\mathsf{XOR}(S) \in \mathcal{I}] \leq 1/n^{10}$ for each subset $S \subseteq E$ such that $|S| \neq 1$.

The randomly chosen set $S_j = [b_1, \ldots, b_M]$ satisfies the *ϵ-biased property* if for each set of indices $I \subseteq \{1, \ldots, M\}$, it holds that

$$\Pr[\oplus_{i \in I} b_i = 1] \in [1/2 \pm \epsilon] \ . \tag{2}$$

For our proposes, it is sufficient to set $\epsilon = 1/10$. Naor and Naor [NN93] provide a deterministic construction of an ϵ-bias space with size $O(M/\operatorname{poly}(\epsilon))$. That is, they particularly provide a collection $\mathcal{F}$ of functions, where $|F| = O(M/\operatorname{poly}(\epsilon))$, where each $f \in \mathcal{F}$ is a function mapping $[M]$ to $\{0,1\}$, and if we pick a random function $f \in \mathcal{F}$, we would have $\Pr[\oplus_{i \in I} f(i) = 1] \in [1/2 \pm \epsilon]$ for all nonempty subsets I. The randomness in this statement is in the choice of $f \in \mathcal{F}$. We define the set S_j in our identifier construction by picking simply one random function f_j from $\mathcal{F}$. Note that choosing one such random function requires $\log |\mathcal{F}| = \Theta(\log M) = \Theta(\log n)$ bits.

Since the construction of $\mathcal{F}$ is deterministic, all vertices can compute locally the family function $\mathcal{F}$ and the only thing that needs to be communicated between the nodes is the random bits used for selecting the ℓ random functions f_j in $\mathcal{F}$. Since electing a random function in $\mathcal{F}$ requires $\Theta(\log |\mathcal{F}|)$ bits, overall the total number of bits required for specifying these $\ell = \Theta(\log n)$ functions is $\Theta(\log^2 n)$ bits. Clearly, this is achievable in $O(1)$ rounds in the Congested Clique, e.g., by letting a leader node select the $\Theta(\log^2 n)$ bits using private randomness, and then communicating them to all nodes via the help of $O(\log n)$ relay nodes. This completes the description of the construction of $\mathcal{I}$.

Now, we argue that this construction satisfies the property we desire from these random edge identifiers. Consider a subset $E' \subset E$ such that $|E'| \neq 1$. We bound the probability that $\mathsf{XOR}(E') = \mathrm{ID}(e)$. To do that, we distinguish between two cases. First, assume that $e \in E'$. Then if $\mathsf{XOR}(E') = \mathrm{ID}(e)$, it holds that $\mathsf{XOR}(T) = [0, 0, ..., 0]$ where $T = E' \setminus \{e\} \neq \emptyset$. Hence, by Equation (2), we get that $\Pr[\mathsf{XOR}(E') = \mathrm{ID}(e)] = \Pr[\mathsf{XOR}(T) = [0, 0, ..., 0]] \leq 0.4^{\log n} = 1/n^c$. Next, consider the complementary case where $e \notin E'$. In this case, $\mathsf{XOR}(E') = \mathrm{ID}(e)$ implies that $\mathsf{XOR}(T) = [0, 0, ..., 0]$ where $T = E' \cup \{e\} \neq \emptyset$ and again, by Equation (2), this happens with probability at most $1/n^c$. By union bounding over all m edges, we get that $\Pr[\mathsf{XOR}(E') \in \mathcal{I}] \leq 1/n^{10}$. □

Consistent edge sampling for the sketches: Our goal is that for each edge $e = (u, v)$, nodes u and v will know consistently whether the edge e is sampled or not in the sketch. To do that, only the endpoint of the larger id, let it be u, makes the random choices regarding whether the edge is sampled. The node u then sends to v on their edge (u, v) a description concerning all the indices in which the edge (u, v) has been sampled in all of its sketches[5]. These indices will be described in the message by putting a special separation mark between two indices of the same sketch and a different mark between indices of different sketches. We now show that this message contains $O(\log n)$ bits:

Lemma 2.5. *The message describing the indices of successful samplings has $O(\log n)$ bits, w.h.p.*

Proof. Consider a edge $e = (u, v)$ and without loss of generality let u be the node with the larger identifier. Note that overall the iterations of our connectivity algorithm, the number of sketches computed by each node u is $O(\log n) + O(\log \log n) + \ldots + O(1) = O(\log n)$. We now show that indices of these samplings in which e is sampled can be written, w.h.p., in $O(\log n)$ bits. That is, the message containing the sampled indices for *all* these $O(\log n)$ sketches has at most $O(\log n)$ bits, w.h.p.

Consider the i^{th} sketch of u and let r_i be the random variable indicating the sum of indices of the rows for which (u, v) has been sampled. Note that the sampled indices can be described using $O(r_i)$ bits, that is, using a number of bits is linear in the summation of the indices of the sampled rows. We first argue that the probability distribution of this random variable r_i is upper bounded by that of a geometric random variable. Particularly, we upper bound the probability that $r_i = y$ for each integer $y \geq 1$ by a function exponentially decaying in y. Then, we invoke a concentration of summation of geometrically distributed random variables to conclude the proof.

Note that if $r_i = y$, then the sum of the indices of the sampled rows is y and the probability that the edge (u, v) has been sampled into these particular rows is at least 2^{-y}. This is regardless of the particular set of indices whose sum is y. To see this, consider an arbitrary possible set of row indices $i_1, \ldots, i_\ell$ satisfying $\sum_j^\ell i_j = y$. Then, the probability that e has been sampled to these rows is at most $\prod_j^\ell 2^{-i_j} = 2^{\sum_j^\ell i_j} = 2^{-y}$. To bound the probability that $r_i = y$, we then union bound over all distinct ways of representing y as a sum of natural numbers. By a *partition number*[6] result of Erdős [Erd42], there are at most $2^{\Theta(\sqrt{y})}$ such options. Since the probability that edge e is sampled for each such options (i.e, that e was sampled to these exact rows) is bounded by 2^{-y}, overall $\Pr[r_i = y] \leq 2^{\Theta(\sqrt{y})} \cdot 2^{-y} \leq 2^{-y/2}$.

Let z_i be a random variable with a geometric distribution with parameter $1/2$, i.e., where $\Pr[z_i = y] = 2^{-y}$ for $y \in \mathbb{N}$. We have $\Pr[r_i = y] \leq \Pr[z_i = y/2]$. Let $Z = \sum_{i=1}^k z_i$ and $R = \sum_{i=1}^k r_i$ where $k = O(\log n)$ is the total number of sketches computed by u throughout the algorithm. Hence, $\Pr[R \geq 20k] \leq \Pr[Z \geq 10k]$. It is known that the summation of geometric random variables is distributed according to a negative binomial distribution and it is concentrated around its expectation. More precisely, $\Pr[Z \geq 20k] \leq exp(-k) = 1/n^5$. Therefore, $\Pr[R \geq 40 \log n] \leq 1/n^5$, thus completing the proof. □

[5]Note that the sketch information is independent of the current component decomposition and hence the node can compute in the beginning of the algorithm all the sketches and send to its neighbor this consistency information.

[6]The *partition number* of an integer y is the number of ways to represent y as a summation of positive integers.

Lemma 2.6. *Consider the setting where a leader node u^* knows a forest with $O(n/\log^2 x)$ active connected components $\mathcal{C} = \{C_1, \ldots, C_k\}$. There is an $O(1)$-round routing scheme that delivers the sketches of all these components to the leader u^*.*

Proof. Recall that every node has $\Theta(\log x)$ sketches, each of size $O(\log x \log n)$ bits, and that each component sketch is simply the bit-wise XOR of the corresponding node sketches. We next describe how to deliver these component sketches to the leader in $O(1)$ rounds. The leader u^* partitions the node-set of each active component C_i into $\lceil |C_i|/(n/k) \rceil$ subsets, each of size at most n/k. This is a total of at most $2k \ll n$ subsets. The leader u^* then elects a distinct relay node w_i for each subset, and sends the id of w_i to all the nodes in that subset.

Next, we deliver the component sketches from the nodes to the leader in a two-step manner, using the help of these relay nodes. First, all nodes send their sketches to their corresponding relay nodes. Then, each relay node computes the bitwise XOR of its received sketches and forwards the result to the leader u^*, who can readily compute the component sketches. We next argue that each of these steps can be performed in $O(1)$ rounds, using Lenzen's routing [Len13b].

Lenzen [Len13b] provides a routing scheme such that if each node is the source for $O(n)$ messages and each node is the the designation for $O(n)$ messages, then all these messages can be routed from their sources to their destinations within $O(1)$ rounds. In the first forwarding step above, each node needs to send $O(\log^2 x \log n)$ bits—which is equivalent to $O(\log^2 x)$ messages—to its relay and each relay needs to receive at most $n/k \ll n$ messages from the nodes in its subset. Further, in the second forwarding step, each relay node needs to send $O(\log^2 x)$ messages to the leader u^* and the leader needs to receive at most $2k \cdot O(\log^2 x) = O(n/\log^2 x) \cdot O(\log^2 x) = O(n)$ messages from the relays. Hence, both of these steps can be easily performed using Lenzen's routing. □

2.3 Step (S2): Handling the High-Degree Components

In the second step (S2), each node v randomly and uniformly picks one of the edges incident on v and sends this edge to the leader u^*. The leader adds these edges to the forest, one by one, ignoring the cycles. This results in a new component decomposition $\mathcal{C}_{out}$. We next show the following.

Lemma 2.7. *W.h.p, the decomposition $\mathcal{C}_{out}$ contains at most $n/(2x)$ growable components.*

Proof. First note that by Lemma 2.3, the number of growable low-degree components in $\mathcal{C}'$ is at most $n/(4x)$. Hence, the total number of components of $\mathcal{C}_{out}$ that contain at least one growable low-degree component of $\mathcal{C}'$ is at most $n/(4x)$. Note that if a component in $\mathcal{C}_{out}$ contains a component of $\mathcal{C}'$ that was not growable, this actually means that the two components are the same and both are not growable. To

complete the proof, we show that the number of components of $\mathcal{C}_{out}$ that *do not* contain any low-degree component of $\mathcal{C}'$ is also at most $n/(4x)$.

Recall that for each component of $\mathcal{C}_{out}$, we call it *small* if it has less than $8x$ nodes, and *large* otherwise. Clearly, the number of large components of $\mathcal{C}_{out}$ is at most $n/(8x)$. Hence, to complete the proof, what remains is to show that in $\mathcal{C}_{out}$, the number of small components that *do not* contain a low-degree component from $\mathcal{C}'$ is at most $n/(8x)$.

To prove this, we consider the random process of adding edges to the forest gradually, as follows: This process is performed only for the sake of analysis. Throughout, we maintain a counter κ, which at the end will provide an upper bound on the number of small connected components in $\mathcal{C}_{out}$ that do not contain a low-degree component of C'.

First, mark all low-degree components of C' as *processed*, whereas all the high-degree components of C' are considered *unprocessed*. We gradually process unprocessed components and add the related sampled edges to the forest. For clarity, we perform this in phases. In phase $i \geq 1$, pick an arbitrary unprocessed small component C and let it be the head-component of the i^{th} chain. We then sample an edge – as will be described next – from the component C' that was joined last to this chain, where at the start, C' is simply C. To sample an edge from C', we pick an arbitrary vertex $v' \in C'$ of degree at least $x^5/8$ – and sample one of its edges uniformly at random. Since C' is both small and high-degree, such vertex v' exists.

If this randomly sampled edge connects to an unprocessed component C'', then we mark C'' as processed and add C'' to the chain. On the other hand, if the sampled edge connects C' to a component that is already processed, then the extension of the chain is halted and we update the counter κ in the following manner: If the sampled edge connects C' to a processed component within the *same* chain, and the total number of nodes in the connected component created by this chain is less than $8x$, then increment the counter κ by 1. If the sampled edge connects C' to a component in some previous chain, or to one of the low-degree components, then the counter κ remains as is. At this point, phase i ends. We continue this process until all components are marked as processed. See Figure 3 for an illustration.

If this randomly sampled edge connects to an unprocessed component C'', then we mark C'' as processed and add C'' to the chain. If the sampled edge connects C' to a component that is already processed, then the extension of the chain is halted and we update the counter κ in the following manner: If the sampled edge connects C' to a processed component within the *same* chain, and the total number of nodes in the connected component created by this chain is less than $8x$, then increment the counter κ by 1. On the other hand, if the sampled edge connects C' to a component in some previous chain, or to one of the low-degree components, then the counter κ remains as is. At this point, phase i ends. We continue this process until all components are marked as processed. See Figure 3 for an illustration.

Since the counter is incremented only when the chain consists of at most $8x$ nodes and when the random edge does not connect it to a previously counted component, we get that at the end, the number of low-cardinality components that do not contain a low-degree component from C' is upper bounded by κ. To complete the proof, we next show that with high probability, $\kappa \leq n/(8x)$.

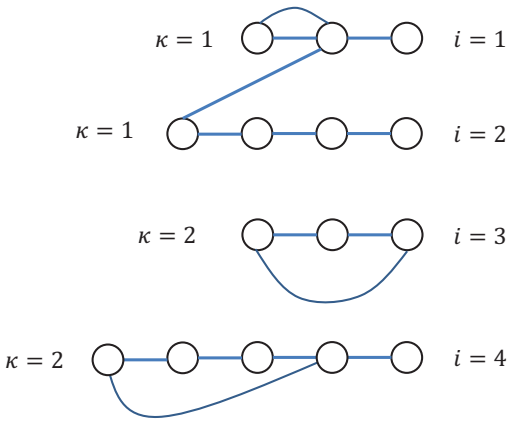

Figure 3: Illustration for Lemma 2.7. The figure shows the chain created at steps $i = 1, 2, 3, 4$. The counter κ is increased only when the sampled edge of the last added component is internal to the chain and the number of vertices in the chain's components is small. For example, the counter is not incremented in step $i = 2$, since the sampled edge hits the chain of step $i = 1$. In addition, the counter is not incremented in step $i = 4$, because the total number of vertices in the chain's components is above $8x$.

To prove this claim, let κ_i be the value of the counter at the end of phase i and let ℓ be the total number of phases. We first claim that for every $i \in \{1, \ldots, \ell\}$, $\mathbb{E}[\kappa_i | \kappa_{i-1}] \leq \kappa_{i-1} + 1/x^2$. Consider the i^{th} chain $[C_{i_1}, \ldots, C_{i_k}]$ generated at phase i and let $v_{i_j} \in C_{i_j}$ be the vertex whose edge has been taken (i.e., the vertex with degree at least $x^5/8$) for every $j \in \{1, \ldots k\}$. The probability that the counter is increased by 1 at the end of the phase is bounded by the probability that somewhere throughout the steps during which the chain had at most $4x$ nodes, the following happens: for at least one of the nodes $v_{i_j} \in C_{i_j}$ which was sampled for an edge, the randomly selected edge had both of its endpoints in the same chain. Throughout these steps, for every $j \in \{1, \ldots, k\}$, since the out-degree of the vertex v_{i_j} is greater than $x^5/8 - (8x)^2 \geq x^5/16$, the probability that a random edge of the node $v_{i_j} \in C_{i_j}$ connects to some node in $\bigcup_{\ell=1}^{j} C_{i_\ell}$ is at most $\frac{(8x)^2}{x^5/16} \leq c \cdot x^3$. Hence, by applying the union bound on all $8x$ "bad" events, we get that the probability that at least one of these edges is internal is $c' \cdot x^2$. Thus, $\mathbb{E}[\kappa_i | \kappa_{i-1}] \leq \kappa_{i-1} + c'/x^2$. This implies $\mathbb{E}[\kappa_\ell] \leq c' \cdot n/x^2$. Next, note that probabilistic events of the counter being increased by 1 are independent among different chains, as in the case of each chain i, the event only depends on the random edges sampled from the component under consideration during phase i, and this component will not be processed again in later phases. In addition, since there are high-degree components, $x^6 \leq n$ which means $2n/x^2 = \Omega(\log n)$. Hence, by Chernoff bound, w.h.p., $\kappa_\ell \leq 2c' \cdot n/x^2 \leq n/(8x)$. □

2.4 Step (S3): Clean Up

We now explain how in $O(1)$ rounds, we can identify and deactivate small connected components of $\mathcal{C}_{out}$ that are not growable. By Lemma 2.7 the number of growable components is at most $n/(2x)$. Moreover, the number of (non-growable) large components is at most $n/(8x)$. Therefore, once we deactivate small non-growable components, the total number of active components would be at most

$n/(8x)+n/(2x) \leq n/x$, as desired. We remark that we perform this clean up step only when $x \leq O(\log^2 n)$, that is, for all iterations of our connectivity algorithm except the very last one where we set $x = (n+1)$. Notice that in that last iteration, w.h.p., after step (S1), all components are non-growable—i.e., the forest is maximal—and thus we do not need to continue steps (S2) and (S3).

Lemma 2.8. *In $O(1)$ rounds, we can identify and deactivate component of F that are small and non-growable.*

Proof. Note that the leader u^* knows the whole forest F at the end of step (S2). For each small component C_i of F—that is, each component that has less than $8x$ nodes—the leader picks one relay node w_i. Then, the leader sends the identifiers of all nodes of component C_i to the relay node w_i, simultaneously for all small components, in O(1) rounds, using Lenzen's routing [Len13a]. Then, each relay node w_i that has received identifiers $v_1, \ldots, v_\ell$ of its corresponding component, delivers all of these identifiers to all of them ℓ nodes. That is, relay w_i creates ℓ messages destined for each of these nodes v_i where each of the messages describes the identifiers of $v_1, \ldots, v_\ell$, for a total of at most $(8x)^2 \leq O(\log^4 n)$ messages. We then use Lenzen's routing [Len13a] to deliver all these messages from relay nodes to nodes of small components. Note that this can be done because each node needs to send $(8x)^2 \leq O(\log^4 n)$ messages and each node needs to receive at most $(8x) \leq O(\log^2 n)$ messages. Finally, each node in these small components that now has received the identifiers of all its component-mates checks whether it has an edge going out of its component and informs the leader about this, using a single message, all of which get delivered to the leader in one round. Hence, at the end, the leader knows which of these small components are not growable, and then it marks them as deactivated. □

3. MST IN $O(\log^* N)$ ROUNDS

Here, we explain how using the graph connectivity algorithm presented in the previous section, we can compute a Minimum Spanning Tree (MST) of a weighted graph $G = (V, E, w)$ in $O(\log^* n)$ rounds.

Technically, the solution relies on three components: (1) a result of Karger, Klein, and Tarjan (KKT) [KKT95] that reduces the general MST problem to MST on sparse graphs, (2) a method of Hegeman et al. [HPP+15] which turns MST of graphs with low sparsity into many independent instances of the graph connectivity problem, and (3) an extension of our connectivity algorithm from the previous section that solves many instances of the connectivity problem simultaneously in $O(\log^* n)$ rounds. In the following, we discuss these three components.

MST Sparsification via KKT: We explain how a randomized trick of Karger, Klein, and Tarjan (KKT) [KKT95] transforms the MST problem on graph G into two MST problems on graphs G_1 and G_2, each with $O(n^{3/2})$ edges.

We first need some notations. For a graph G, let F be a forest, and for two nodes u and v, let $w_F(u,v)$ be the heaviest edge on the path connecting u and v in F; if there is no such path, let $w_F(u,v) = \infty$. We call an edge $e = (u,v)$ is *F-heavy* if $w(u,v) > w_F(u,v)$, and F-light otherwise. Notice that F-heavy edges cannot be in the MST of G.

To transform MST on G to MST on two graphs G_1 and G_2 with with $O(n^{3/2})$ edges each, let $p = \frac{1}{\sqrt{n}}$ and let $G_1 = G(p)$ be a randomly selected subgraph of G where each G-edge is included in $G(p)$ independently with probability p. W.h.p., G_1 has $O(n^{3/2})$ edges. Now suppose we have already computed an *minimum spanning forest* F of G_1, let E_2 be the set of F-light edges of G, and let $G_2 = (V, E_2)$. Clearly, an MST of G_2 is an MST G. The result of Karger, Klein, and Tarjan (KKT) [KKT95] shows that G_2 has at most $O(n/p) = O(n^{3/2})$ edges, w.h.p. Hence, the task of computing an MST of G is reduced to computing an MST F of G_1, identifying the set E_2 of F-light edges, and then computing an MST of G_2. Since in the Congested Clique the forest F can be made known to all nodes in $O(1)$ round, the task is reduced to solving MST in two graphs each with $O(n^{3/2})$ edges, one after the other. Next we explain how to solve each of these problems in $O(\log^* n)$ rounds.

From Sparse MST to Many Connectivity Problems: Consider a weighted graph $G' = (V, E', w)$ with $O(n^{3/2})$ edges. Consider the ordering π of the edges E' by increasing weight. Notice that an edge $e = (u,v)$ is in the MST of G' if and only if in the graph edge-induced by the set of edges lighter than e—i.e., those appearing before e in this ordering π—nodes u and v are in two different components. We next show, following [HPP+15], that one can check this condition for all edges of E' simultaneously—thus computing the MST—by solving $\Theta(\sqrt{n})$ graph connectivity problems.

For i from 1 to $K = \Theta(\sqrt{n})$, define subset E_i to be the edges with rank in $[(i-1)n+1, in]$ in the aforementioned ordering π of edges in E'. Use the Congested Clique sorting result of Lenzen [Len13a] so that in $O(1)$ rounds, each node knows the ranking of its incident edges in the ordering. Thus, each node knows for each of its incident edges e, to which edge-set E_i edge e belongs. Pick K leader nodes u_1, $\ldots$, u_K, and then using Lenzen's routing [Len13a], in $O(1)$ rounds, make each leader node u_i know all the edges in set E_i. Leader u_i will be responsible for detecting which of the edges in E_i belong to the MST. Let $H_i = (v, \cup_{j=1}^{i-1} E_j)$. If u_i knows the connected components—or equivalently, a maximal forest F_i—of H_i, then it can locally identify which of E_i edges belong to the MST, as follows: u_i processes the E_i edges one by one in the increasing weight order; each time it adds the new edge to the forest F_i, and also to the MST of G', and then discards all the unprocessed E_i edges e such that $F_i + e$ has a loop, that is, edges e that are internal to a component of F_i.

Thus, to solve MST of G', what remains is that each leader node u_i should learn the connected components (equivalently, a maximal forest) of graph H_i and this should happen for all $K = \Theta(\sqrt{n})$ graphs H_i. This is where our $O(\log^* n)$ connectivity algorithm comes in, as we discuss next.

Solving Many Connectivity Problems: Here, we explain how to extend the algorithm of the previous section to solve connectivity in many graphs simultaneously. Particularly, this extension will allow the leaders u_1, u_2, $\ldots$, u_K to know, respectively, the connected component decompositions of graphs $H_1 = (V, E_1)$, $H_2 = (V, E_2)$ $\ldots$, $H_K = (V, E_K)$, in $O(\log^* n)$ rounds.

Recall from Section 2 that to solve one instance of the graph connectivity problem, our main subroutine had three steps: (S1) a step where we deliver the sketches of components to the leader, (S2) a step where we deliver to the leader one randomly selected edge incident on each node,

and (S3) a clean-up step where we identify and deactivate small connected components that cannot grow.

Clearly, the step (S2) can be performed in parallel for all the K graphs $H_1, H_2, \ldots, H_k$, in just one round. This is because, each node v can directly send one message to each leader u_i, describing the (at most) one edge in H_i incident on v that it sampled. We next explain how steps (S1) and (S3) can also be performed in $O(1)$ rounds, simultaneously for all graphs H_i.

In the next lemma, we show that step (S1) can be performed in $O(1)$ rounds, simultaneously for all the K graphs H_i. This is by a simple extension of Lemma 2.6, as follows:

Lemma 3.1. *Consider the setting where each leader node u_i knows a forest F_i with at most $k = O(n/\log^2 x)$ active connected components in graph H_i. There is an $O(1)$-round routing scheme that, for all $i \in [1, K]$, simultaneously delivers the sketches of all components of F_i to the leader u_i.*

Proof. The approach is mostly the same as in Lemma 2.6. Here we just point out the simple changes that need to be made. As in Lemma 2.6, each leader u_i partitions the node-set of each of the active connected components of F_i into subsets of size n/k where $k = \Theta(n/\log^2 x)$. This breaks the forest F_i into at most $2k$ subsets. Leader u_i picks (at most) $2k$ relay nodes $w_1, w_2, \ldots, w_{2k}$, one for each of these subsets. In fact, we can allow all the leaders to pick the same $2k$ relay nodes, particularly the $2k$ nodes with smallest IDs (although the assignments will be different). Then each leader u_i delivers the ID of relay node w_j to the nodes in the j^{th} subset in the partition of F_i. We can do this at the same time for all the K leaders because to deliver these relay node IDs, each node v needs to receive one ID from each leader u_i, and it can directly receive it from the leader.

Now, each node v sends its sketch for each graph H_i to its corresponding relay node. This requires v to send out a total of $K \cdot \Theta(\log^2 x) \leq \Theta(\sqrt{n}\log^2 n) \ll n$ messages and it requires each relay node to receive at most $K \cdot n/k = \Theta(\sqrt{n}\log^2 n) \ll n$ messages. Hence, the step of delivering sketches from nodes to relays can be done in $O(1)$ rounds, using Lenzen's routing [Len13b].

Finally, we need to deliver the sketches from the relay nodes to the leaders. This requires each relay to send at most $\Theta(K\log^2 x) \leq \Theta(\sqrt{n}\log^2 n) \ll n$ messages and each leader to receive at most $2k \cdot \Theta(\log^2 x) = O(n/\log^2 x) \cdot \Theta(\log^2 x) = O(n)$ messages. Thus, this again be done in $O(1)$ rounds using Lenzen's routing [Len13b]. □

We remark how Lemma 2.5 extends to solving K connectivity problems simultaneously. Recall that Lemma 2.5 discusses how two nodes u and v connected by an edge $e = (u, v) \in G$ can know consistently whether e is sampled in each of the sketch subsets or not, simply by $O(\log n)$ bits of communication. Instead of repeating this for the K graph connectivity problems we want to solve, we make the $O(\log n)$ bits describe the sampling outcomes in the union graph $\cup_{i\in[K]} H_i$, and then u and v can use the edge e for the computation of each graph H_i, considering whether $e \in H_i$ or not. That is, we use the same $O(\log n)$ bits of randomness for all the connectivity problems. This makes the execution of these graph connectivity problems probabilistically dependent. However, since we know that each of them succeeds with high probability, and since they run independent of each other, we do not need any independence; we can do a simple union bound and say that, with high probability, all the graph connectivity instances succeed.

Finally, it remains to explain how Lemma 2.8 can be extended to perform step (S3) for all the graphs:

Lemma 3.2. *In $O(1)$ rounds, simultaneously we can identify and deactivate components of the forests $F_1, F_2, \ldots, F_k$ of graphs $H_1, H_2, \ldots, H_k$ that are small and non-growable.*

Proof. The procedure for each graph is the same as in Lemma 2.8. The only point to note is that, all these messages can be routed simultaneously using Lenzen's routing. Particularly, each relay node will need to receive at most $K(8x) \leq K \cdot O(\log^2 n) \leq O(\sqrt{n}\log^2 n)$ messages from the corresponding leaders and each relay node will need to send at most $K(8x)^2 \leq O(\sqrt{n}\log^4 n)$ messages to the nodes. Both of these bounds are much smaller than n and thus, Lenzen's routing [Len13a] can route all these messages simultaneously in $O(1)$ rounds. □

4. REFERENCES

[AGM12] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proc. of ACM-SIAM Symp. on Disc. Alg. (SODA)*, pages 459–467, 2012.

[BFARR15] Florent Becker, Antonio Fernandez Anta, Ivan Rapaport, and Eric Reémila. Brief announcement: A hierarchy of congested clique models, from broadcast to unicast. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, PODC '15, pages 167–169. ACM, 2015.

[BHP12] Andrew Berns, James Hegeman, and Sriram V Pemmaraju. Super-fast distributed algorithms for metric facility location. In *Automata, Languages, and Programming*, pages 428–439. Springer, 2012.

[CHKK+15] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 143–152. ACM, 2015.

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[DKO14] Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 367–376. ACM, 2014.

[DLP12] Danny Dolev, Christoph Lenzen, and Shir Peled. âĂIJtri, tri againâĂİ: Finding triangles and small subgraphs in a distributed setting. In *Distributed Computing*, pages 195–209. Springer, 2012.

[Erd42] Pául Erdos. On an elementary proof of some asymptotic formulas in the theory of partitions. *Annals of Mathematics*, pages 437–450, 1942.

[Gha16] Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proc. of ACM-SIAM Symp. on Disc. Alg. (SODA)*, 2016.

[HP14] James W Hegeman and Sriram V Pemmaraju. Lessons from the congested clique applied to MapReduce. In *the Proceedings of the International Colloquium on Structural Information and Communication Complexity*, pages 149–164. Springer, 2014.

[HP15] James W Hegeman and Sriram V Pemmaraju. Lessons from the congested clique applied to mapreduce. *Theoretical Computer Science*, 608:268–281, 2015.

[HPP+15] 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 *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 91–100. ACM, 2015.

[HPS14] James W Hegeman, Sriram V Pemmaraju, and Vivek B Sardeshmukh. Near-constant-time distributed algorithms on a congested clique. In *Distributed Computing*, pages 514–530. Springer, 2014.

[KKM13] Bruce M Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1131–1142. SIAM, 2013.

[KKT95] David R Karger, Philip N Klein, and Robert E Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM (JACM)*, 42(2):321–328, 1995.

[KNPR15] Hartmut Klauck, Danupon Nanongkai, Gopal Pandurangan, and Peter Robinson. Distributed computation of large-scale graph problems. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 391–410. SIAM, 2015.

[Knu76] Donald Ervin Knuth. Mathematics and computer science: coping with finiteness. *Science (New York, NY)*, 194(4271):1235–1242, 1976.

[KSV10] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 938–948. Society for Industrial and Applied Mathematics, 2010.

[Len13a] Christoph Lenzen. Optimal deterministic routing and sorting on the congested clique. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 42–50, 2013.

[Len13b] Christoph Lenzen. Optimal deterministic routing and sorting on the congested clique. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 42–50, 2013.

[Lin92] Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[LPPSP03] Zvi Lotker, Elan Pavlov, Boaz Patt-Shamir, and David Peleg. MST construction in $O(\log \log n)$ communication rounds. In *the Proceedings of the Symposium on Parallel Algorithms and Architectures*, pages 94–100. ACM, 2003.

[Nan14] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proc. of the Symp. on Theory of Comp. (STOC)*, 2014.

[NMN01] Jaroslav Nešetřil, Eva Milková, and Helena Nešetřilová. Otakar boruvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. *Discrete Mathematics*, 233(1):3–36, 2001.

[NN93] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM journal on computing*, 22(4):838–856, 1993.

[Pel00] David Peleg. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[PRS15] Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. Almost optimal distributed algorithms for large-scale graph problems. *arXiv preprint arXiv:1503.02353*, 2015.

[PST11] Boaz Patt-Shamir and Marat Teplitsky. The round complexity of distributed sorting. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 249–256, 2011.