Fast Distributed Algorithms for Connectivity and MST in Large Graphs

Gopal Pandurangan*
Dept. of Computer Science
University of Houston
gopalpandurangan@gmail.com

Peter Robinson
Royal Holloway
University of London
peter.robinson@rhul.ac.uk

Michele Scquizzato
Dept. of Computer Science
University of Houston
michele@cs.uh.edu

ABSTRACT

Motivated by the increasing need to understand the algorithmic foundations of distributed large-scale graph computations, we study a number of fundamental graph problems in a message-passing model for distributed computing where $k \geq 2$ machines jointly perform computations on graphs with n nodes (typically, $n \gg k$). The input graph is assumed to be initially randomly partitioned among the k machines, a common implementation in many real-world systems. Communication is point-to-point, and the goal is to minimize the number of communication rounds of the computation.

Our main result is an (almost) optimal distributed randomized algorithm for graph connectivity. Our algorithm runs in $\tilde{O}(n/k^2)$ rounds ($\tilde{O}$ notation hides a polylog(n) factor and an additive polylog(n) term). This improves over the best previously known bound of $\tilde{O}(n/k)$ [Klauck et al., SODA 2015], and is optimal (up to a polylogarithmic factor) in view of an existing lower bound of $\tilde{\Omega}(n/k^2)$. Our improved algorithm uses a bunch of techniques, including linear graph sketching, that prove useful in the design of efficient distributed graph algorithms. We then present fast randomized algorithms for computing minimum spanning trees, (approximate) min-cuts, and for many graph verification problems. All these algorithms take $\tilde{O}(n/k^2)$ rounds, and are optimal up to polylogarithmic factors. We also show an almost matching lower bound of $\tilde{\Omega}(n/k^2)$ for many graph verification problems using lower bounds in random-partition communication complexity.

Keywords

Distributed graph algorithms; massive graphs; graph sketching; graph connectivity; minimum spanning trees

*Supported, in part, by US-Israel Binational Science Foundation grant 2008348, NSF grant CCF-1527867, and NSF grant CCF-1540512.

SPAA '16, July 11 - 13, 2016, Pacific Grove, CA, USA

ISBN 978-1-4503-4210-0/16/07…$15.00

DOI: http://dx.doi.org/10.1145/2935764.2935785

1. INTRODUCTION

The focus of this paper is on distributed computation on large-scale graphs, which is increasingly becoming important with the rise of massive graphs such as the Web graph, social networks, biological networks, and other graph-structured data and the consequent need for fast algorithms to process such graphs. Several large-scale graph processing systems such as Pregel [27] and Giraph [1] have been recently designed based on the message-passing distributed computing model [26, 34]. We study a number of fundamental graph problems in a model which abstracts the essence of these graph-processing systems, and present almost tight bounds on the time complexity needed to solve these problems. In this model, introduced in [19] and explained in detail in Section 1.1, the input graph is distributed across a group of $k \geq 2$ machines that are pairwise interconnected via a communication network. The k machines jointly perform computations on an arbitrary n-vertex input graph, where typically $n \gg k$. The input graph is assumed to be initially randomly partitioned among the k machines (a common implementation in many real world graph processing systems [27, 36]). Communication is point-to-point via message passing. The computation advances in synchronous rounds, and there is a constraint on the amount of data that can cross each link of the network in each round. The goal is to minimize the time complexity, i.e., the number of rounds required by the computation. This model is aimed at investigating the amount of "speed-up" possible vis-a-vis the number of available machines, in the following sense: when k machines are used, how does the time complexity scale in k? Which problems admit linear scaling? Is it possible to achieve super-linear scaling?

[19] presents lower and upper bounds for several fundamental graph problems in the k-machine model. In particular, for the graph connectivity problem it shows a lower bound of $\tilde{\Omega}(n/k^2)$ rounds.[1] It also presents an $\tilde{O}(n/k)$-round algorithm for graph connectivity and spanning tree (ST) verification. This algorithm thus exhibits a scaling linear in the number of machines k. The question of existence of a faster algorithm, and in particular of an algorithm matching the $\tilde{\Omega}(n/k^2)$ lower bound, was left open in [19]. In this paper we answer this question affirmatively by presenting an $\tilde{O}(n/k^2)$-round algorithm for graph connectivity, thus achieving a speedup quadratic in k. This is optimal up to polylogarithmic (in n) factors.

[1]Throughout this paper $\tilde{O}(f(n))$ denotes $O(f(n)\operatorname{polylog} n + \operatorname{polylog} n)$, and $\tilde{\Omega}(f(n))$ denotes $\Omega(f(n)/\operatorname{polylog} n)$.

This result is important for two reasons. First, it shows that there are non-trivial graph problems for which we can obtain *superlinear* (in k) speed-up. To elaborate further on this point, we shall take a closer look at the proof of the lower bound for connectivity shown in [19]. Using communication complexity techniques, that proof shows that any (possibly randomized) algorithm for the graph connectivity problem has to exchange $\tilde{\Omega}(n)$ bits of information across the k machines, for any $k \geq 2$. Since there are $k(k-1)/2$ links in a complete network with k machines, when each link can carry $O(\text{polylog}(n))$ bits per round, in each single round the network can deliver at most $\tilde{\Theta}(k^2)$ bits of information, and thus a lower bound of $\tilde{\Omega}(n/k^2)$ rounds follows. The connectivity algorithm of this paper thus shows that the optimal speed-up factor of $\Theta(k^2)$ is attainable. Second, this implies that many other important graph problems can be solved in $\tilde{O}(n/k^2)$ rounds as well. These include computing a spanning tree, minimum spanning tree (MST), approximate min-cut, and many verification problems such as spanning connected subgraph, cycle containment, and bipartiteness.

It is important to note that under a different output requirement (explained next) there exists a $\tilde{\Omega}(n/k)$-round lower bound for computing a spanning tree of a graph [19], which also implies the same lower bound for other fundamental problems such as computing an MST, breadth-first tree, and shortest paths tree. However, this lower bound holds under the requirement that each vertex (i.e., the machine which hosts the vertex) must know at the end of the computation the "status" of all of its incident edges, that is, whether they belong to a ST or not, and output their respective status. (This is the output criterion that is usually required in distributed algorithms [26, 34].) The proof of the lower bound exploits this criterion to show that any algorithm will require some machine receiving $\Omega(n)$ bits of information, and since any machine has $k-1$ incident links, this results in a $\tilde{\Omega}(n/k)$ lower bound. On the other hand, if we relax the output criterion to require the final status of each edge to be known by *some* machine (different machines might know the status of different edges), then we show that this can be accomplished in $\tilde{O}(n/k^2)$ rounds using the fast connectivity algorithm of this paper.

1.1 The Model

We now describe the adopted model of distributed computation, the *k-machine model* (a.k.a. the *Big Data model*), introduced in [19] and further investigated in [9, 35, 33]. The model consists of a set of $k \geq 2$ machines $N = \{M_1, M_2, \ldots, M_k\}$ that are pairwise interconnected by bidirectional point-to-point communication links. Each machine executes an instance of a distributed algorithm. The computation advances in synchronous rounds where, in each round, machines can exchange messages over their communication links and perform some local computation. Each link is assumed to have a bandwidth of $O(\text{polylog}(n))$ bits per round, i.e., $O(\text{polylog}(n))$ bits can be transmitted over each link in each round. (As discussed in [19] (cf. Theorem 4.1), it is easy to rewrite bounds to scale in terms of the actual inter-machine bandwidth.) Machines do not share any memory and have no other means of communication. There is an alternate (but equivalent) way to view this communication restriction: instead of putting a bandwidth restriction on the links, we can put a restriction on the amount of information that each *machine* can communicate (i.e., send/receive) in each round. The results that we obtain in the bandwidth-restricted model will also apply to the latter model [19]. Local computation within a machine is considered to happen instantaneously at zero cost, while the exchange of messages between machines is the costly operation. (However, we note that in all the algorithms of this paper, every machine in every round performs a computation bounded by a polynomial in n.) We assume that each machine has access to a private source of true random bits.

Although the k-machine model is a fairly general model of computation, we are mostly interested in studying graph problems in it. Specifically, we are given an input graph G with n vertices, each associated with a unique integer ID from $[n]$, and m edges. To avoid trivialities, we will assume that $n \geq k$ (typically, $n \gg k$). Initially, the entire graph G is not known by any single machine, but rather partitioned among the k machines in a "balanced" fashion, i.e., the nodes and/or edges of G are partitioned approximately evenly among the machines. We assume a *vertex-partition* model, whereby vertices, along with information of their incident edges, are partitioned across machines. Specifically, the type of partition that we will assume throughout is the *random vertex partition (RVP)*, that is, each vertex of the input graph is assigned randomly to one machine. (This is the typical way used by many real systems, such as Pregel [27], to partition the input graph among the machines; it is easy to accomplish, e.g., via hashing.[2]) However, we notice that our upper bounds also hold under the much weaker assumption whereby it is only required that nodes and edges of the input graph are partitioned approximately evenly among the machines; on the other hand, lower bounds under RVP clearly apply to worst-case partitions as well.

More formally, in the random vertex partition variant, each vertex of G is assigned independently and uniformly at random to one of the k machines. If a vertex v is assigned to machine M_i we say that M_i is the *home machine* of v and, with a slight abuse of notation, write $v \in M_i$. When a vertex is assigned to a machine, *all its incident edges* are assigned to that machine as well; i.e., the home machine will know the IDs of the neighbors of that vertex as well as the identity of the home machines of the neighboring vertices (and the weights of the corresponding edges in case G is weighted). Note that an immediate property of the RVP model is that the number of vertices at each machine is *balanced*, i.e., each machine is the home machine of $\tilde{\Theta}(n/k)$ vertices with high probability. A convenient way to implement the RVP model is through hashing: each vertex (ID) is hashed to one of the k machines. Hence, if a machine knows a vertex ID, it also knows where it is hashed to.

Eventually, each machine M_i, for each $1 \leq i \leq k$, must set a designated local output variable o_i (which need not depend on the set of vertices assigned to M_i), and the *output configuration* $o = \langle o_1, \ldots, o_k \rangle$ must satisfy certain feasibility conditions for the problem at hand. For example, for the minimum spanning tree problem each o_i corresponds to a set of edges, and the edges in the union of such sets must form an MST of the input graph. The *time complexity* of an algorithm is the maximum number of rounds until termination.

[2]In Section 1.3 we will discuss an alternate partitioning model, the *random edge partition (REP)* model, where each edge of G is assigned independently and randomly to one of the k machines, and show how the results in the random vertex partition model can be related to the random edge partition model.

1.2 Our Contributions and Techniques

The main result of this paper, presented in Section 2, is a randomized Monte Carlo algorithm in the k-machine model that determines the connected components of an undirected graph G correctly with high probability and that terminates in $\tilde{O}(n/k^2)$ rounds.[3] This improves upon the previous best bound of $\tilde{O}(n/k)$ [19], since it is strictly superior in the wide range of parameter $k = \Theta(n^\epsilon)$, for all constants $\epsilon \in (0, 1)$. Improving over this bound is non-trivial since various attempts to get a faster connectivity algorithm fail due to the fact that they end up congesting a particular machine too much, i.e., up to n bits may need to be sent/received by a machine, leading to a $\tilde{O}(n/k)$ bound (as a machine has only $k-1$ links). For example, a simple algorithm for connectivity is simply flooding: each vertex floods the lowest labeled vertex that it has seen so far; at the end each vertex will have the label of the lowest labeled vertex in its component.[4] It can be shown that the above algorithm takes $\Theta(n/k + D)$ rounds (where D is the graph diameter) in the k-machine model by using the Conversion Theorem of [19]. Hence new techniques are needed to break the n/k-round barrier.

Our connectivity algorithm is the result of the application of the following three techniques.

1. Randomized Proxy Computation. This technique, similar to known techniques used in randomized routing algorithms [39], is used to load-balance congestion at any given machine by redistributing it evenly across the k machines. This is achieved, roughly speaking, by re-assigning the executions of individual nodes uniformly at random among the machines. It is crucial to distribute the computation *and* communication across machines to avoid congestion at any particular machine. In fact, this allows one to move away from the communication pattern imposed by the topology of the input graph (which can cause congestion at a particular machine) to a more balanced communication.

2. Distributed Random Ranking (DRR). DRR [8] is a simple technique that will be used to build trees of low height in the connectivity algorithm. Our connectivity algorithm is divided into phases, in each of which we do the following: each current component (in the first phase, each vertex is a component by itself) chooses one *outgoing edge* and then components are combined by merging them along outgoing edges. If done naively, this may result in a long chain of merges, resulting in a component tree of high diameter; communication along this tree will then take a long time. To avoid this we resort to DRR, which suitably reduces the number of merges. With DRR, each component chooses a random *rank*, which is simply a random number, say in the interval $[1, n^3]$; a component (say C_1) will merge with the component on the other side of its selected outgoing edge (say C_2) if and only if the rank of C_2 is larger than the rank of C_1. If the rank of C_1 is higher, then C_1 does not merge with C_2 and thus it becomes the root of a DRR tree, which is a tree induced by the components and the set of the outgoing edges that have been used in the above merging procedure. We will show that the height of a DRR tree is bounded by $O(\log n)$ with high probability.

3. Linear Graph Sketching. Linear graph sketching [2, 3, 28] is crucially helpful in efficiently finding an outgoing edge of a component. A *sketch* for a vertex (or a component) is a short ($O(\text{polylog}\, n)$) bit vector that efficiently encodes the adjacency list of the vertex. Sampling from this sketch gives a random (outgoing) edge of this vertex (component). A very useful property is the linearity of the sketches: adding the sketches of a set of vertices gives the sketch of the component obtained by combining the vertices; the edges between the vertices (i.e., the intra-component edges) are automatically "cancelled", leaving only a sketch of the outgoing edges. Linear graph sketches were originally used to process dynamic graphs in the (semi-) streaming model [2, 3, 28]. Here, in a distributed setting, we use them to reduce the amount of communication needed to find an outgoing edge; in particular, graph sketches will avoid us from checking whether an edge is an inter-component or an intra-component edge, and this will crucially reduce communication across machines. We note that earlier distributed algorithms such as the classical GHS algorithm [14] for the MST problem would incur too much communication since they involve checking the status of each edge of the graph.

We observe that it does not seem straightforward to effectively exploit these techniques in the k-machine model: for example, linear sketches can be easily applied in the distributed streaming model by sending to a coordinator machine the sketches of the partial stream, which then will be added to obtain the sketch of the entire stream. Mimicking this trivial strategy in the k-machine model model would cause too much congestion at one node, leading to a $\tilde{O}(n/k)$ time bound.

Using the above techniques and the fast connectivity algorithm, in Section 3 we give algorithms for many other important graph problems. In particular, we present a $\tilde{O}(n/k^2)$-round algorithm for computing an MST (and hence an ST). We also present $\tilde{O}(n/k^2)$-round algorithms for approximate min-cut, and for many graph verification problems including spanning connected subgraph, cycle containment, and bipartiteness. All these algorithms are optimal up to a polylogarithmic factor. In Section 4 we show a lower bound of $\tilde{\Omega}(n/k^2)$ rounds for many verification problems by simulating the k-machine model in a 2-party model of communication complexity where the inputs are randomly assigned to the players. This simulation has to be done in a careful manner since the k-machine model is synchronous, whereas the communication complexity model is asynchronous.

1.3 Related Work and Comparison

The theoretical study of large-scale graph computations in distributed systems is relatively new. Several works have been devoted to developing MapReduce graph algorithms (see, e.g., [18, 21, 24] and references therein). We note that the flavor of the theory developed for MapReduce is quite different compared to the one for the k-machine model. Minimizing communication is also the key goal in MapReduce algorithms; however this is usually achieved by making sure that the data is made small enough quickly (that is, in a small number of MapReduce rounds) to fit into the memory of a single machine (see, e.g., the MapReduce algorithm for MST in [21]).

For a comparison of the k-machine model (a.k.a Big Data

[3] Since the focus is on the scaling of the time complexity with respect to k, we omit explicitly stating the polylogarithmic factors in our run time bounds. However, the hidden polylogarithmic factor is not large—at most $O(\log^3 n)$.

[4] This algorithm has been implemented in a variant of Giraph [38].

model) with other parallel and distributed models proposed for large-scale data processing, including Bulk-Synchronous Parallel (BSP) model [40], MapReduce [18], and the congested clique, we refer to [41]. In particular, according to [41], "Among all models with restricted communication the "big data" [k-machine] model is the one most similar to the MapReduce model".

The k-machine model is closely related to the BSP model; it can be considered to be a simplified version of BSP, where local computation is ignored and synchronization happens at the end of every round (the synchronization cost is ignored). Unlike the BSP and refinements thereof, which have several different parameters that make analysis of algorithms complicated [41], the k-machine model is characterized by just one parameter, the number of machines, and this might ease the job of designing and analyzing algorithms.

The k-machine model is related to the classical $\mathcal{CONGEST}$ model [34], and in particular to the *congested clique* model, which recently has received considerable attention (see, e.g., [25, 23, 22, 12, 30, 7, 15]). The main difference is that the k-machine model is aimed at the study of large-scale computations, where the size n of the input is significantly bigger than the number of available machines k, and thus many vertices of the input graph are mapped to the same machine, whereas the two aforementioned models are aimed at the study of distributed network algorithms, where $n = k$ and thus each vertex corresponds to a dedicated machine. More "local knowledge" is available per vertex (since it can access for free information about other vertices in the same machine) in the k-machine model compared to the other two models. On the other hand, all vertices assigned to a machine have to communicate through the links incident on this machine, which can limit the bandwidth (unlike the other two models where each vertex has a dedicated processor). These differences manifest in the time complexity. In particular, the fastest known distributed algorithm in the congested clique model for a given problem may not give rise to the fastest algorithm in the k-machine model. For example, the fastest algorithms for MST in the congested clique model ([25, 15]) require $\Theta(n^2)$ messages; implementing these algorithms in the k-machine model requires $\Theta(n^2/k^2)$ rounds. Conversely, the slower GHS algorithm [14] gives an $\tilde{O}(n/k)$ bound in the k-machine model. The recently developed techniques (see, e.g., [11, 31, 13, 15, 12]) used to prove time lower bounds in the standard $\mathcal{CONGEST}$ model and in the congested clique model are not directly applicable here.

The work closest in spirit to ours is the recent work of Woodruff and Zhang [42]. This paper considers a number of basic statistical and graph problems in a distributed message-passing model similar to the k-machine model. However, there are some important differences. First, their model is asynchronous, and the cost function is the communication complexity, which refers to the total number of bits exchanged by the machines during the computation. Second, a *worst-case* distribution of the input is assumed, while we assume a random distribution. Third, which is an important difference, they assume an edge partition model for the problems on graphs, that is, the edges of the graph (as opposed to its vertices) are partitioned across the k machines. In particular, for the connectivity problem, they show a message complexity lower bound of $\tilde{\Omega}(nk)$ which essentially translates to a $\tilde{\Omega}(n/k)$ round lower bound in the k-machine model; it can be shown by using their proof technique that this lower bound also applies to the *random edge partition (REP)* model, where edges are partitioned randomly among machines, as well. On the other hand, it is easy to show an $\tilde{O}(n/k)$ upper bound for the connectivity in the REP model for connectivity and MST.[5] Hence, in the REP model, $\tilde{\Theta}(n/k)$ is a tight bound for connectivity and other related problems such as MST. However, in contrast, in the RVP model (arguably, a more natural partition model), we show that $\tilde{\Theta}(n/k^2)$ is the tight bound. Our results are a step towards a better understanding of the complexity of distributed graph computation vis-a-vis the partition model.

2. THE CONNECTIVITY ALGORITHM

In this section we present our main result, a Monte Carlo randomized algorithm for the k-machine model that determines the connected components of an undirected graph G correctly with high probability and that terminates in $\tilde{O}(n/k^2)$ rounds with high probability (Theorem 1). This algorithm is optimal (up to polylog(n)-factors) by virtue of a matching lower bound of $\tilde{\Omega}(n/k^2)$ rounds, which was shown in [19] by a reduction from the 2-party communication complexity of set disjointness under random input partitions.

Before delving into the details of our algorithm, as a warm-up we briefly discuss simpler, but less efficient, approaches. The easiest way to solve *any* problem in our model is to first collect all available graph data at a single machine and then solve the problem locally. For example, one could first elect a referee among the machines, which requires $O(1)$ rounds [20], and then instruct every machine to send its local data to the referee machine. Since the referee machine needs to receive $O(m)$ information in total but has only $k-1$ links of bounded bandwidth, this requires $\Omega(m/k)$ rounds.

A more refined approach to obtain a distributed algorithm for the k-machine model is to use the Conversion Theorem of [19], which provides a simulation of a congested clique algorithm $\mathcal{A}$ in $\tilde{O}(M/k^2 + \Delta' T/k)$ rounds in the k-machine model, where M is the message complexity of $\mathcal{A}$, T is its round complexity, and Δ' is an upper bound to the total number of messages sent (or received) by a single node in a single round. (All these parameters refer to the performance of $\mathcal{A}$ in the congested clique model.) Unfortunately, existing algorithms (e.g., [14, 37]) typically require Δ' to scale to the maximum node degree, and thus the converted time complexity bound in the k-machine model becomes $\tilde{\Omega}(n/k)$ at best. Therefore, in order to break the $\tilde{\Omega}(n/k)$ barrier, we must develop new techniques that directly exploit the additional locality available in the k-machine model.

In the next subsection we give a high level overview of our algorithm, and then formally present all the technical details in the subsequent subsections.

2.1 Overview of the Algorithm

Our algorithm follows a Boruvka-style strategy [6], that is, it repeatedly merges adjacent *components* of the input graph,

[5]The high-level idea of the MST algorithm in the REP model is: (1) First "filter" the edges assigned to one machine using the cut and cycle properties of a MST [17]; this leaves each machine with $O(n)$ edges; (2) Convert this edge distribution to a RVP which can be accomplished in $\tilde{O}(n/k)$ rounds via hashing the vertices randomly to machines and then routing the edges appropriately; then apply the RVP bound.

which are connected subgraphs, to form larger (connected) components. The output of each of these phases is a *labeling* of the nodes of G such that nodes that belong to the same current component have the same label. At the beginning of the first phase, each node is labeled with its own unique ID, forms a distinct component, and is also the *component proxy* of its own component. Note that, at any phase, a component contains up to n nodes, which might be spread across different machines; we use the term *component part* to refer to all those nodes of the component that are held by the same machine. Hence, at any phase every component is partitioned in at most k component parts. At the end of the algorithm, every node $v \in G$ will have a component label $\ell(v)$ such that two nodes have the same label if and only if they belong to the same connected component of G.

Our algorithm relies on *linear graph sketches* as a tool to enable communication-efficient merging of multiple components. Intuitively speaking, a (random) linear sketch $\mathbf{s}_u$ of a node u's graph neighborhood returns a sample chosen uniformly at random from u's incident edges. Interestingly, such a linear sketch can be represented as matrices using only $O(\text{polylog}(n))$ bits [16, 28]. A crucial property of these sketches is that they are linear: that is, given sketches $\mathbf{s}_u$ and $\mathbf{s}_v$, the *combined sketch* $\mathbf{s}_u + \mathbf{s}_v$ ("+" refers to matrix addition) has the property that, w.h.p., it yields a random sample of the edges incident to (u, v) in a graph where we have contracted the edge (u, v) to a single node. We describe the technical details in Section 2.3.

We now describe how to communicate these graph sketches in an efficient manner: Consider a component C that is split into j parts $P_1, P_2, \ldots, P_j$, the nodes of which are hosted at machines $M_1, M_2, \ldots, M_j$. To find an outgoing edge for C, we first instruct each machine M_i to construct a linear sketch of the graph neighborhood of each of the nodes in part P_i. Then, we sum up these $|P_i|$ sketches, yielding a sketch $\mathbf{s}_{P_i}$ for the neighborhood of part P_i. To combine the sketches of the j distinct parts, we now select a *random component proxy* machine $M_{C,r}$ for the current component C at round r (see Section 2.2). Next, machine M_i sends $\mathbf{s}_{P_i}$ to machine $M_{C,r}$; note that this causes at most k messages to be sent to the component proxy. Finally, machine $M_{C,r}$ computes $\mathbf{s}_C = \sum_{i=1}^{j} \mathbf{s}_{P_i}$, and then uses $\mathbf{s}_C$ to sample an edge incident to some node in C, which, by construction, is guaranteed to have its endpoint in a distinct component C'. (See Section 2.4.)

At this point, each component proxy has sampled an inter-component edge inducing the edges of a *component graph* $\mathcal{C}$ where each vertex corresponds to a component. To enable the efficient merging of components, we employ the *distributed random ranking* (DRR) technique of [8] to break up any long paths of $\mathcal{C}$ into more manageable directed trees of depth $O(\log n)$. To this end, every component chooses a rank independently and uniformly at random from $[0, 1]$,[6] and each component (virtually) connects to its neighboring component (according to $\mathcal{C}$) via a (conceptual) directed edge if and only if the latter has a higher rank. Thus, this process results in a collection of disjoint rooted trees, rooted at the node of highest (local) rank. We show in Section 2.5 that each of such trees has depth $O(\log n)$.

The merging of the components of each tree $\mathcal{T}$ proceeds from the leafs upward (in parallel for each tree). In the first merging phase, each leaf C_j of $\mathcal{T}$ merges with its parent C' by relabeling the component labels of all of their nodes with the label of C'. Note that the proxy M_{C_j} knows the labeling of C', as it has computed the outgoing edge from a vertex in C_j to a vertex in C'. Therefore, machine M_{C_j} sends the label of C_j to all the machines that hold a part of C_j. In Section 2.5 we show that this can be done in parallel (for all leafs of all trees) in $\tilde{O}(n/k^2)$ rounds. Repeating this merging procedure $O(\log n)$ times, guarantees that each tree has been merged to a single component.

Finally, in Section 2.6 we prove that $O(\log n)$ repetitions of the above process suffice to ensure that the components at the end of the last phase correspond to the connected components of the input graph G.

2.2 Communication via Random Proxy Machines

Recall that our algorithm iteratively groups vertices into "components" (which are connected subgraphs of the actual components of G) and subsequently merges such components according to the topology of G. Each of these components may be split into multiple component parts spanning multiple machines. Hence, to ensure efficient load balancing of the messages that machines need to send on behalf of the component parts that they hold, the algorithm performs all communication via *proxy machines*.

Our algorithm proceeds in phases and each phase consists of iterations. Consider the ρ-iteration of the j-th phase of the algorithm, with $\rho, j \geq 1$. We construct a "sufficiently" random hash function $h_{j,\rho}$, such that, for each component C, the machine with ID $h_{j,\rho}(C) \in [k]$ is selected as the proxy machine for component C. First, machine M_1 generates $\ell = \tilde{\Theta}(n/k)$ random bits from its private source of randomness. M_1 will distribute these random bits to all other machines via the following simple routing mechanism that proceeds in sequences of two rounds. M_1 selects k bits $b_1, b_2, \ldots, b_{k-1}$ from the set of its ℓ private random bits that remain to be distributed, and sends bit b_i across its i-th link to machine M_{i+1}. Upon receiving b_i, machine M_{i+1} broadcasts b_i to all machines in the next round. This ensures that bits $b_1, b_2, \ldots, b_{k-1}$ become common knowledge within two rounds. Repeating this process to distribute all the $\ell = \tilde{\Theta}(n/k)$ bits takes $\tilde{O}(n/k^2)$ rounds, after those all the machines have the ℓ random bits generated by M_1. We leverage a result of [4] (cf. in its formulation as Theorem 2.1 in [5]), which tells us that we can generate a random hash function such that it is d-wise independent by using only $O(d \log n)$ true random bits. We instruct machine M_1 to disseminate $d = \ell \log n = n \,\text{polylog}(n)/k$ of its random bits according to the above routing process and then each machine locally constructs the same hash function $h_{j,\rho}$, which is then used to determine the component proxies throughout iteration ρ of phase j.

We now show that communication via such proxy machines is fast in the k-machine model.

LEMMA 1. *Suppose that each machine M generates a message of size $O(\text{polylog}(n))$ bits for each component part residing on M; let m_i denote the message of part P_i and let C be the component of which P_i is a part. If each m_i is addressed to the proxy machine M_C of component C, then all messages are delivered within $\tilde{O}(n/k^2)$ rounds with high probability.*

PROOF. Observe that, except for the very first phase of

[6] It is easy to see that an accuracy of $\Theta(\log n)$ bits suffices to break ties w.h.p.

the algorithm, the claim does not immediately follow from a standard balls-into-bins argument because not all the destinations of the messages are chosen independently and uniformly at random, as any two messages m_i and $m_{i'}$ of the same component have the same destination.

Let us stipulate that any component part held by machine M_i is the i-th component part of its component, and denote this part with $P_{i,j}$, $i \in [k]$, $j \in [n]$, where $P_{i,j} = \emptyset$ means that in machine i there is no component part for component j. Suppose that the algorithm is in phase j' and iteration ρ. By construction, the hash function $h_{j',\rho}$ is $\tilde{\Theta}(n/k)$-wise independent, and all the component parts held by a single machine are parts of different components. Since M_i has at most $\tilde{\Theta}(n/k)$ distinct component parts w.h.p., it follows that all the proxy machines selected by the component parts held by machine M_i are distributed independently and uniformly at random. Let y be the number of distinct component parts held by a machine M_i that is, $y = |\{P_{i,j} : P_{i,j} \neq \emptyset\}| = \tilde{O}(n/k)$ (w.h.p.).

Consider a link of M_i connecting it to another machine M_1. Let X_t be the indicator variable that takes value 1 if M_1 is the component proxy of part t (of M_i), and let $X_t = 0$ otherwise. Let $X = \sum_{i=1}^{y} X_i$ be the number of component parts that chose their proxy machine at the endpoint of link (M_i, M_1). Since $\Pr(X_i = 1) = 1/(k-1)$, we have that the expected number of messages that have to be sent by this machine over any specific link is $\mathrm{E}[X] = y/(k-1)$.

First, consider the case $y \geq 11k \log n$. As the X_i's are $\tilde{\Theta}(n/k)$-wise independent, all proxies by the component parts of M_i are chosen independently and thus we can apply a standard Chernoff bound (see, e.g., [29]), which gives

$$\Pr\left(X \geq \frac{7y}{4(k-1)}\right) \leq e^{-3y/16(k-1)} < e^{\frac{-2k\log n}{k}} < \frac{1}{n^2}.$$

By applying the union bound over the $k \leq n$ machines we conclude that w.h.p. every machine sends $\tilde{O}(n/k^2)$ messages to each proxy machine, and this requires $\tilde{O}(n/k^2)$ rounds.

Consider now the case $y < 11k \log n$. It holds that $6\mathrm{E}[X] = 6y/(k-1) < 6 \cdot 11k \log n/(k-1) \leq 132 \log n$, and thus, by a standard Chernoff bound,

$$\Pr\left(X \geq 132 \log n\right) \leq 2^{-132 \log n} = \frac{1}{n^{132}}.$$

Analogously to the first case, applying the union bound over the $k \leq n$ machines yields the result. □

2.3 Linear Graph Sketches

As we will see in Section 2.5, our algorithm proceeds by merging components across randomly chosen inter-component edges. In this subsection we show how to provide these sampling capabilities in a communication-efficient way in the k-machine model by implementing random linear graph sketches. Our description follows the notation of [28].

Recall that each vertex u of G is associated with a unique integer ID from $[n]$ (known to its home machine) which, for simplicity, we also denote by u.[7] For each vertex u we define the *incidence vector* $\mathbf{a}_u \in \{-1, 0, 1\}^{\binom{n}{2}}$ *of* u, which describes the incident edges of u, as follows: $\mathbf{a}_u[(x,y)] = 0$ if $(x,y) \notin E(G)$ or, if $(x,y) \in E(G)$, then $\mathbf{a}_u[(x,y)] = 1$ if $u = x < y$, or $a_u[(x,y)] = -1$ if $x < u = y$. Note that the vector $\mathbf{a}_u + \mathbf{a}_v$ corresponds to the incidence vector of the contracted edge (u,v). Intuitively speaking, summing up incidence vectors "zeroes out" edges between the corresponding vertices, hence the vector $\sum_{u \in C} \mathbf{a}_u$ represents the outgoing edges of a component C.

Since each incidence vector $\mathbf{a}_u$ requires polynomial space, it would be inefficient to directly communicate vectors to component proxies. Instead, we construct a random linear sketch $\mathbf{s}_u$ of polylog(n)-size that has the property of allowing us to sample uniformly at random a nonzero entry of $\mathbf{a}_u$ (i.e., an edge incident to u). (This is referred to as ℓ_0-sampling in the streaming literature, see e.g. [28].) It is shown in [16] that ℓ_0-sampling can be performed by linear projections. Therefore, at the beginning of each phase j of our algorithm, we instruct each machine to to create a new (common) polylog$(n) \times \binom{n}{2}$ sketch matrix L_j, which we call *phase j sketch matrix*.[8] Then, each machine M creates a sketch $\mathbf{s}_u = L_j \cdot \mathbf{a}_u$ for each vertex u that resides on M. Hence, each $\mathbf{s}_u$ can be represented by a polylogarithmic number of bits.

Observe that, by linearity, we have $L_j \cdot \mathbf{a}_u + L_j \cdot \mathbf{a}_v = L_j \cdot (\mathbf{a}_u + \mathbf{a}_v)$. In other words, a crucial property of sketches is that the sum $\mathbf{s}_u + \mathbf{s}_v$ is itself a sketch that allows us to sample an edge incident to the contracted edge (u,v). We summarize these properties in the following statement.

LEMMA 2. *Consider a phase j, and let P a subgraph of G induced by vertices $\{u_1, \ldots, u_\ell\}$. Let $\mathbf{s}_{u_1}, \ldots, \mathbf{s}_{u_\ell}$ be the associated sketches of vertices in P constructed by applying the* phase j sketch matrix *to the respective incidence vectors. Then, the combined sketch $\mathbf{s}_P = \sum_{i=1}^{\ell} \mathbf{s}_{u_i}$ can be represented using $O(\mathrm{polylog}(n))$ bits and, by querying $\mathbf{s}_P$, it is possible (w.h.p.) to sample a random edge incident to P (in G) that has its other endpoint in $G \setminus P$.*

Constructing Linear Sketches Without Shared Randomness. Our construction of the linear sketches described so far requires $\tilde{O}(n)$ fully independent random bits that would need to be shared by all machines. It is shown in Theorem 1 (cf. also Corollary 1) of [10] that it is possible to construct such an ℓ_0-sampler (having the same linearity properties) by using $\Theta(n)$ random bits that are only $\Theta(\log n)$-wise independent. Analogously as in Section 2.2, we can generate the required $\Theta(\log^2 n)$ true random bits at machine M_1, distribute them among all other machines in $O(1)$ rounds, and then invoke Theorem 2.1 of [5] at each machine in parallel to generate the required (shared) $\Theta(\log n)$-wise independent random bits for constructing the sketches.

2.4 Outgoing Edge Selection

Now that we know how to construct a sketch of the graph neighborhood of any set of vertices, we will describe how to combine these sketches in a communication-efficient way in the k-machine model. The goal of this step is, for each (current) component C, to find an outgoing edge that connects C to some other component C'.

Recall that C itself might be split into parts $P_1, P_2, \ldots, P_j$ across multiple machines. Therefore, as a first step, each machine M_i locally constructs the combined sketch for each part that resides in M_i. By Lemma 2, the resulting sketches

[7] Note that the asymptotics of our results do not change if the size of the ID space is $O(\mathrm{poly}(n))$.

[8] Here we describe the construction as if nodes have access to a source of shared randomness (to create the sketch matrix). We later show how to remove this assumption.

have polylogarithmic size each and present a sketch of the incidences of their respective component parts. Next, we combine the sketches of the individual parts of each component C to a sketch of C, by instructing the machines to send the sketch of each part P_i (of component C) to the proxy machine of C. By virtue of Lemma 1, all of these messages are delivered to the component proxies within $\tilde{O}(n/k^2)$ rounds. Finally, the component proxy machine of C combines the received sketches to yield a sketch of C, and randomly samples an outgoing edge of C (see Lemma 2). Thus, at the end of this procedure, every component (randomly) selected exactly one neighboring component. We now show that the complexity of this procedure is $\tilde{O}(n/k^2)$ w.h.p.

LEMMA 3. *Every component can select exactly one outgoing edge in $\tilde{O}(n/k^2)$ rounds with high probability.*

PROOF. Clearly, since at every moment each node has a unique component's label, each machine holds $\tilde{O}(n/k)$ component's parts w.h.p. Each of these parts selected at most one edge, and thus each machine "selected" $\tilde{O}(n/k)$ edges w.h.p. All these edges have to be sent to the corresponding proxy. By Lemma 1, this requires $\tilde{O}(n/k^2)$ rounds.

The procedure is completed when the proxies communicate the decision to each of the at most k components' parts. This entails as many messages as in the first part to be routed using exactly the same machines' links used in the first part, with the only difference being that messages now travel in the opposite direction. The lemma follows. □

2.5 Merging of Components

After the proxy machine of each component C has selected one edge connecting C to a different component, all the neighboring components are merged so as to become a new, bigger component. This is accomplished by relabeling the nodes of the graph such that all the nodes in the same (new) component have the same label. Notice that the merging is thus only virtual, that is, component parts that compose a new component are not moved to a common machine; rather, nodes (and their incident edges) remain in their home machine, and just get (possibly) assigned a new label.

We can think of the components along with the sampled outgoing edges as a *component graph* $\mathcal{C}$. We use the *distributed random ranking* (DRR) technique [8] to avoid having long chains of components (i.e., long paths in $\mathcal{C}$). That is, we will (conceptually) construct a forest of directed trees that is a subgraph (modulo edge directions) of the component graph $\mathcal{C}$ and where each tree has depth $O(\log n)$.[9] The component proxy of each component C chooses a rank independently and uniformly at random from $[0, 1]$. (It is easy to show that $\Theta(\log n)$ bits provide sufficient accuracy to break ties w.h.p.) Now, the proxy machine of C (virtually) connects C to its neighboring component C' if and only if the rank chosen by the latter's proxy is higher. In this case, we say that C' becomes the *parent* of C and C is a *child* of C'.

LEMMA 4. *After $\tilde{O}(n/k^2)$ rounds, the structure of the DRR-tree is completed with high probability.*

[9]Instead of using DRR trees, an alternate and simpler idea is the following. Let every component select a number in $[0, 1]$. A merging can be done only if the outgoing edge (obtained from the sketch) connects a component with ID 0 to a component with ID 1. One can show that this merging procedure also gives the same time bound.

PROOF. We need to show that every proxy machine of a non-root component knows its smaller-ranking parent component and every root proxy machine knows that it is root. Note that during this step the proxy machines of the child components communicate with the respective parent proxy machines. Moreover, the number of messages sent for determining the ordering of the DRR-trees is guaranteed to be $O(n)$ with high probability, since $\mathcal{C}$ has only $O(n)$ edges. By instantiating Lemma 1, it follows that the delivery of these messages can be completed in $\tilde{O}(n/k^2)$ rounds w.h.p.

Since links are bidirectional, the parent proxies are able to send their replies within the same number of rounds, by re-running the message schedule of the child-to-parent communication in reverse order. □

If a component has the highest rank among all its neighbors (in $\mathcal{C}$), we call it a *root component*. Since every component except root components connects to a component with higher rank, the resulting structure is a set of disjoint rooted trees.

In the next step, we will merge all components of each tree into a single new component such that all vertices that are part of some component in this tree receive the label of the root. Consider a tree $\mathcal{T}$. We proceed level-wise (in parallel for all trees) and start the merging of components at the leafs that are connected to a (lower-ranking) parent component C.

LEMMA 5. *There is a distributed algorithm that merges all trees of the DRR forest in $\tilde{O}(dn/k^2)$ rounds with high probability, where d is the largest depth of any tree.*

PROOF. We proceed in d iterations by merging the (current) leaf components with their parents in the tree. Thus it is sufficient to analyze the time complexity of a single iteration. To this end, we describe a procedure that changes the component labels of all vertices that are in leaf components in the DRR forest to the label of the respective parent in $\tilde{O}(n/k^2)$ rounds.

At the beginning of each iteration, we select a new proxy for each component C by querying the shared hash function $h_{j,\rho}(C)$, where ρ is the current iteration number. This ensures that there are no dependencies between the proxies used in each iteration. We know from Lemma 4 that there is a message schedule such that leaf proxies can communicate with their respective parent proxy in $\tilde{O}(n/k^2)$ rounds (w.h.p.) and vice versa, and thus every leaf proxy knows the component label of its parent. We have already shown in Lemma 3 that we can deliver a message from each component part to its respective proxy (when combining the sketches) in $\tilde{O}(n/k^2)$ rounds. Hence, by re-running this message schedule, we can broadcast the parent label from the leaf proxy to each component part in the same time. Each machine that receives the parent label locally changes the component label of the vertices that are in the corresponding part. □

The following result is proved in [8, Theorem 11].

LEMMA 6. *The depth of each DRR tree is $O(\log n)$ with high probability.*

2.6 Analysis of the Time Complexity

We now show that the number of phases required by the algorithm to determine the connected components of the input graph is $O(\log n)$. At the beginning of each phase

i, distributed across the k machines there are c_i distinct components. At the beginning of the algorithm each node is identified as a component, and thus $c_0 = n$. The algorithm ends at the completion of phase φ, where φ is the smallest integer such that $c_\varphi = cc(G)$, where $cc(G)$ denotes the number of connected components of the input graph G. If pairs of components were merged in each phase, it would be straightforward to show that the process would terminate in at most $O(\log n)$ phases. However, in our algorithm each component connects to its neighboring component if and only if the latter has a higher rank. Nevertheless, it is not difficult to show that this slightly different process also terminates in $O(\log n)$ phases w.h.p. (that is, components gets merged "often enough"). The intuition for this result is that, since components' ranks are taken randomly, for each component the probability that its neighboring component has a higher rank is exactly one half. Hence, on average half of the components will not be merged with their own neighbor: each of these components thus becomes a root of one component, which means that, on average, the number of new components will be half as well.

LEMMA 7. *After $12 \log n$ phases, the component labels of the vertices correspond to the connected components of G with high probability.*

PROOF. Replace the c_i's with corresponding random variables C_i's, and consider the stochastic process defined by the sequence $C_0, C_1, \ldots, C_\varphi$. Let $\bar{C}_i$ be the random variable that counts the number of components that actually participate at the merging process of phase i, because they do have an outgoing edge to another component. Call these components *participating components*. Clearly, by definition, $\bar{C}_i \leq C_i$.

We now show that, for every phase $i \in [\varphi - 1]$, $\mathrm{E}[\mathrm{E}[\bar{C}_{i+1} \mid \bar{C}_i]] \leq \mathrm{E}[\bar{C}_i]/2$. To this end, fix a generic phase i and a random ordering of its $\bar{C}_i$ participating components. Define random variables $X_{i,1}, X_{i,2}, \ldots, X_{i,\bar{C}_i}$ where $X_{i,j}$ takes value 1 if the j-th participating component will be a root of a participating tree/component for phase $i+1$, and 0 otherwise. Then, $\bar{C}_{i+1} \mid \bar{C}_i = \sum_{j=1}^{\bar{C}_i} X_{i,j}$ is the number of participating components for phase $i+1$. As we noticed before, for any $i \in [\varphi - 1]$ and $j \in [\bar{C}_i]$, the probability that a participating component will not be merged to its neighboring component, and thus become a root of a tree/component for phase $i+1$ is exactly one half. Therefore, $\Pr(X_{i,j} = 1) \leq 1/2$. Hence, by the linearity of expectation, we have that

$$\mathrm{E}[\bar{C}_{i+1} \mid \bar{C}_i] = \sum_{j=1}^{\bar{C}_i} \mathrm{E}[X_{i,j}] = \sum_{j=1}^{\bar{C}_i} \Pr(X_{i,j} = 1) \leq \frac{\bar{C}_i}{2}.$$

Then, using again the linearity of expectation,

$$\mathrm{E}[\mathrm{E}[\bar{C}_{i+1} \mid \bar{C}_i]] \leq \mathrm{E}\left[\frac{\bar{C}_i}{2}\right] = \frac{\mathrm{E}[\bar{C}_i]}{2}.$$

We now leverage this result to prove the claimed statement. Let us call a phase *successful* if it reduces the number of participating components by a factor of at most 3/4. By Markov's inequality, the probability that phase i is not successful is

$$\Pr\left(\mathrm{E}[\bar{C}_{i+1} \mid \bar{C}_i] > \frac{3}{4}\mathrm{E}[\bar{C}_i]\right) < \frac{\mathrm{E}[\mathrm{E}[\bar{C}_{i+1} \mid \bar{C}_i]]}{(3/4)\mathrm{E}[\bar{C}_i]} \leq \frac{\mathrm{E}[\bar{C}_i]}{2} \cdot \frac{4}{3\mathrm{E}[\bar{C}_i]} = \frac{2}{3},$$

and thus the probability that a phase of the algorithm is successful is at least 1/3. Now consider a sequence of $12 \log n$ phases of the algorithm. We shall prove that within that many phases the algorithm w.h.p. has reduced the number of participating components a sufficient number of times so that the algorithm has terminated, that is, $\varphi \leq 12 \log n$ w.h.p. Let X_i be an indicator variable that takes value 1 if phase i is successful, and 0 otherwise (this also includes the case that the i-th phase does not take place because the algorithm already terminated). Let $X = \sum_{i=1}^{12 \log n} X_i$ be the number of successful phases out of the at most $12 \log n$ phases of the algorithm. Since $\Pr(X_i = 1) \geq 1/3$, by the linearity of expectation we have that

$$\mathrm{E}[X] = \sum_{i=1}^{12 \log n} \mathrm{E}[X_i] = \sum_{i=1}^{12 \log n} \Pr(X_i = 1) \geq \frac{12 \log n}{3} = 4 \log n.$$

As the X_i's are independent we can apply a standard Chernoff bound, which gives

$$\Pr(X \leq \log n) \leq e^{-4 \log n (3/4)^2/2} = e^{-\frac{9}{8} \log n} < \frac{1}{n}.$$

Hence, with high probability $12 \log n$ phases are enough to determine all the components of the input graph. □

THEOREM 1. *There is a distributed algorithm in the k-machine model that determines the connected components of a graph G in $\tilde{O}(n/k^2)$ rounds with high probability.*

PROOF. By Lemma 7, the algorithm finishes in $O(\log n)$ phases with high probability. To analyze the time complexity of an individual phase, recall that it takes $\tilde{O}(n/k^2)$ rounds to sample an outgoing edge (see Lemma 3). Then, building the DRR forest requires $\tilde{O}(n/k^2)$ additional rounds, according to Lemma 4. Merging each DRR tree $\mathcal{T}$ in a level-wise fashion (in parallel) takes $\tilde{O}(dn/k^2)$ rounds (see Lemma 5), where d is the depth of $\mathcal{T}$ which, by virtue of Lemma 6, is bounded by $O(\log n)$. Since each of these time bounds hold with high probability, and the algorithm consists of $O(\log n)$ phases with high probability, by the union bound we conclude that the total time complexity of the algorithm is $\tilde{O}(n/k^2)$ with high probability. □

We conclude the section by noticing that it is easy to output the actual number of connected components after the termination of our algorithm: every machine just needs to send "YES" directly to the proxies of each of the components' labels it holds, and subsequently such proxies will send the labels of the components for which they received "YES" to one predetermined machine. Since the communication is performed via the components' proxies, it follows from Lemma 1 that the first step takes $\tilde{O}(n/k^2)$ rounds w.h.p., and the second step takes only $O(\log n)$ rounds w.h.p.

3. APPLICATIONS

In this section we describe how to use our fast connectivity algorithm as a building block to solve several other fundamental problems in the k-machine model in time $\tilde{O}(n/k^2)$. Here we consider the problem of constructing a minimum spanning tree (MST), and defer the treatment of other problems (namely, approximate min-cut and several graph verification problems) to the full version of the paper [32].

The minimum spanning tree (MST) problem requires the machines to jointly output a set of edges that form a tree,

connect all nodes, and have the minimum possible total weight. Interestingly, [19] shows that $\tilde{\Omega}(n/k)$ rounds are necessary for constructing *any* spanning tree (ST), assuming that, for every spanning tree edge (u, v), the home machine of u and the home machine of v must *both* eventually output (u, v) as being part of the ST. We now show that we can break the $\tilde{\Omega}(n/k)$ barrier, under the slightly less stringent requirement that at least *one* machine outputs each spanning tree edge e, but not necessarily any of the home machines of e. Note that for the MST problem we assume that each edge $e = (u, v)$ of the input graph G has an associated *edge weight* $w(e)$ that is known to the home machines of u and v.

Our algorithm mimics the multi-pass MST construction procedure of [2], originally devised for the (centralized) streaming model. To this end we modify our connectivity procedure of Section 2, by ensuring that when a component proxy C chooses an outgoing edge e, this is the *minimum weight outgoing edge* (MWOE) of C with high probability.

We now describe the i-the phase of this MST construction in more detail: Analogously to our algorithm in Section 2, the proxy of each component C determines an outgoing edge e_0 which, by the guarantees of our sketch construction (Lemma 2), is chosen uniformly at random from all possible outgoing edges of C.

We then repeat the following edge-elimination process $t = \Theta(\log n)$ times: The proxy broadcasts $w(e_0)$ to every component part of C. Recall from Lemma 3 that this communication is possible in $\tilde{O}(n/k^2)$ rounds. Upon receiving this message, the machine M of a part P of C now constructs a new sketch $\mathbf{s}_u$ for each $u \in P$, but first zeroes out all entries in $\mathbf{a}_u$ that refer to edges of weight $> w(e_0)$. (See Section 2.3 for a more detailed description of $\mathbf{a}_u$ and $\mathbf{s}_u$.) Again, we combine the sketches of all vertices of all parts of C at the proxy of C, which in turn samples a new outgoing edge e_1 for C. Since each time we sample a randomly chosen edge and eliminate all higher weight edges, it is easy to see that the edge e_t is the MWOE of C w.h.p. Thus, the proxy machine of C includes the edge e_t as part of the MST output. Note that this additional elimination procedure incurs only a logarithmic time complexity overhead.

At the end of each phase, we proceed by (virtually) merging the components along their MWOEs in a similar manner as for the connectivity algorithm (see Section 2.5), thus requiring $\tilde{O}(n/k^2)$ rounds in total.

Let E be the set of added outgoing edges. Since the components of the connectivity algorithm eventually match the actual components of the input graph, the graph H on the vertices $V(G)$ induced by E connects all vertices of G. Moreover, since components are merged according to the trees of the DRR-process (see Section 2.5), it follows that H is cycle-free.

We can now fully classify the complexity of the MST problem in the k-machine model:

THEOREM 2. *There exists an algorithm for the k-machine model that outputs an MST in*

- *$\tilde{O}(n/k^2)$ rounds, if each MST-edge is output by at least one machine, or in*
- *$\tilde{O}(n/k)$ rounds, if each MST-edge e is output by both machines that hold an endpoint of e.*

Both bounds are tight up to polylogarithmic factors.

4. LOWER BOUNDS FOR VERIFICATION PROBLEMS

In this section we show that $\tilde{\Omega}(n/k^2)$ rounds is a fundamental lower bound for many graph verification problems in the k-machine model. Even though many verification problems are known to satisfy a lower bound of $\tilde{\Omega}(D + \sqrt{n})$ in the classic distributed $\mathcal{CONGEST}$ model [11], the reduction of [11] encodes a $\Theta(\sqrt{n})$-instance of set disjointness, requiring at least one node to receive $\tilde{\Theta}(\sqrt{n})$ information across a single short "highway" path or via $\Theta(\sqrt{n})$ longer paths of length $\Theta(\sqrt{n})$. Moreover, we assume the random vertex partition model, whereas the results of [11] assume a worst case distribution. Lastly, any pair of machines can communicate directly in the k-machine model, thus breaking the $\Omega(D)$ bound for the $\mathcal{CONGEST}$ model. Our complexity bounds follows from the communication complexity of 2-player set disjointness in the *random input partition model* (see [19]). The main result of the section is the following theorem. The proof is deferred to the full version of the paper [32].

THEOREM 3. *There exists a constant $\gamma > 0$ such that any γ-error algorithm $\mathcal{A}$ has round complexity of $\tilde{\Omega}(n/k^2)$ on an n-node vertex graph of diameter 2 in the k-machine model, if $\mathcal{A}$ solves any of the following problems: connectivity [19], spanning connected subgraph, cycle containment, e-cycle containment, s-t-connectivity, cut, edge on all paths, and s-t-cut.*

5. CONCLUSIONS

There are several natural directions for future work. Our connectivity algorithm is randomized: it would be interesting to study the deterministic complexity of graph connectivity in the k-machine model. Specifically, does graph connectivity admit a $\tilde{O}(n/k^2)$ deterministic algorithm? Investigating higher-order connectivity, such as 2-edge/vertex connectivity, is also an interesting research direction. A general question motivated by the algorithms presented in this paper is whether one can design algorithms that have superlinear scaling in k for other fundamental graph problems. Some recent results in this directions are in [33].

Acknowledgments. The authors would like to thank Mohsen Ghaffari, Seth Gilbert, Andrew McGregor, Danupon Nanongkai, and Sriram V. Pemmaraju for helpful discussions, and the anonymous reviewers for their helpful comments.

6. REFERENCES

[1] Giraph, http://giraph.apache.org/.
[2] K. J. Ahn, S. Guha, and A. McGregor. Analyzing graph structure via linear measurements. In *ACM-SIAM SODA*, pages 459–467, 2012.
[3] K. J. Ahn, S. Guha, and A. McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *ACM PODS*, pages 5–14, 2012.
[4] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, 1986.
[5] N. Alon, R. Rubinfeld, S. Vardi, and N. Xie. Space-efficient local computation algorithms. In *ACM-SIAM SODA*, pages 1132–1139, 2012.

[6] O. Boruvka. O Jistém Problému Minimálním (About a Certain Minimal Problem). *Práce Mor. Prírodoved. Spol. v Brne III*, 3, 1926.

[7] K. Censor-Hillel, P. Kaski, J. H. Korhonen, C. Lenzen, A. Paz, and J. Suomela. Algebraic methods in the congested clique. In *PODC*, pages 143–152, 2015.

[8] J. Chen and G. Pandurangan. Almost-optimal gossip-based aggregate computation. *SIAM J. Comput.*, 41(3):455–483, 2012.

[9] F. Chung and O. Simpson. Distributed algorithms for finding local clusters using heat kernel pagerank. In *WAW*, pages 77–189, 2015.

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

[11] A. Das Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM J. Comput.*, 41(5):1235–1265, 2012.

[12] A. Drucker, F. Kuhn, and R. Oshman. On the power of the congested clique model. In *PODC*, pages 367–376, 2014.

[13] M. Elkin, H. Klauck, D. Nanongkai, and G. Pandurangan. Can quantum communication speed up distributed computation? In *ACM PODC*, pages 166–175, 2014.

[14] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.

[15] J. W. Hegeman, G. Pandurangan, S. V. Pemmaraju, V. B. Sardeshmukh, and M. Scquizzato. Toward optimal bounds in the congested clique: Graph connectivity and MST. In *PODC*, pages 91–100, 2015.

[16] H. Jowhari, M. Saglam, and G. Tardos. Tight bounds for L_p samplers, finding duplicates in streams, and related problems. In *ACM PODS*, pages 49–58, 2011.

[17] 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, 1995.

[18] H. J. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *ACM-SIAM SODA*, pages 938–948, 2010.

[19] H. Klauck, D. Nanongkai, G. Pandurangan, and P. Robinson. Distributed computation of large-scale graph problems. In *ACM-SIAM SODA*, pages 391–410, 2015.

[20] S. Kutten, G. Pandurangan, D. Peleg, P. Robinson, and A. Trehan. Sublinear bounds for randomized leader election. *Theoret. Comput. Sci.*, 561:134–143, 2015.

[21] S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii. Filtering: a method for solving graph problems in MapReduce. In *ACM SPAA*, pages 85–94, 2011.

[22] C. Lenzen. Optimal deterministic routing and sorting on the congested clique. In *PODC*, pages 42–50, 2013.

[23] C. Lenzen and R. Wattenhofer. Tight bounds for parallel randomized load balancing. *Distrib. Comput.*, 29(2):127–142, 2016.

[24] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2014.

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

[26] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., 1996.

[27] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *ACM SIGMOD*, pages 135–146, 2010.

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

[29] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[30] D. Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *STOC*, pages 565–573, 2014.

[31] D. Nanongkai, A. D. Sarma, and G. Pandurangan. A tight unconditional lower bound on distributed randomwalk computation. In *ACM PODC*, pages 257–266, 2011.

[32] G. Pandurangan, P. Robinson, and M. Scquizzato. Fast distributed algorithms for connectivity and MST in large graphs. *CoRR*, abs/1503.02353, 2016.

[33] G. Pandurangan, P. Robinson, and M. Scquizzato. Tight bounds for distributed graph computations. *CoRR*, abs/1602.08481, 2016.

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

[35] J. Qiu, S. Jha, A. Luckow, and G. C. Fox. Towards HPC-ABDS: An initial high-performance big data stack. 2014. Available: http://grids.ucs.indiana.edu/ptliupages/publications/nist-hpc-abds.pdf.

[36] I. Stanton. Streaming balanced graph partitioning algorithms for random graphs. In *ACM-SIAM SODA*, pages 1287–1301, 2014.

[37] R. Thurimella. Sub-linear distributed algorithms for sparse certificates and biconnected components. *J. Algorithms*, 23(1):160–179, 1997.

[38] Y. Tian, A. Balmin, S. A. Corsten, S. Tatikonda, and J. McPherson. From "think like a vertex" to "think like a graph". *PVLDB*, 7(3):193–204, 2013.

[39] L. G. Valiant. A scheme for fast parallel communication. *SIAM J. Comput.*, 11(2):350–361, 1982.

[40] L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, 1990.

[41] S. Vassilvitskii. Models for parallel computation (a hitchhikers' guide to massively parallel universes), http://grigory.us/blog/massively-parallel-universes/, 2015.

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