Distributed Verification and Hardness of Distributed Approximation*

Atish Das Sarma[†‡] Stephan Holzer[§] Liah Kor[¶] Amos Korman[‖**]
Danupon Nanongkai[††‡‡] Gopal Pandurangan[‡‡] David Peleg[¶ **] Roger Wattenhofer[§]

October 18, 2011

Abstract

We study the *verification* problem in distributed networks, stated as follows. Let H be a subgraph of a network G where each vertex of G knows which edges incident on it are in H. We would like to verify whether H has some properties, e.g., if it is a tree or if it is connected (every node knows at the end of the process whether H has the specified property or not). We would like to perform this verification in a decentralized fashion via a distributed algorithm. The time complexity of verification is measured as the number of rounds of distributed communication.

In this paper we initiate a systematic study of distributed verification, and give almost tight lower bounds on the running time of distributed verification algorithms for many fundamental problems such as connectivity, spanning connected subgraph, and $s-t$ cut verification. We then show applications of these results in deriving strong unconditional time lower bounds on the *hardness of distributed approximation* for many classical optimization problems including minimum spanning tree, shortest paths, and minimum cut. Many of these results are the first non-trivial lower bounds for both exact and approximate distributed computation and they resolve previous open questions. Moreover, our unconditional lower bound of approximating minimum spanning tree (MST) subsumes and improves upon the previous hardness of approximation bound of Elkin [STOC 2004] as well as the lower bound for (exact) MST computation of Peleg and Rubinovich [FOCS 1999]. Our result implies that there can be no distributed

*The preliminary version of this paper appeared as [5] in the Proceeding of the 43rd ACM Symposium on Theory of Computing, (STOC) 2011.

[†]Google Research, Google Inc., Mountain View, USA. E-mail: dassarma@google.com.

[‡]Part of the work done while at Georgia Institute of Technology.

[§]Computer Engineering and Networks Laboratory (TIK), ETH Zurich, CH-8092 Zurich, Switzerland. E-mail: {stholzer, wattenhofer}@tik.ee.ethz.ch.

[¶]Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, 76100 Israel. E-mail: {liah.kor,david.peleg}@weizmann.ac.il. Supported by a grant from the United States-Israel Binational Science Foundation (BSF).

[‖]CNRS and LIAFA, Univ. Paris 7, Paris, France. E-mail: amos.korman@liafa.jussieu.fr. Supported by the ANR projects ALADDIN and PROSE and by the INRIA project GANG.

[**]Supported by a France-Israel cooperation grant ("Mutli-Computing" project) from the France Ministry of Science and Israel Ministry of Science.

[††]Faculty of Computer Science, The University of Vienna, Vienna, Austria. E-mail: danupon@gmail.com

[‡‡]Division of Mathematical Sciences, Nanyang Technological University, Singapore 637371 & Department of Computer Science, Brown University, Providence, RI 02912, USA. E-mail: gopalpandurangan@gmail.com. Supported in part by the following grants: Nanyang Technological University grant M58110000, US NSF grant CCF-1023166, and a grant from the US-Israeli Binational Science Foundation (BSF).

approximation algorithm for MST that is significantly faster than the current exact algorithm, for *any* approximation factor.

Our lower bound proofs show an interesting connection between communication complexity and distributed computing which turns out to be useful in establishing the time complexity of exact and approximate distributed computation of many problems.

1 Introduction

Large and complex networks, such as the human society, the Internet, or the brain, are being studied intensely by different branches of science. Each individual node in such a network can directly communicate only with its neighboring nodes. Despite being restricted to such *local* communication, the network itself should work towards a *global* goal, i.e., it should organize itself, or deliver a service.

In this work we investigate the possibilities and limitations of distributed/decen-tralized computation, i.e., to what degree local information is sufficient to solve global tasks. Many tasks can be solved entirely via local communication, for instance, how many friends of friends one has. Research in the last 30 years has shown that some classic combinatorial optimization problems such as matching, coloring, dominating set, or approximations thereof can be solved using small (i.e., polylogarithmic) local communication. For example, a maximal independent set can be computed in time $O(\log n)$ [25], but not in time $\Omega(\sqrt{\log n / \log\log n})$ [18] (n is the network size). This lower bound even holds if message sizes are unbounded.

However many important optimization problems are "global" problems from the distributed computation point of view. To count the total number of nodes, to determining the diameter of the system, or to compute a spanning tree, information necessarily must travel to the farthest nodes in a system. If exchanging a message over a single edge costs one time unit, one needs $\Omega(D)$ time units to compute the result, where D is the network diameter. If message size was unbounded, one can simply collect all the information in $O(D)$ time, and then compute the result. Hence, in order to arrive at a realistic problem, we need to introduce communication limits, i.e., each node can exchange messages with each of its neighbors in each step of a synchronous system, but each message can have at most B bits (typically B is small, say $O(\log n)$). However, to compute a spanning tree, even single-bit messages are enough, as one can simply breadth-first-search the graph in time $O(D)$ and this is optimal [28].

But, can we *verify* whether an existing subgraph that is claimed to be a spanning tree indeed is a correct spanning tree?! In this paper we show that this is not generally possible in $O(D)$ time – instead one needs $\Omega(\sqrt{n} + D)$ time. (Thus, in contrast to traditional non-distributed complexity, verification is harder than computation in the distributed world!). Our paper is more general, as we show interesting lower and upper bounds (these are almost tight) for a whole selection of verification problems. Furthermore, we show a key application of studying such verification problems to proving strong unconditional time lower bounds on exact and approximate distributed computation for many classical problems.

1.1 Technical Background and Previous Work

Distributed Computing Consider a synchronous network of processors with unbounded computational power. The network is modeled by an undirected n-vertex graph, where vertices model the processors and edges model the links between the processors. The processors (henceforth, vertices) communicate by exchanging messages via the links (henceforth, edges). The vertices have

limited global knowledge, in particular, each of them has its own local perspective of the network (a.k.a graph), which is confined to its immediate neighborhood. The vertices may have to compute (cooperatively) some global function of the graph, such as a spanning tree (ST) or a minimum spanning tree (MST), via communicating with each other and running a distributed algorithm designed for the task at hand. There are several measures to analyze the performance of such algorithms, a fundamental one being the running time, defined as the worst-case number of *rounds* of distributed communication. This measure naturally gives rise to a complexity measure of problems, called the *time complexity*. On each round at most B bits can be sent through each edge in each direction, where B is the bandwidth parameter of the network. The design of efficient algorithms for this model (henceforth, the B model), as well as establishing lower bounds on the time complexity of various fundamental graph problems, has been the subject of an active area of research called (locality-sensitive) *distributed computing* (see [28] and references therein.)

Distributed Algorithms, Approximation, and Hardness Much of the initial research focus in the area of distributed computing was on designing algorithms for solving problems exactly, e.g., distributed algorithms for ST, MST, and shortest paths are well-known [28, 26]. Over the last few years, there has been interest in designing distributed algorithms that provide approximate solutions to problems. This area is known as *distributed approximation*. One motivation for designing such algorithms is that they can run faster or have better communication complexity albeit at the cost of providing suboptimal solution. This can be especially appealing for resource-constrained and dynamic networks (such as sensor or peer-to-peer networks). For example, there is not much point in having an optimal algorithm in a dynamic network if it takes too much time, since the topology could have changed by that time. For this reason, in the distributed context, such algorithms are well-motivated even for network optimization problems that are not NP-hard, e.g., minimum spanning tree, shortest paths etc. There is a large body of work on distributed approximation algorithms for various classical graph optimization problems (e.g., see the surveys by Elkin [7] and Dubhashi et al. [6], and the work of [15] and the references therein).

While a lot of progress has been made in the design of distributed approximation algorithms, the same has not been the case with the theory of lower bounds on the approximability of distributed problems, i.e., *hardness* of distributed approximation. There are some inapproximability results that are based on lower bounds on the time complexity of the exact solution of certain problems and on integrality of the objective functions of these problems. For example, a fundamental result due to Linial [23] says that 3-coloring an n-vertex ring requires $\Omega(\log^* n)$ time. In particular, it implies that any 3/2-approximation protocol for the vertex-coloring problem requires $\Omega(\log^* n)$ time. On the other hand, one can state inapproximability results assuming that vertices are computationally limited; under this assumption, any NP-hardness inapproximability result immediately implies an analogous result in the distributed model. However, the above results are not interesting in the distributed setting, as they provide no new insights on the roles of locality and communication [10].

There are but a few significant results currently known on the hardness of distributed approximation. Perhaps the first important result was presented for the MST problem by Elkin in [10]. Specifically, he showed strong *unconditional* lower bounds (i.e., ones that do not depend on complexity-theoretic assumptions) for distributed approximate MST (more on this result below). Later, Kuhn, Moscibroda, and Wattenhofer [18] showed lower bounds on time approximation trade-offs for several problems.

1.2 Distributed Verification

The above discussion summarized two major research aspects in distributed computing, namely studying distributed algorithms and lower bounds for (1) exact and (2) approximate solutions to various problems. The third aspect — that turns out to have remarkable applications to the first two — called *distributed verification*, is the main subject of the current paper. In distributed verification, we want to efficiently check whether a given subgraph of a network has a specified property via a distributed algorithm[1]. Formally, given a graph $G = (V, E)$, a subgraph $H = (V, E')$ with $E' \subseteq E$, and a predicate Π, it is required to decide whether H satisfies Π (i.e., when the algorithm terminates, every node knows whether H satisfies Π). The predicate Π may specify statements such as "H is connected" or "H is a spanning tree" or "H contains a cycle". (Each vertex in G knows which of its incident edges (if any) belong to H.) The goal is to study bounds on the time complexity of distributed verification. The time complexity of the verification algorithm is measured with respect to parameters of G (in particular, its size n and diameter D), independently from H.

We note that verification is different from construction problems, which have been the traditional focus in distributed computing. Indeed, distributed algorithms for constructing spanning trees, shortest paths, and other problems have been well studied ([28, 26]). However, the corresponding verification problems have received much less attention. To the best of our knowledge, the only distributed verification problem that has received some attention is the MST (i.e., verifying if H is a MST); the recent work of Kor et al. [16] gives a $\Omega(\sqrt{n}/B + D)$ deterministic lower bound on distributed verification of MST, where D is the diameter of the network G. That paper also gives a matching upper bound (see also [17]). Note that distributed *construction* of MST has rather similar lower and upper bounds [29, 12]. Thus in the case of the MST problem, verification and construction have the same time complexity. We later show that the above result of Kor et al. is subsumed by the results of this paper, as we show that verifying *any* spanning tree takes so much time.

Motivations The study of distributed verification has two main motivations. The first is understanding the complexity of verification versus construction. This is obviously a central question in the traditional RAM model, but here we want to focus on the same question in the distributed model. Unlike in the centralized setting, it turns out that verification is *not* in general easier than construction in the distributed setting! In fact, as was indicated earlier, distributively verifying a spanning tree turns out to be harder than constructing it in the worst case. Thus understanding the complexity of verification in the distributed model is also important. Second, from an algorithmic point of view, for some problems, understanding the verification problem can help in solving the construction problem or showing the inherent limitations in obtaining an efficient algorithm. In addition to these, there is yet another motivation that emerges from this work: We show that distributed verification leads to showing *strong unconditional lower bounds on distributed computation (both exact and approximate)* for a variety of problems, many hitherto unknown. For example, we show that establishing a lower bound on the spanning connected subgraph verification problem leads to establishing lower bounds for the minimum spanning tree, shortest path tree, minimum cut etc. Hence, studying verification problems may lead to proving hardness of approximation as well as lower bounds for exact computation for new problems.

[1]Such problems have been studied in the sequential setting, e.g., Tarjan [32] studied verification of MST.

1.3 Our Contributions

In this paper, our main contributions are twofold. First, we initiate a systematic study of *distributed verification*, and give almost tight uniform lower bounds on the running time of distributed verification algorithms for many fundamental problems. Second, we make progress in establishing strong hardness results on the distributed approximation of many classical optimization problems. Our lower bounds also apply seamlessly to exact algorithms. We next state our main results (the precise theorem statements are in the respective sections as mentioned below).

1. Distributed Verification We show a lower bound of $\Omega(\sqrt{n/(B\log n)} + D)$ for many verification problems in the B model, including *spanning connected subgraph*, *s-t connectivity*, *cycle-containment*, *bipartiteness*, *cut*, *least-element list*, and $s-t$ *cut* (cf. definitions in Section 5). These bounds apply to *Monte Carlo randomized* algorithms as well and clearly hold also for asynchronous networks. Moreover, it is important to note that our lower bounds apply even to graphs of small diameter ($D = O(\log n)$). Furthermore we present slightly weaker lower bounds for even smaller (constant) diameters. (Indeed, the problems studied in this paper are "global" problems, i.e., the network diameter of G imposes an inherent lower bound on the time complexity.)

Additionally, we show that another fundamental problem, namely, the spanning tree verification problem (i.e., verifying whether H is a spanning tree) has the same lower bound of $\Omega(\sqrt{n/(B\log n)}+ D)$ (cf. Section 6). However, this bound applies to only deterministic algorithms. This result strengthens the deterministic lower bound result of minimum spanning tree verification by Kor et al. [16] in that it shows that the same lower bound holds even on the simpler problem of spanning tree verification. Moreover, we note the interesting fact that although finding a spanning tree (e.g., a breadth-first tree) can be done in $O(D)$ rounds [28], verifying if a given subgraph is a spanning tree requires $\tilde{\Omega}(\sqrt{n} + D)$ rounds! Thus the verification problem for spanning trees is harder than its construction in the distributed setting. This is in contrast to this well-studied problem in the centralized setting. Apart from the spanning tree verification problem, we also show deterministic lower bounds for other verification problems, including *Hamiltonian cycle* and *simple path* verification.

Our lower bounds are almost tight as we show that there exist algorithms that run in $O(\sqrt{n}\log^* n+ D)$ rounds (assuming $B = O(\log n)$) for almost all the verification problems addressed here (cf. Section 8).

2. Bounds on Hardness of Distributed Approximation An important consequence of our verification lower bound is that it leads to lower bounds for exact and approximate distributed computation. We show the unconditional time lower bound of $\Omega(\sqrt{n/(B\log n)} + D)$ for approximating many optimization problems, including *MST, shortest $s-t$ path, shortest path tree*, and *minimum cut* (Section 7). The important point to note is that the above lower bound applies for *any* approximation ratio $\alpha \geq 1$. Thus the same bound holds for exact algorithms as well (that is $\alpha = 1$). All these hardness bounds hold for randomized algorithms. (In fact, these bounds hold for *Monte Carlo* randomized algorithms while previous lower bounds [29, 10, 24] hold only for *Las Vegas* randomized algorithms.) As in our verification lower bounds, these bounds apply even to graphs of small ($O(\log n)$) diameter. Figure 1 summarizes our lower bounds for various diameters.

Our results improve over previous ones (e.g., Elkin's lower bound for approximate MST and shortest path tree [10]) and subsumes some well-established exact bounds (e.g., Peleg and Rubinovich lower bound for MST [29]) as well as show new strong bounds (both for exact and approxi-

Diameter D	Previous lower bound for MST- and shortest-path tree-approx. [10] (for exact algorithms, use $\alpha = 1$)	New lower bound for MST-, shortest-path tree-approx. and all problems in Fig. 2.
$n^\delta,\ 0 < \delta < 1/2$	$\Omega\left(\sqrt{\frac{n}{\alpha B}}\right)$	$\Omega\left(\sqrt{\frac{n}{B}}\right)$
$\Theta(\log n)$	$\Omega\left(\sqrt{\frac{n}{\alpha B \log n}}\right)$	$\Omega\left(\sqrt{\frac{n}{B \log n}}\right)$
Constant ≥ 3	$\Omega\left((\frac{n}{\alpha B})^{\frac{1}{2}-\frac{1}{2D-2}}\right)$	$\Omega\left((\frac{n}{B})^{\frac{1}{2}-\frac{1}{2D-2}}\right)$
4	$\Omega\left((\frac{n}{\alpha B})^{1/3}\right)$	$\Omega\left((\frac{n}{B})^{1/3}\right)$
3	$\Omega\left((\frac{n}{\alpha B})^{1/4}\right)$	$\Omega\left((\frac{n}{B})^{1/4}\right)$

Figure 1: Lower bounds of randomized α-approximation algorithms on graphs of various diameters. Bounds in the first column are for the MST and shortest path tree problems [10] while those in the second column are for these problems and many problems listed in Figure 2. We note that these bounds almost match the $O(\sqrt{n}\log^* n + D)$ upper bound for the MST problem [12, 21] and are independent of the approximation-factor α. Also note a simple observation that lower bounds for graphs of diameter D also hold for graphs of larger diameters.

mate computation) for many other problems (e.g., minimum cut), thus answering some questions that were open earlier (see the survey by Elkin [7]).

The new lower bound for approximating MST simplifies and improves upon the previous $\Omega(\sqrt{n/(\alpha B \log n)} + D)$ lower bound by Elkin [10], where α is the approximation factor. [10] showed a *tradeoff* between the running time and the approximation ratio of MST. Our result shows that approximating MST requires $\Omega(\sqrt{n/(B \log n)} + D)$ rounds, *regardless of* α. Thus our result shows that there is actually no trade-off, since there can be no distributed approximation algorithm for MST that is significantly faster than the current exact algorithm [21, 9], for any approximation factor $\alpha > 1$.

1.4 Overview of Technical Approach

We prove our lower bounds by establishing an interesting connection between communication complexity and distributed computing. Our lower bound proofs consider the family of graphs evolved through a series of papers in the literature [10, 24, 29]. However, while previous results [29, 10, 24, 16] rely on counting the number of states needed to solve the *mailing problem* (along with some sophisticated techniques for its variant, called *corrupted mailing problem*, in the case of approximation algorithm lower bounds) and use Yao's method [36] (with appropriate input distributions) to get lower bounds for randomized algorithms, our results are achieved using a few steps of simple reductions, starting from problems in communication complexity, as follows (also see Figure 2 for details).

(Section 3) First, we reduce the lower bounds of problems in the standard communication complexity model [20] to the lower bounds of the equivalent problems in the "distributed version" of communication complexity. Specifically, we prove the *Simulation Theorem* (cf. Section 3) which relates the *communication* lower bound from the standard communication complexity model [20] to compute some appropriately chosen function f, to the distributed *time* complexity lower bound for computing the same function in a specially chosen graph G. In the standard model, Alice and Bob can communicate directly (via a bidirectional edge of bandwidth one). In the distributed model, we assume that Alice and Bob are some vertices of G and they together wish to compute the function

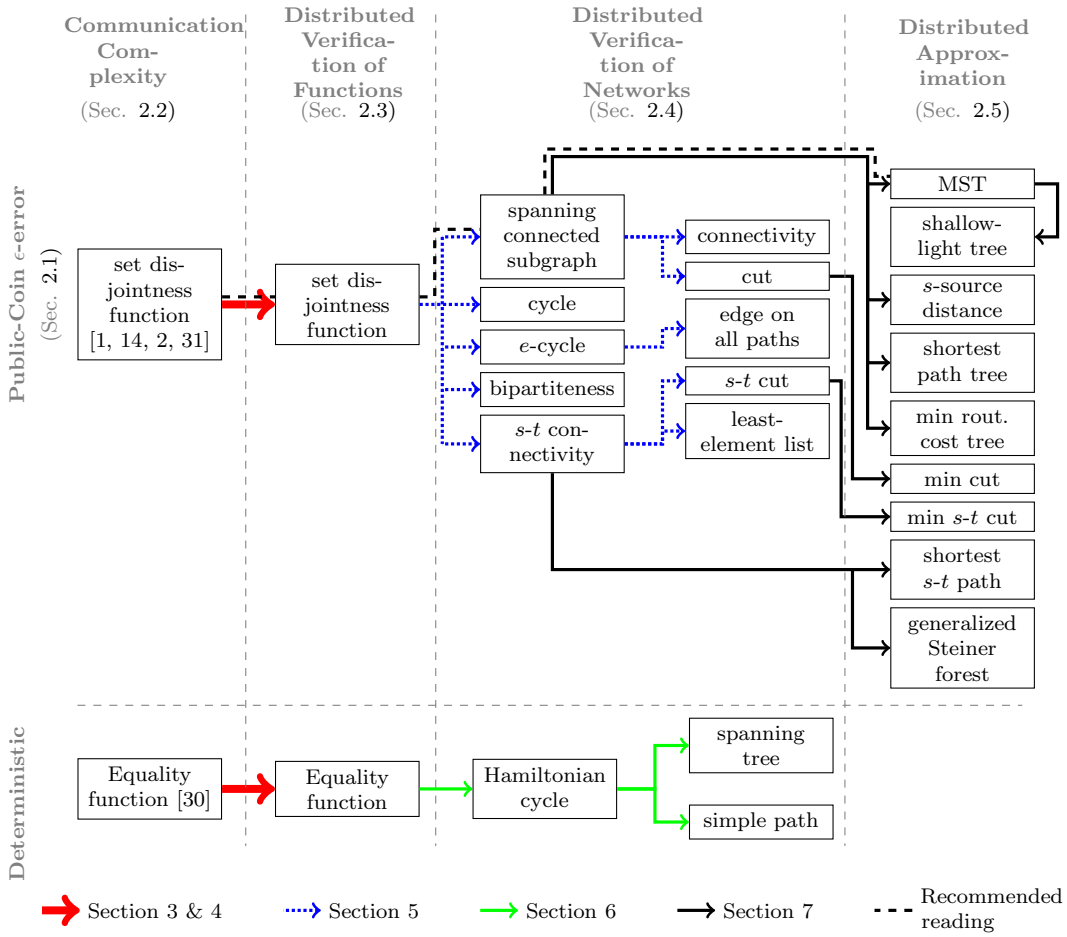

Figure 2: Problems and reductions between them to obtain randomized and deterministic lower bounds. For all problems, we obtain lower bounds as in Figure 1. In order to get the whole picture of the paper, we recommend reading along the black dashed line. Definitions of (Monte Carlo) randomized algorithms can be found in Section 2.1. Definitions of problems in communication complexity, distributed verification of functions, distributed verification of networks and distributed approximation, can be found in Section 2.2, 2.3, 2.4 and 2.5, respectively.

f using the communication graph G. The choice of graph G is critical. We use a graph called $G(\Gamma, d, p)$ (parameterized by Γ, d and p) that was first used in [10]. We show a reduction from the standard model to the distributed model, the proof of which relies on some observations used in previous results (e.g., [29]).

(Section 4) The connection established in the first step allows us to bypass the state counting argument and Yao's method, and reduces our task in proving lower bounds of verification problems to merely picking the right function f to reduce from. The function f that is useful in showing our randomized lower bounds is the *set disjointness function* [1, 14, 2, 31], which is the quintessential problem in the world of communication complexity with applications to diverse areas and has been studied for decades (see a recent survey in [3]). Following a result well known in communication complexity [20], we show that the distributed version of this problem has an $\Omega(\sqrt{n/(B \log n)})$ lower bound on graphs of small diameter.

(Section 5 & 6) We then reduce this problem to the verification problems using simple reductions similar to those used in data streams [13]. The set disjointness function yields randomized lower bounds and works for many problems (see Figure 2), but it does not reduce to certain other problems such as spanning tree. To show lower bounds for these other problems, we use a different function f called *equality function*. However, this reduction yields only deterministic lower bounds for the corresponding verification problems.

(Section 7) Finally, we reduce the verification problem to hardness of distributed approximation for a variety of problems to show that the same lower bounds hold for approximation algorithms as well. For this, we use a reduction whose idea is similar to one used to prove hardness of approximating TSP (Traveling Salesman Problem) on general graphs (see, e.g., [34]): We convert a verification problem to an optimization problem by introducing edge weights in such a way that there is a large gap between the optimal values for the cases where H satisfies, or does not satisfy a certain property. This technique is surprisingly simple, yet yields strong unconditional hardness bounds — many hitherto unknown, left open (e.g., minimum cut) [7] and some that improve over known ones (e.g., MST and shortest path tree) [10]. As mentioned earlier, our approach shows that approximating MST by *any* factor needs $\tilde{\Omega}(\sqrt{n})$ time, while the previous result due to Elkin gave a bound that depends on α (the approximation factor), i.e. $\tilde{\Omega}(\sqrt{n/\alpha})$, using more sophisticated techniques.

Figure 2 summarizes these reductions that will be proved in this paper. Our proof technique via this approach is quite general and conceptually straightforward to apply as it hides all complexities in the well studied communication complexity. Yet, it yields tight lower bounds for many problems (we show almost matching upper bounds for many problems in Section 8). It also has some advantages over the previous approaches. First, in the previous approach, we have to start from scratch every time we want to prove a lower bound for a new problem. For example, extending from the mailing problem in [29] to the corrupted mailing problem in [10] requires some sophisticated techniques. Our new technique allows us to use known lower bounds in communication complexity to do such a task. Secondly, extending a deterministic lower bound to a randomized one is sometimes difficult. As in our case, our randomized lower bound of the spanning connected subgraph problem would be almost impossible without connecting it to the communication complexity lower bound of the set disjointness problem (whose strong randomized lower bound is a result of years of studies [1, 14, 2, 31]). One important consequence is that this technique allows us to obtain lower bounds for *Monte Carlo* randomized algorithms while previous lower bounds hold only for Las Vegas randomized algorithms. We believe that this technique could lead to many new lower

bounds of distributed algorithms.

Recent results After the preliminary version of this paper ([5]) appeared, the connection between communication complexity and distributed algorithm lower bounds has been further used to develop some new lower bounds. In [27], the Simulation Theorem (cf. Theorem 3.1) is extended to show a connection between bounded-round communication complexity and distributed algorithm lower bounds. It is then used to show a tight lower bound for distributed random walk algorithms. In [11], lower bounds of computing diameter of a network and related problems are shown by reduction from the communication complexity of set disjointness. This is done by considering the communication at the bottleneck of the network (sometimes called *bisection width* [20, 22]). A similar argument is also used in [19] to show lower bounds on directed networks.

2 Preliminaries

To make it easy to look up for definitions, we collect all necessary definitions in this section. We recommend the readers to skip this section in the first read and come back when necessary.

This section is organized as follows (also see Figure 2 for a pointer to a subsection for each definition). In Subsection 2.1, we define the notion of ϵ-error randomized public-coin algorithms and the worse-case running time of these algorithms. In Subsection 2.2, we define the communication complexity model and the set disjointness and equality problems. We then extend this model to the model of distributed verification of functions in Subsection 2.3. In Subsection 2.4, we give a formal definition of the distributed verification problem which we explained informally in Section 1. We also define the specific distributed verification problems considered in this paper. Finally, in Subsection 2.5, we define the notion of approximation algorithms.

2.1 Randomized (Monte Carlo) Public-Coin Algorithms and the Worst-Case Running Time

In this paper, we show lower bounds of distributed algorithms that are *Monte Carlo*. Recall that a Monte Carlo algorithm is a randomized algorithm whose output may be incorrect with some probability. Formally, let $\mathcal{A}$ be any algorithm for computing a function f. We say that $\mathcal{A}$ computes f with *ϵ-error* if for every input x, $\mathcal{A}$ outputs $f(x)$ with probability at least $1 - \epsilon$. Note that a 0-error algorithm is deterministic.

We note the fact that lower bounds of Monte Carlo algorithms also imply lower bounds of *Las Vegas* algorithms (whose output is always correct but the running time is only in expectation). Thus, lower bounds in this paper hold for both types of algorithms.

Public coin We say that a randomized distributed algorithm uses a *public coin* if all nodes have an access to a common random string (chosen according to some probability distribution). In this paper, we are interested in the lower bounds of public-coin randomized distributed algorithms. We note that these lower bounds also imply time lower bounds of *private-coin* randomized distributed algorithms, where nodes do not share a random string, since allowing a public coin only gives more power to the algorithms.

Worst-case running time For any public-coin randomized distributed algorithm $\mathcal{A}$ on a network G and input $\mathcal{I}$ (given to nodes in G), we define the *worst-case running time* of $\mathcal{A}$ on input $\mathcal{I}$ to be the maximum number of rounds needed to run $\mathcal{A}$ among all possible (shared) random strings. The worst-case running time of $\mathcal{A}$ is the maximum, over all inputs $\mathcal{I}$, of the worst-case running time of $\mathcal{A}$ on $\mathcal{I}$.

2.2 Communication Complexity

In this paper, we consider the standard model of communication complexity. To avoid confusion, we define the model as a special case of the distributed algorithm model. We refer to [20] for the conventional definition, further details and discussions.

In this model, there are two nodes in the network connected by an edge. We call one node *Alice* and the other node *Bob*. Alice and Bob each receive a b-bit binary string, for some integer $b \geq 1$, denoted by x and y respectively. Together, they both want to compute $f(x, y)$ for a Boolean function $f : \{0,1\}^b \times \{0,1\}^b \to \{0,1\}$. In the end of the process, we want both Alice and Bob to know the value of $f(x, y)$. We are interested in the worst-case running time of distributed algorithms on this network when one bit can be sent on the edge in each round (thus the running time is equal to the number of bits Alice and Bob exchange).

For any Boolean function f and $\epsilon > 0$, we let $R_\epsilon^{cc-pub}(f)$ denote the minimum worst-case running time of the best ϵ-error randomized algorithm for computing f in the communication complexity model.

In this paper, we are interested in two Boolean functions, *set disjointness (*`disj`*)* and *equality (*`eq`*)* functions, defined as follows.

- **Set Disjointness function (`disj`).** Given two b-bit strings x and y, the *set disjointness function*, denoted by $\texttt{disj}(x, y)$, is defined to be one if the inner product $\langle x, y \rangle$ is 0 (i.e., there is no i such that $x_i = y_i = 1$) and zero otherwise.
- **Equality function (`eq`).** Given two b-bit strings x and y, the *equality function*, denoted by $\texttt{eq}(x, y)$, is defined to be one if $x = y$ and zero otherwise.

2.3 Distributed Verification of Functions

We consider the same problem as in the case of communication complexity. That is, Alice and Bob receive b-bit binary strings x and y respectively and they want to compute $f(x, y)$ for some Boolean function f. However, Alice and Bob are now distinct vertices in a B-model distributed network G (cf. Section 1.1). We denote Alice's node (which receives x) by s and Bob's node (which receives y) by r. At the end of the process, both s and r will output $f(x, y)$. We are interested in the worst-case running time a distributed algorithm needs in order to compute function f.

For any network G (with two nodes marked as s and r), Boolean function f and $\epsilon > 0$, we let $R_\epsilon^G(f)$ denote the worst-case running time of the best ϵ-error randomized distributed algorithm for computing f on G.

In this model, we consider the set disjointness and equality functions as in the communication complexity model (cf. Subsection 2.2).

2.4 Distributed Verification of Networks

We already gave an informal definition of this problem in Section 1. We now define the problem formally. In the distributed network G, we describe its subgraph H as an input as follows. Each node v in G with neighbors $u_1, \ldots, u_{d(v)}$, where $d(v)$ is the degree of v, has $d(v)$ *Boolean indicator variables* $Y_v(u_1), \ldots, Y_v(u_{d(v)})$ indicating which of the edges incident to v participate in the subgraph H. The indicator variables must be consistent, i.e., for every edge (u, v), $Y_v(u) = Y_u(v)$ (this is easy to verify locally with a single round of communication).

Let H_Y be the set of edges whose indicator variables are 1; that is,

$$H_Y = \{(u, v) \in E \mid Y_u(v) = 1\}.$$

Given a predicate Π (which may specify statements such as "H_Y is connected" or "H_Y is a spanning tree" or "H_Y contains a cycle"), the output for a verification problem at each vertex v is an assignment to a (Boolean) output variable A^v, where $A^v = 1$ if H_Y satisfies the predicate Π, and $A^v = 0$ otherwise.

We say that a distributed algorithm $\mathcal{A}_\Pi$ verifies predicate Π if, for every graph G and subgraph H_Y of G, all nodes in G knows whether H_Y satisfies Π after we run $\mathcal{A}_\Pi$; that is, after the execution of $\mathcal{A}_\Pi$ on graph G, at each vertex v the output variable A^v is one if H_Y satisfies predicate Π, and zero otherwise. Note again that the time complexity of the verification algorithm is measured with respect to the size and diameter of G (independently from H_Y). When Y is clear from the context, we use H to denote H_Y.

We now define problems considered in this paper.

- **connected spanning subgraph verification:** We want to verify whether H is connected and spans all nodes of G, i.e., every node in G is incident to some edge in H.
- **cycle containment verification:** We want to verify if H contains a cycle.
- **e-cycle containment verification:** Given an edge e in H (known to vertices adjacent to it), we want to verify if H contains a cycle containing e.
- **bipartiteness verification**: We want to verify whether H is bipartite.
- **s-t connectivity verification**: In addition to G and H, we are given two vertices s and t (s and t are known by every vertex). We would like to verify whether s and t are in the same connected component of H.
- **connectivity verification**: We want to verify whether H is connected.
- **cut verification:** We want to verify whether H is a cut of G, i.e., G is not connected when we remove edges in H.
- **edge on all paths verification:** Given two nodes u, v and an edge e. We want to verify whether e lies on all paths between u and v in H. In other words, e is a u-v cut in H.
- **s-t cut verification**: We want to verify whether H is an s-t cut, i.e., when we remove all edges E_H of H from G, we want to know whether s and t are in the same connected component or not.

- **least-element list verification [4, 15]:** The input of this problem is different from other problems and is as follows. Given a distinct rank (integer) $r(v)$ to each node v in the weighted graph G, for any nodes u and v, we say that v is the *least element* of u if v has the lowest rank among vertices of distance at most $d(u, v)$ from u. Here, $d(u, v)$ denotes the weighted distance between u and v. The *Least-Element List* (LE-list) of a node u is the set $\{\langle v, d(u, v)\rangle \mid v \text{ is the least element of u}\}$.

  In the least-element list verification problem, each vertex knows its rank as an input, and some vertex u is given a set $S = \{\langle v_1, d(u, v_1)\rangle, \langle v_2, d(u, v_2)\rangle, \ldots\}$ as an input. We want to verify whether S is the least-element list of u.

- **Hamiltonian cycle verification:** We would like to verify whether H is a Hamiltonian cycle of G, i.e., H is a simple cycle of length n.

- **spanning tree verification:** We would like to verify whether H is a tree spanning G.

- **simple path verification:** We would like to verify that H is a simple path, i.e., all nodes have degree either zero or two in H except two nodes that have degree one and there is no cycle in H.

2.5 Approximation Algorithms

In a graph optimization problem $\mathcal{P}$ in a distributed network, such as finding a MST, we are given a non-negative weight $\omega(e)$ on each edge e of the network (each node knows the weights of all edges incident to it). Each pair of network and weight function (G, ω) comes with a nonempty set of *feasible solutions* for a problem $\mathcal{P}$; e.g., for the case of finding a MST, all spanning trees of G are feasible solutions. The goal of $\mathcal{P}$ is to find a feasible solution that minimizes or maximizes the total weight. We call such a solution an *optimal solution*. For example, a spanning tree of minimum weight is an optimal solution for the MST problem.

For any $\alpha \geq 1$, an *α-approximate solution* of $\mathcal{P}$ on weighted network (G, ω) is a feasible solution whose weight is not more than α (respectively, $1/\alpha$) times of the weight of the optimal solution of $\mathcal{P}$ if $\mathcal{P}$ is a minimization (respectively, maximization) problem. We say that an algorithm $\mathcal{A}$ is an α-approximation algorithm for problem $\mathcal{P}$ if it outputs an α-approximate solution for any weighted network (G, ω). In case of randomized algorithms (cf. Subsection 2.1), we say that an α-approximation T-time algorithm is ϵ-error if it outputs an answer that is not α-approximate with probability at most ϵ and always finishes in time T, regardless of the input and the choice of random string.

In this paper, we consider the following problems.

- In the **minimum spanning tree** problem [10, 29], we want to compute the weight of the minimum spanning tree (i.e., the spanning tree of minimum weight). In the end of the process all nodes should know this weight.

- Consider a network with two cost functions associated to edges, weight and length, and a root node r. For any spanning tree T, the radius of T is the maximum length (defined by the length function) between r and any leaf node of T. Given a root node r and the desired radius ℓ, a **shallow-light tree** [28] is the spanning tree whose radius is at most ℓ and the total weight is minimized (among trees of the desired radius).

- Given a node s, the **s-source distance** problem [8] is to find the distance from s to every node. In the end of the process, every node knows its distance from s.

- In the **shortest path tree** problem [10], we want to find the shortest path spanning tree rooted at some input node s, i.e., the shortest path from s to any node t must have the same weight as the unique path from s to t in the solution tree. In the end of the process, each node should know which edges incident to it are in the shortest path tree.

- The **minimum routing cost spanning tree** problem [35] is defined as follows. We think of the weight of an edge as the cost of routing messages through this edge. The routing cost between any node u and v in a given spanning tree T, denoted by $c_T(u, v)$, is the distance between them in T. The routing cost of the tree T itself is the sum over all pairs of vertices of the routing cost for the pair in the tree, i.e., $\sum_{u,v \in V} c_T(u, v)$. Our goal is to find a spanning tree with minimum routing cost.

- A set of edges E' is a **cut** of G if G is not connected when we delete E'. The **minimum cut** problem [7] is to find a cut of minimum weight. A set of edges E' is an *s-t cut* if there is no path between s and t when we delete E' from G. The **minimum s-t cut** problem is to find an s-t cut of minimum weight.

- Given two nodes s and t, the **shortest s-t path** problem is to find the length of the shortest path between s and t.

- The **generalized Steiner forest** problem [15] is defined as follows. We are given k disjoint subsets of vertices $V_1, ..., V_k$ (each node knows which subset it is in). The goal is to find a minimum weight subgraph in which each pair of vertices belonging to the same subsets is connected. In the end of the process, each node knows which edges incident to it are in the solution.

Note that in the minimum spanning tree, minimum cut, minimum s-t cut, and shortest s-t path problems, an α-approximation algorithm should find a solution that has total weight at most α times the weight of the optimal solution. For the s-source distance problem, an α-approximation algorithm should find an approximate distance $d(v)$ of every vertex v such that $distance(s, v) \leq d(v) \leq \alpha \cdot distance(s, v)$ where $distance(s, v)$ is the distance of s from v. Similarly, an α-approximation algorithm for the shortest path tree problem should find a spanning tree T such that, for any node v, the length ℓ of a unique path from s to v in T satisfies $\ell \leq \alpha \cdot distance(s, v)$.

3 From Communication Complexity to Distributed Computing

In this section, we show a connection between the communication complexity model (cf. Section 2.2) and the model of distributed verification of functions (cf. Section 2.3) on a family of graphs called $G(\Gamma, d, p)$. This family of graphs was first defined in [10] (which was extended from [29]). We will define this graph in Subsection 3.1 for completeness.

The main result of this section shows that if there is a fast ϵ-error algorithm for computing f on $G(\Gamma, d, p)$, then there is a fast ϵ-error algorithm for Alice and Bob to compute f in the communication complexity model. We call this the *Simulation Theorem*. We state the theorem below. The rest of this section is devoted to define the graph $G(\Gamma, d, p)$ and to prove the theorem.

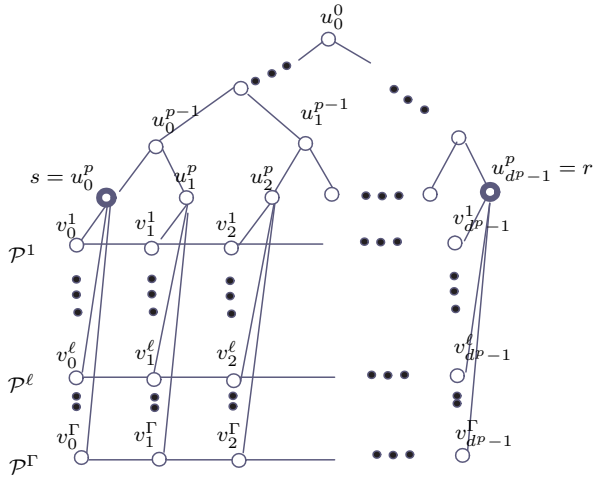

Figure 3: An example of $G(\Gamma, d, p)$ (here $d = 2$).

Theorem 3.1 (Simulation Theorem)**.** *For any Γ, d, p, B, $\epsilon \geq 0$, and function $f : \{0,1\}^b \times \{0,1\}^b \to \{0,1\}$, if there is an ϵ-error distributed algorithm on $G(\Gamma, d, p)$ that computes f faster than $\frac{d^p-1}{2}$ time, i.e.,*

$$R_\epsilon^{G(\Gamma,d,p)}(f) < \frac{d^p - 1}{2}$$

then there is an ϵ-error algorithm in the communication complexity model that computes f in at most $2dpBR_\epsilon^{G(\Gamma,d,p)}(f)$ time. In other words,

$$R_\epsilon^{cc-pub}(f) \leq 2dpBR_\epsilon^{G(\Gamma,d,p)}(f)\,.$$

We first describe the graph $G(\Gamma, d, p)$ with parameters Γ, d and p and distinct vertices s and r.

3.1 Description of $G(\Gamma, d, p)$ [10]

We now describe the network $G(\Gamma, d, p)$ in detail. The two basic units in the construction are *paths* and a *tree*. There are Γ paths, denoted by $\mathcal{P}^1, \mathcal{P}^2, \ldots, \mathcal{P}^\Gamma$, each having d^p nodes, i.e., for $\ell = 1, 2, \ldots \Gamma$,

$$V(\mathcal{P}^\ell) = \{v_0^\ell, \ldots, v_{d^p-1}^\ell\} \qquad \text{and} \qquad E(\mathcal{P}^\ell) = \{(v_i^\ell, v_{i+1}^\ell) \mid 0 \leq i < d^p - 1\}\,.$$

There is a tree, denoted by $\mathcal{T}$ having depth p where each non-leaf node has d children (thus, there are d^p leaf nodes). We denote the nodes of $\mathcal{T}$ at level ℓ from left to right by $u_0^\ell, \ldots, u_{d^\ell-1}^\ell$ (so, u_0^0 is the root of $\mathcal{T}$ and $u_0^p, \ldots, u_{d^p-1}^p$ are the leaves of $\mathcal{T}$). For any ℓ and j, the leaf node u_j^p is connected to the corresponding path node v_j^ℓ by a *spoke* edge (u_j^p, v_j^ℓ). Finally, we set the two special nodes (which will receive input strings x and y) as $s = u_0^p$ and $r = u_{d^p-1}^p$. Figure 3 depicts this network. We note the following lemma proved in [10].

Lemma 3.2. *[10] The number of vertices in $G(\Gamma, d, p)$ is $n = \Theta(\Gamma d^p)$ and its diameter is $2p + 2$.*

3.2 Terminologies

For any $1 \leq i \leq \lfloor (d^p - 1)/2 \rfloor$, define the *i-left* and the *i-right* of path $\mathcal{P}^\ell$ as

$$L_i(\mathcal{P}^\ell) = \{v_j^\ell \mid j \leq d^p - 1 - i\} \qquad \text{and} \qquad R_i(\mathcal{P}^\ell) = \{v_j^\ell \mid j \geq i\},$$

respectively. Thus, $L_0(\mathcal{P}^\ell) = R_0(\mathcal{P}^\ell) = V(\mathcal{P}^\ell)$. Define the i-left of the tree $\mathcal{T}$, denoted by $L_i(\mathcal{T})$, as the union of the set $S = \{u_j^p \mid j \leq d^p - 1 - i\}$ and all ancestors of all vertices in S. Similarly, the i-right $R_i(\mathcal{T})$ of the tree $\mathcal{T}$ is the union of set $S = \{u_j^p \mid j \geq i\}$ and all ancestors of all vertices in S. Now, the *i-left* and *i-right* sets of $G(\Gamma, d, p)$ are the union of those left and right sets,

$$L_i = \bigcup_\ell L_i(\mathcal{P}^\ell) \cup L_i(\mathcal{T}) \qquad \text{and} \qquad R_i = \bigcup_\ell R_i(\mathcal{P}^\ell) \cup R_i(\mathcal{T}).$$

For $i = 0$, we modify the definition and set $L_0 = V \setminus \{r\}$ and $R_0 = V \setminus \{s\}$. See Figure 3.2.

Let $\mathcal{A}$ be any *deterministic* distributed algorithm run on graph $G(\Gamma, d, p)$ for computing a function f. Fix any input strings x and y given to s and r respectively. Let $\varphi_{\mathcal{A}}(x, y)$ denote the execution of $\mathcal{A}$ on x and y. Denote the *state* of the vertex v at the end of round t during the execution $\varphi_{\mathcal{A}}(x, y)$ by $\sigma_{\mathcal{A}}(v, t, x, y)$.

We note the following important property of distributed algorithms. *The state of a vertex v at the end of time t is uniquely determined by its input and the sequence of messages on each of its incoming links from time 1 to t.* Intuitively, this is because a distributed algorithm is simply a set of algorithms run on different nodes in a network. The algorithm on each node behaves according to its input and the sequence of messages sent to it so far. From this, for example, we can conclude that in two different executions $\varphi_{\mathcal{A}}(x, y)$ and $\varphi_{\mathcal{A}}(x', y')$, a vertex reaches the same state at time t (i.e., $\sigma_{\mathcal{A}}(v, t, x, y) = \sigma_{\mathcal{A}}(v, t, x', y')$) if and only if it receives the same sequence of messages on each of its incoming links.

For a given set of vertices $U = \{v_1, \ldots, v_\ell\} \subseteq V$, a *configuration*

$$C_{\mathcal{A}}(U, t, x, y) = \langle \sigma_{\mathcal{A}}(v_1, t, x, y), \ldots, \sigma_{\mathcal{A}}(v_\ell, t, x, y) \rangle$$

is a vector of the states of the vertices of U at the end of round t of the execution $\varphi_{\mathcal{A}}(x, y)$.

3.3 Observations

We note the following crucial observations developed in [29, 10, 24, 16]. We will need Lemma 3.4 to prove Theorem 3.1 in the next subsection.

Observation 3.3. *For any set $U \subseteq U' \subseteq V$, $C_{\mathcal{A}}(U, t, x, y)$ can be uniquely determined by $C_{\mathcal{A}}(U', t-1, x, y)$ and all messages sent to U from $V \setminus U'$ at time t.*

Proof. Recall that the state of each vertex v in U can be uniquely determined by its state $\sigma_{\mathcal{A}}(v, t-1, x, y)$ at time $t-1$ and the messages sent to it at time t. Moreover, the messages sent to v from vertices inside U' can be determined by $C_{\mathcal{A}}(U', t-1, x, y)$. Thus if the messages sent from vertices in $V \setminus U'$ are given then we can determine all messages sent to U at time t and thus we can determine $C_{\mathcal{A}}(U, t, x, y)$. □

From now on, to simplify notations, when $\mathcal{A}$, x and y are clear from the context, we use C_{L_t} and C_{R_t} to denote $C_{\mathcal{A}}(L_t, t, x, y)$ and $C_{\mathcal{A}}(R_t, t, x, y)$, respectively. The lemma below states that C_{L_t}

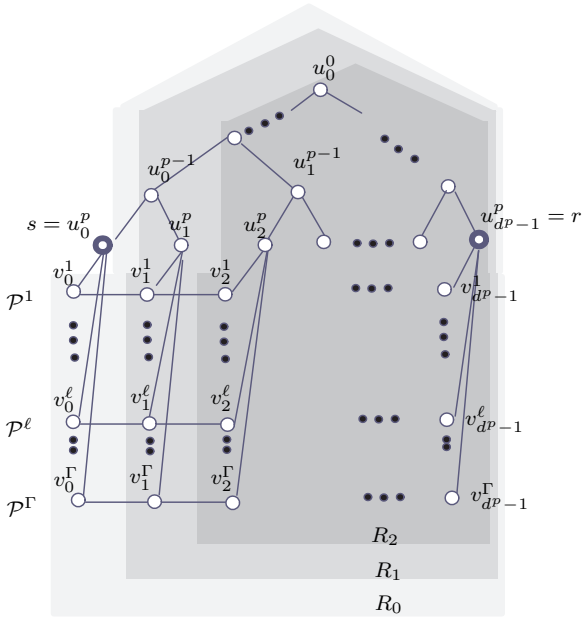

Figure 4: Examples of i-right sets.

(C_{R_t}, respectively) can be determined by $C_{L_{t-1}}$ ($C_{R_{t-1}}$, respectively) and dp messages generated by some vertices in R_{t-1} (L_{t-1} respectively) at time t. It essentially follows from Observation 3.3 and an observation that there are at most d^p edges linking between vertices in $V \setminus R_{t-1}$ ($V \setminus L_{t-1}$ respectively) and vertices in R_t (L_t respectively).

Lemma 3.4. *Fix any deterministic algorithm $\mathcal{A}$ and input strings x and y. For any $0 < t < (d^p - 1)/2$, there exist functions g_L and g_R, B-bit messages $M_1^{L_{t-1}}, \ldots, M_{dp}^{L_{t-1}}$ sent by some vertices in L_{t-1} at time t, and B-bit messages $M_1^{R_{t-1}}$, ..., $M_{dp}^{R_{t-1}}$ sent by some vertices in R_{t-1} at time t such that*

$$C_{L_t} = g_L(C_{L_{t-1}}, M_1^{R_{t-1}}, \ldots, M_{dp}^{R_{t-1}}), \text{ and} \tag{1}$$

$$C_{R_t} = g_R(C_{R_{t-1}}, M_1^{L_{t-1}}, \ldots, M_{dp}^{L_{t-1}}). \tag{2}$$

Proof. We prove Eq. (2) only. (Eq. (1) is proved in exactly the same way.) First, observe the following facts about neighbors of nodes in R_t.

- All neighbors of all path vertices in R_t are in R_{t-1}. *Example:* In Figure 3.2, path vertices in R_2 are $v_2^\ell, \ldots, v_{d^p-1}^\ell$ for $\ell = 1, \ldots, \Gamma$. Observe that all neighbors of these vertices, i.e. $v_1^\ell, \ldots, v_{d^p-1}^\ell$ for all ℓ and $u_1^p, \ldots, u_{d^p-1}^p$, are in R_1.
- All neighbors of all leaf vertices in $V(\mathcal{T}) \cap R_t$ are in R_{t-1}. *Example:* In Figure 3.2 vertices in R_2 are $u_2^p, \ldots, u_{d^p-1}^p$. Their neighbors, i.e. $v_2^\ell, \ldots, v_{d^p-1}^\ell$ for all ℓ and $u_2^{p-1}, \ldots, u_{d^{p-1}-1}^{p-1}$, are all in R_1.
- For any non-leaf tree vertex u_i^ℓ, for any ℓ and i, if u_i^ℓ is in R_t then its parent and vertices $u_{i+1}^\ell, u_{i+2}^\ell, \ldots, u_{d^\ell-1}^\ell$ are in R_{t-1}. *Example:* In Figure 3.2, u_1^{p-1} is in R_2. Thus, its parent (u_0^{p-2}) and $u_2^{p-1}, \ldots, u_{d^{p-1}-1}^{p-1}$ are in R_1.

- For any i and ℓ, if u_i^ℓ is in R_t then all children of u_{i+1}^ℓ are in R_t (otherwise, all children of u_i^ℓ are not in R_t and so is u_i^ℓ, a contradiction). *Example:* In Figure 3.2, u_1^{p-1} is in R_2. Thus, all children of u_2^{p-1} are in R_2.

Let $u^\ell(R_t)$ denote the leftmost vertex that is at level ℓ of $\mathcal{T}$ and in R_t, i.e., $u^\ell(R_t) = u_i^\ell$ where i is such that $u_i^\ell \in R_t$ and $u_{i-1}^\ell \notin R_t$. (For example, in Figure 3.2, $u^{p-1}(R_1) = u_0^{p-1}$ and $u^{p-1}(R_2) = u_1^{p-1}$.) From the above observations, we conclude that the only neighbors of nodes in R_t that are not in R_{t-1} are children of $u^\ell(R_t)$, for all ℓ. In other words, all edges linking between vertices in R_t and $V \setminus R_{t-1}$ are in the following form: $(u^\ell(R_t), u')$ for some ℓ and child u' of $u^\ell(R_t)$.

Setting $U' = R_{t-1}$ and $U = R_t$ in Observation 3.3, we have that C_{R_t} can be uniquely determined by $C_{R_{t-1}}$ and messages sent to $u^\ell(R_t)$ from its children in $V \setminus R_{t-1}$. Note that each of these messages contains at most B bits since they correspond to a message sent on an edge in one round.

Observe further that, for any $t < (d^p - 1)/2$, $V \setminus R_{t-1} \subseteq L_{t-1}$ since L_{t-1} and R_{t-1} share some path vertices. Moreover, each $u^\ell(R_t)$ has d children. Therefore, if we let $M_1^{L_{t-1}}, \ldots, M_{dp}^{L_{t-1}}$ be the messages sent from children of $u^0(R_t), u^1(R_t), \ldots, u^{p-1}(R_t)$ in $V \setminus R_{t-1}$ to their parents (note that if there are less than dp such messages then we add some empty messages) then we can uniquely determine C_{R_t} by $C_{R_{t-1}}$ and $M_1^{L_{t-1}}, \ldots, M_{dp}^{L_{t-1}}$. Eq. (2) thus follows. □

Using the above lemma, we can now prove Theorem 3.1.

3.4 Proof of the Simulation Theorem (cf. Theorem 3.1)

Let f be the function in the theorem statement. Let $\mathcal{A}_\epsilon$ be any ϵ-error distributed algorithm for computing f on network $G(\Gamma, d, p)$. Fix a random string $\bar{r}$ used by $\mathcal{A}_\epsilon$ (shared by all vertices in $G(\Gamma, d, p)$) and consider the *deterministic* algorithm $\mathcal{A}$ run on the input of $\mathcal{A}_\epsilon$ and the fixed random string $\bar{r}$. Let $T_\mathcal{A}$ be the worst case running time of algorithm $\mathcal{A}$ (over all inputs). We note that $T_\mathcal{A} < (d^p - 1)/2$, as assumed in the theorem statement. We show that Alice and Bob, when given $\bar{r}$ as the public random string, can simulate $\mathcal{A}$ using at most $2dpBT_\mathcal{A}$ communication bits, as follows.

Alice and Bob make $T_\mathcal{A}$ *iterations* of communications. Initially, Alice computes C_{L_0} which depends only on x. Bob also computes C_{R_0} which depends only on y. In each iteration $t > 0$, we assume that Alice and Bob know $C_{L_{t-1}}$ and $C_{R_{t-1}}$, respectively, before the iteration starts. Then, Alice and Bob will exchange at most $2dpB$ bits so that Alice and Bob know C_{L_t} and C_{R_t}, respectively, at the end of the iteration.

To do this, Alice sends to Bob the messages $M_1^{L_{t-1}}, \ldots, M_{dp}^{L_{t-1}}$ as in Lemma 3.4. Alice can generate these messages since she knows $C_{L_{t-1}}$ (by assumption). Then, Bob can compute C_{R_t} using Eq. (2) in Lemma 3.4. Similarly, Bob sends dp messages to Alice and Alice can compute C_{L_t}. They exchange at most $2dpB$ bits in total in each iteration since there are $2dp$ messages, each of B bits, exchanged.

After $T_\mathcal{A}$ iterations, Alice knows $C(L_{T_\mathcal{A}}, T_\mathcal{A}, x, y)$ and Bob knows $C(R_{T_\mathcal{A}}, T_\mathcal{A}, x, y)$. In particular, they know the output of $\mathcal{A}$ (output by s and r) since Alice and Bob know the states of s and r, respectively, after $\mathcal{A}$ terminates. They can thus output the output of $\mathcal{A}$.

Since Alice and Bob output exactly the output of A, they will answer correctly if and only if $\mathcal{A}$ answers correctly. Thus, if $\mathcal{A}$ is ϵ-error then so is the above communication protocol between Alice and Bob. Moreover, Alice and Bob communicate at most $2dpBT_\mathcal{A}$ bits. The theorem follows.

4 Distributed Verification of Set Disjointness and Equality Functions

In this section, we show lower bounds of distributed algorithms for verifying set disjointness and equality. The definitions of both problems can be found in Section 2.2 and the model of distributed verification of functions can be found in Section 2.3. The results in this section are simple corollaries of the Simulation Theorem (cf. Theorem 3.1) and will serve as important building blocks in showing lower bounds in later sections.

4.1 Randomized Lower Bound of Set Disjointness Function

To prove the lower bound of verifying `disj`, we simply use the communication complexity lower bound of computing `disj` [1, 14, 2, 31], i.e., $R_\epsilon^{cc-pub}(\texttt{disj}) = \Omega(b)$ where b is the size of input strings x and y.

Lemma 4.1. *For any Γ, d, p, there exists a constant $\epsilon > 0$ such that*

$$R_\epsilon^{G(\Gamma,d,p)}(\texttt{disj}) = \Omega(\min(d^p, \frac{b}{dpB})),$$

where b is the size of input strings x and y of `disj`*; i.e., any ϵ-error algorithm computing function* `disj` *on $G(\Gamma, d, p)$ requires $\Omega(\min(d^p, \frac{b}{dpB}))$ time.*

Proof. If $R_\epsilon^{G(\Gamma,d,p)}(\texttt{disj}) \geq (d^p-1)/2$ then $R_\epsilon^{G(\Gamma,d,p)}(\texttt{disj}) = \Omega(d^p)$ and we are done. Otherwise the conditions of Theorem 3.1 are fulfilled and it implies that $R_\epsilon^{cc-pub}(\texttt{disj}) \leq 2dpB \cdot R_\epsilon^{G(\Gamma,d,p)}(\texttt{disj})$. Now we use the fact that $R_\epsilon^{cc-pub}(\texttt{disj}) = \Omega(b)$ for the function `disj` on b-bit inputs, for some $\epsilon > 0$ [1, 14, 2, 31] (also see [20, Example 3.22] and references therein). It follows that $R_\epsilon^{G(\Gamma,d,p)}(\texttt{disj}) = \Omega(b/(dpB))$. □

4.2 Deterministic Lower Bound of Equality Function

To prove the lower bound of verifying `eq`, we simply use the deterministic communication complexity lower bound of computing `eq` [37], i.e., $R_0^{cc-pub}(\texttt{eq}) = \Omega(b)$ where b is the size of input strings x and y (see, e.g., [20, Example 1.21] and references therein).

Lemma 4.2. *For any Γ, d, p,*

$$R_0^{G(\Gamma,d,p)}(\texttt{eq}) = \Omega(\min(d^p, \frac{b}{dpB})),$$

where b is the size of input strings x and y of `eq`*; i.e., any deterministic algorithm computing function* `eq` *on $G(\Gamma, d, p)$ requires $\Omega(\min(d^p, \frac{b}{dpB}))$ time.*

Proof. If $R_0^{G(\Gamma,d,p)}(\texttt{eq}) \geq (d^p - 1)/2$ then $R_0^{G(\Gamma,d,p)}(\texttt{eq}) = \Omega(d^p)$ and we are done. Otherwise, the conditions of Theorem 3.1 are fulfilled and it implies that $R_0^{cc-pub}(\texttt{eq}) \leq 2dpB \cdot R_0^{G(\Gamma,d,p)}(\texttt{eq})$. Now we use the fact that $R_0^{cc-pub}(\texttt{eq}) = \Omega(b)$ for the function `eq` on b-bit inputs. It follows that $R_\epsilon^{G(\Gamma,d,p)}(\texttt{eq}) = \Omega(b/(dpB))$. □

5 Randomized Lower Bounds for Distributed Verification

In this section, we present randomized lower bounds for many verification problems on graphs of various diameters, as shown in Figure 1. These problems are defined in Section 2.4. The key ingredient is the lower bound of verifying the set disjointness function on distributed networks (cf. Lemma 4.1). The general theorem is as follows.

Theorem 5.1. *For any $p \geq 1$, $B \geq 1$, and $n \in \{2^{2p+1}pB, 3^{2p+1}pB, \ldots\}$, there exists a constant $\epsilon > 0$ such that any ϵ-error distributed algorithm for any of the following problems requires $\Omega((n/(pB))^{\frac{1}{2}-\frac{1}{2(2p+1)}})$ time on some $\Theta(n)$-vertex graph of diameter $2p+2$ in the B model: Spanning connected subgraph, cycle containment, e-cycle containment, bipartiteness, s-t connectivity, connectivity, cut, edge on all paths, s-t cut and least-element list.*

In particular, for graphs with diameter $D = 4$, we get $\Omega((n/B)^{1/3})$ lower bound and for graphs with diameter $D = \log n$ we get $\Omega(\sqrt{n/(B \log n)})$. Similar analysis also leads to a $\Omega(\sqrt{n/B})$ lower bound for graphs of diameter n^δ for any $\delta > 0$, and $\Omega((n/B)^{1/4})$ lower bound for graphs of diameter three using the same analysis as in [10]. We note again that the lower bound holds even in the public coin model where every vertex shares a random string.

Organization This section is organized as follows. In the first three subsections, we show lower bounds that need a reduction from the set disjointness problem (i.e., problems in the third column in Figure 2): spanning connected subgraph verification in Subsection 5.1, *s-t* connectivity verification in Subsection 5.2 and cycle containment, *e*-cycle containment, and bipartiteness verification in Subsection 5.3 (these problems are proved together as they use the same construction). The lower bounds on the remaining problems (connectivity, cut, edges on all paths, *s-t* cut and least-element list verification) are in Subsection 5.4.

5.1 Lower Bound of Spanning Connected Subgraph Verification Problem

The lower bound of spanning connected subgraph verification essentially follows from the following lemma which says that an algorithm for solving spanning connected subgraph verification can be used to compute `disj` as well.

Lemma 5.2. *For any Γ, $d \geq 2$, p and $\epsilon \geq 0$, if there exists an ϵ-error distributed algorithm for the spanning connected subgraph verification problem on graph $G(\Gamma, d, p)$ then there exists an ϵ-error algorithm for verifying* `disj` *(on Γ-bit inputs) on $G(\Gamma, d, p)$ that uses the same time complexity.*

Proof. Consider an ϵ-error algorithm $\mathcal{A}$ for the spanning connected subgraph verification problem, and suppose that we are given an instance of the set disjointness problem with Γ-bit input strings x and y (given to s and r). We use $\mathcal{A}$ to solve this instance of the set disjointness problem by constructing H as follows.

First, we mark all path edges and tree edges as participating in H. All spoke edges are marked as not participating in subgraph H, except those incident to s and r for which we do the following: For each bit x_i, $1 \leq i \leq \Gamma$, vertex s indicates that the spoke edge (s, v_0^i) participates in H if and only if $x_i = 0$. Similarly, for each bit y_i, $1 \leq i \leq \Gamma$, vertex r indicates that the spoke edge $(r, v_{d^p-1}^i)$ participates in H if and only if $y_i = 0$. (See Figure 5.)

Note that the participation of all edges, except those incident to s and r, is decided independently of the input. Moreover, one round is sufficient for s and r to inform their neighbors of

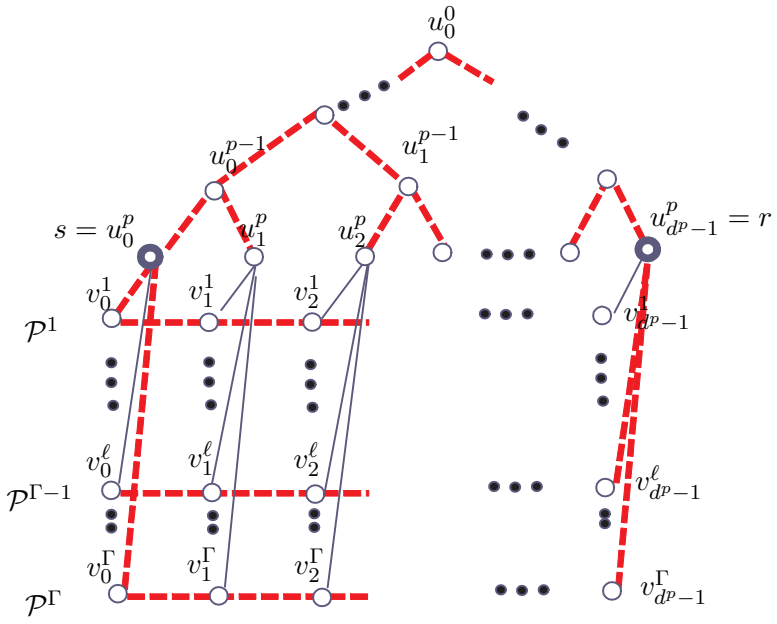

Figure 5: Example of H for the spanning connected subgraph problem (marked with dashed edges (red edges)) when $x = 0...10$ and $y = 1...00$.

the participation of edges incident to them. Hence, one round is enough to construct H. Then, algorithm $\mathcal{A}$ is started.

Once algorithm $\mathcal{A}$ terminates, vertex r determines its output for the set disjointness problem by stating that both input strings are disjoint if and only if the spanning connected subgraph verification algorithm verified that the given subgraph H is indeed a spanning connected subgraph.

Observe that H is a spanning connected subgraph if and only if for all $1 \leq i \leq \Gamma$ at least one of the edges (s, v_0^i) and $(r, v_{d^p-1}^i)$ is in H; thus, by the construction of H, H is a spanning connected subgraph if and only if the input strings x, y are disjoint, i.e., for every i either $x_i = 0$ or $y_i = 0$. Hence the resulting algorithm has correctly solved the given instance of the set disjointness problem when $\mathcal{A}$ correctly solve the spanning connected subgraph verification problem on the constructed subgraph H. This happens with probability at least $1 - \epsilon$. □

Using Lemma 4.1, we obtain the following result.

Corollary 5.3. *For any Γ, d, p, there exists a constant $\epsilon > 0$ such that any ϵ-error algorithm for the spanning connected subgraph verification problem requires $\Omega(\min(d^p, \frac{\Gamma}{dpB}))$ time on some $\Theta(\Gamma d^p)$-vertex graph of diameter $2p + 2$.*

In particular, if we consider $\Gamma = d^{p+1}pB$ then $\Omega(\min(d^p, \Gamma/(dpB))) = \Omega(d^p)$. Moreover, by Lemma 3.2, $G(d^{p+1}pB, d, p)$ has $n = \Theta(d^{2p+1}pB)$ vertices and thus the lower bound of $\Omega(d^p)$ becomes $\Omega((n/(pB))^{\frac{1}{2} - \frac{1}{2(2p+1)}})$. Theorem 5.1 (for the case of spanning connected subgraph) follows.

5.2 Lower Bound of s-t Connectivity Verification Problem

We again modify the proof of Lemma 5.2 to prove the following lemma.

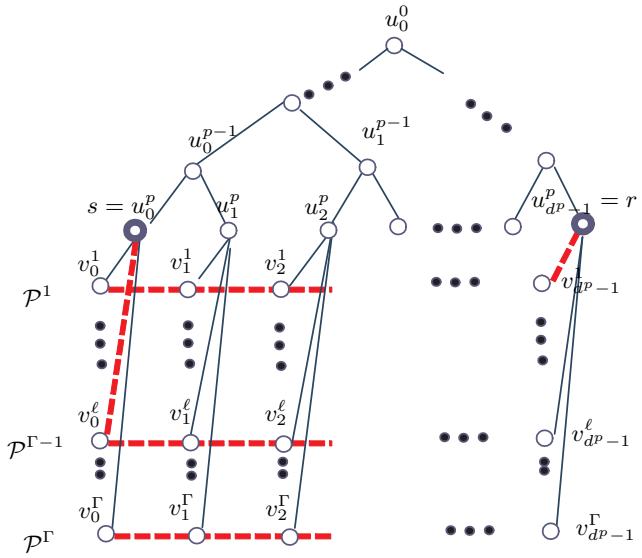

Figure 6: Example of H for s-t connectivity problem (marked with dashed edges (red edges)) when $x = 0...10$ and $y = 1...00$.

Lemma 5.4. *For any Γ, $d \geq 2$, p and $\epsilon \geq 0$ if there exists an ϵ-error distributed algorithm for the s-t connectivity verification problem on graph $G(\Gamma, d, p)$ then there exists an ϵ-error algorithm for verifying* `disj` *(on Γ-bit inputs) on $G(\Gamma, d, p)$ that uses the same time complexity.*

Proof. We use the same argument as in the proof of Lemma 5.2 except that we construct the subgraph H as follows.

First, all path edges are marked as participating in subgraph H. All tree edges are marked as not participating in H. All spoke edges, except those incident to s and r, are also marked as not participating. For each bit x_i, $1 \leq i \leq \Gamma$, vertex s indicates that the spoke edge (s, v_0^i) participates in H if and only if $x_i = 1$. Similarly, for each bit y_i, $1 \leq i \leq \Gamma$, vertex r indicates that the spoke edge $(r, v_{d^p-1}^i)$ participates in H if and only if $y_i = 1$. (See Figure 6.)

Observe that s and r are connected in H if and only if there exists $1 \leq i \leq \Gamma$ such that both edges $(v_0^i, s), (v_{d^p-1}^i, r)$ are in H; thus, by the construction of H, H is s-r connected if and only if the input strings x and y are *not* disjoint. □

5.3 Lower Bounds of Cycle Containment, e-Cycle Containment, and Bipartiteness Verification Problems

We modify the proof of Lemma 5.4 to prove the following lemma which says that an algorithm for solving problems in this section can be used to compute `disj`.

Lemma 5.5. *For any Γ, $d \geq 2$, p and $\epsilon \geq 0$ if there exists an ϵ-error distributed algorithm for solving either the cycle containment, e-cycle containment or bipartiteness verification problem on graph $G(\Gamma, d, p)$ then there exists an ϵ-error algorithm for verifying* `disj` *(on Γ-bit inputs) on $G(\Gamma, d, p)$ that uses the same time complexity.*

Proof. We prove this lemma by modifying the proof of Lemma 5.4. We only note the key difference here.

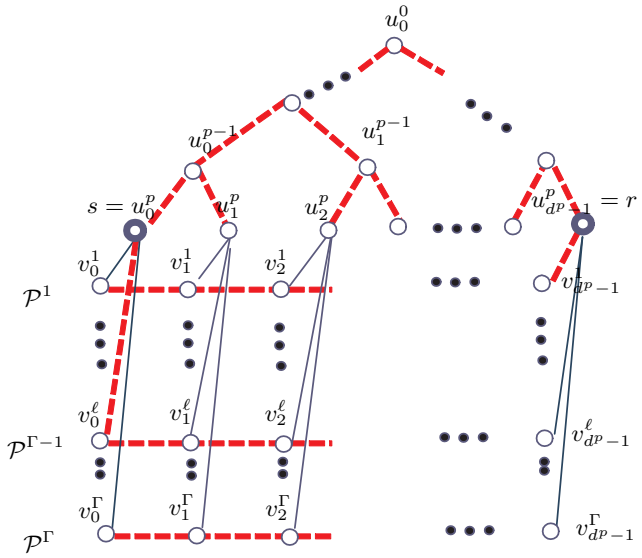

Figure 7: Example of H for the cycle and e-cycle containment and bipartiteness verification problem when $x = 0...10$ and $y = 1...00$.

Cycle containment verification problem: We construct H in the same way as in the proof of Lemma 5.4 *except* that the tree edges are participating in H (see Figure 7).

In the case that the input strings are disjoint, H will consist of the tree connecting s and r as well as 1) paths connected to s but not to r, 2) paths connected to r but not to s and 3) paths connected neither to r nor s. Thus there is no cycle in H. In the case that the input strings are not disjoint, we let i be an index that makes them not disjoint, that is $x_i = y_i = 1$. This causes a cycle in H consisting of some tree edges and path $\mathcal{P}^i$ that are connected by edges (s, v_0^i) and $(v_{d^p-1}^i, r)$ at their endpoints. Thus we have the following claim.

Claim 5.6. *H contains a cycle if and only if the input strings are not disjoint.*

e-cycle containment verification problem: We use the previous construction for H and let e be the tree edge adjacent to s (i.e., e connects s to its parent). Observe that, in this construction, H contains a cycle if and only if H contains a cycle containing e. Therefore, we have the following claim.

Claim 5.7. *e is contained in a cycle in H if and only if the input strings are not disjoint.*

Bipartiteness verification problem: Finally, we can verify if such an edge e is contained in a cycle by verifying the bipartiteness. First, we replace $e = (s, u_0^{p-1})$ by a path (s, v', u_0^{p-1}), where v' is an additional/virtual vertex. This can be done without changing the input graph G by having vertex s simulated algorithms on both s and v'. The communication between s and v' can be done internally. The communication between v' and u_0^{p-1} can be done by s. We construct H' the same way as H with both (s, v') and (v', u_0^{p-1}) marked as participating.

We observe that if the input strings are not disjoint, then either H or H' are not bipartite. To see this, consider two cases: when d^p is even and odd. When d^p is even and the input strings are not disjoint, there exists i such that there is a cycle in H consisting of some tree edges (including e) and path P_i that are connected by edges (s, v_0^i) and $(v_{d^p-1}^i, r)$ at their endpoints. This cycle is

of length $2p + (d^p - 1) + 2$ – an odd number causing H to be not bipartite. If d^p is odd, then by the same argument there is an odd cycle of length $(2p+1) + (d^p - 1) + 2$ in H' (this cycle includes the edges (s, v') and (v', u_0^{p-1}) that replaces e); thus H' is not bipartite.

Now we consider the converse: If the input strings are disjoint, then H does not contain a cycle by the argument of the proof of the cycle containment problem (which uses the same graph). It follows that H' does not contain a cycle as well. Therefore, we have the following claim.

Claim 5.8. *H and H' are both bipartite if and only if the input strings are disjoint.*

□

We note that the above reduction for the bipartiteness verification problem might seem to suggest that one can also prove the lower bound of this problem by reducing from the e-cycle verification problem. However, this is not the case. The reason is that the above proof relies on the fact that H and H' each contains at most one cycle and such cycle must contain e. In general, this might not be the case.

5.4 Lower Bounds of Connectivity, Cut, Edges on All Paths, s-t Cut and Least-element List Verification Problems

Lower bounds of verification problems in this section are proven using the lower bounds of problems in Section 5.1, 5.2 and 5.3.

Connectivity verification problem We reduce from the spanning connected subgraph verification problem. Let $\mathcal{A}(G, H)$ be an algorithm that verifies if H is connected in $O(\tau(n))$ time on any n-vertex graph G and subgraph H. Now we will use this algorithm to verify whether a subgraph H' of G is connected or not.

Recall that, by definition, H' is a spanning connected subgraph if and only if every node is incident to at least one edge in H' and H' is connected. Verifying that every node is incident to at least one edge in H' can be done locally and all nodes can be notified if this is not the case in $O(D)$ rounds (by broadcasting). Checking if H' is connected can be done in $O(\tau(n))$ rounds by running $\mathcal{A}(G, H')$. The total running time for checking if H' is a spanning connected subgraph is thus $O(\tau(n) + D)$. The lower bound of the spanning connected subgraph problem thus applies to the connectivity verification problem as well.

Cut verification problem We again reduce from the spanning connected subgraph problem. Given a subgraph H, we verify if H is a spanning connected subgraph as follows. Let $\bar{H}$ be the graph obtained by removing edges $E(H)$ of H from G. Recall that H is a spanning connected subgraph if and only if $\bar{H}$ is not a cut (see definition of a cut in Section 2.4). Thus, we verify if $\bar{H}$ is a cut and announce that H is a spanning connected subgraph if and only if $\bar{H}$ is not a cut.

s-t cut verification problem We reduce from s-t connectivity. Similar to above, we use the fact that H is s-t connected if and only if $\bar{H}$ is *not* an s-t cut.

Least-element list verification problem We reduce from s-t connectivity. We set the rank of s to 0 and the rank of other nodes to any distinct positive integers. We assign weight 0 to all edges in H and 1 to other edges. Give a set $S = \{< s, 0 >\}$ to vertex t. Then we verify if S is the least-element list of t. Observe that if s and t are connected by H then the distance between them must be 0 and thus S is the least-element list of t. Conversely, if s and t are not connected then the distance between them will be at least one and S will not be the least-element list of t.

Edge on all paths verification problem We reduce from the e-cycle containment problem using the following observation: H does not contain a cycle containing e if and only if e lies on all paths between u and v in H where u and v are two nodes incident to e.

6 Deterministic Lower Bounds of Distributed Verification

In this section, we present deterministic lower bounds for Hamiltonian cycle, spanning tree and simple path verification. These problems are defined in Section 2.4. These lower bounds are proved in almost the same way as in Section 5. The only difference is that we reduce from the deterministic lower bound of the *Equality* problem (cf. Lemma 4.2).

Theorem 6.1. *For any p, $B \geq 1$, and $n \in \{2^{2p+1}pB,\ 3^{2p+1}pB,\ \ldots\}$, any deterministic distributed algorithm for any of the following verification problems requires $\Omega((\frac{n}{pB})^{\frac{1}{2}-\frac{1}{2(2p+1)}})$ time on some $\Theta(n)$-vertex graph of diameter $2p+2$ in the B model: Hamiltonian cycle, spanning tree, and simple path verification.*

We first prove the lower bound of the Hamiltonian cycle problem and later extend to other problems.

6.1 Lower Bound of Hamiltonian Cycle Verification Problem

We construct $G(\Gamma, 2, p)'$ from $G(\Gamma, 2, p)$ by adding edges and vertices to $G(\Gamma, 2, p)$. We argue that the Simulation Theorem (cf. Theorem 3.1) holds on $G(\Gamma, 2, p)'$ as well. Note that since $d = 2$, $\mathcal{T}$ is a binary tree. Let m be $d^p - 1$.

First we add edges in such a way that the subgraph induced by the vertices $\{v_0^1, \ldots, v_0^\Gamma\}$ is a clique and the subgraph induced by the vertices $\{v_m^1, \ldots, v_m^\Gamma\}$ is a clique as well. Observe that the Simulation Theorem holds on $G(\Gamma, 2, p)$ with this change since it only relies on Lemma 3.4 which is true on the modified graph as well.

Now we add edges (u_i^p, u_{i+1}^p) for all $0 \leq i \leq m-1$. Thus we shorten the distance between each pair of nodes u_i^p and u_{i+1}^p from at most three (using two spoke edges and one path-edge from the according $\mathcal{P}^l$) to one (red dashed edges in Figure 8). This will affect the lower bound by at most a constant factor. Now for each node u_i^l we add a path of length $p-l+1$ containing $p-l$ new nodes connecting u_i^l to $u_{i \cdot d^{p-l}+1}^p$ (green dotted paths/nodes in Figure 8). This will increase the number of messages needed in Lemma 3.4 by a factor of two. Thus, the Simulation Theorem still holds.

Further, we add the edges $(v_{m-1}^{\Gamma-2}, u_0^p)$, $(v_m^{\Gamma-3}, u_0^p)$ and (v_m^Γ, u_0^0). This will again increase the number of messages needed in Lemma 3.4 by a constant factor of two and thus the Simulation Theorem still holds.

Finally, we add the following three edges (u_0^0, v_m^Γ), $(s, v_{m-1}^{\Gamma-2})$, and $(s, v_{m-1}^{\Gamma-3})$. Adding these three edges will increase the number of messages needed in Lemma 3.4 by at most three and thus the Simulation Theorem still holds. This completes the description of $G(\Gamma, 2, p)'$.

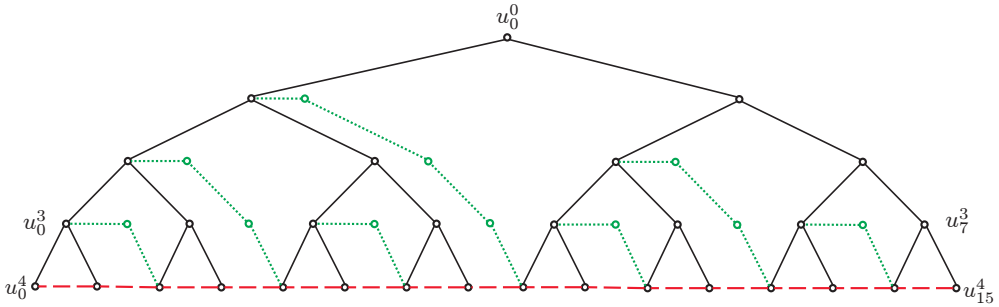

Figure 8: Example of the modification of the tree-part of $G(\Gamma, 2, p)$ in the case $p = 4$. The red dashed edges are new edges (u_i^p, u_{i+1}^p) and the green dotted edges form new paths between nodes connecting u_i^l to $u_{i \cdot d^{p-l}+1}^p$.

To simplify and shorten the proof, we do some preparation. First, we consider strings x and y of length b and define Γ to be $2 + 12b$ – this changes the bound only by a constant factor. Now, from x and y, we construct strings of length m (we assume m to be even)

$$x' := 1x_101x_101x_201x_201 \dots 01x_b01x_b01\overline{x_1}01\overline{x_1}01 \dots 01\overline{x_b}01\overline{x_b}010,$$
$$y' := 1y_101y_101y_201y_201 \dots 01y_b01y_b01\overline{y_1}01\overline{y_1}01 \dots 01\overline{y_b}01\overline{y_b}010$$

where $\overline{x_i}$ and $\overline{y_i}$ denote negations of x_i and y_i respectively.

Now we construct H in five stages. In the first stage we create some short paths that we call lines. In the next two stages we construct from these lines two paths S_1 and S_2 by connecting the lines in special ways with each other (the connections depend on the input strings). In the fourth stage we construct a path S_3 that will connect the leftover lines with each other. These three paths, S_1, S_2, and S_3, will cover all nodes. The final stage is to connect all three paths together.

We will construct these paths in such a way that, the input strings are equal if and only if the resulting graph H is a Hamiltonian cycle. Later we will observe that in the case the strings are equal all three paths will look like disjoint paths when using the graph layout of Figure 3. The formal description of the five stages will be accompanied by a small example in Figures 9 and 10. Here, $x = 01 = y$.

Stage 1 We create the lines by marking most path edges (to be more precise, all edges (v_j^i, v_{j+1}^i) for all $i \in [1, \Gamma]$ and $j \in \{2, \dots, m-2\}$ for $j \in \{1, \dots, m-1\}$) as participating in subgraph H. In addition we add the edges (v_{m-1}^1, v_m^1) and (v_0^1, v_1^1) to H. These basic elements are called lines now (see Figure 9).

Stage 2 Define path S_1 as follows. All spoke edges incident to $\mathcal{H}^1$ are marked as *not* participating in H, except those incident to s and r. For each bit x_i', $1 \le i \le \Gamma$, vertex s indicates that the edge (v_0^i, v_0^{i+1}) participates in H if and only if $x_i' = 1$. Similarly, for each bit y_i', $1 \le i \le \Gamma$, vertex r

indicates that the spoke edge (v_m^i, v_m^{i+1}) participates in H if and only if $y'_i = 0$. Furthermore for $2 \leq i \leq \Gamma$ each edge (v_0^i, v_1^i) participates in H if and only if $x'_{i-1} \neq x'_i$. Similarly for $2 \leq i \leq \Gamma$ each edge (v_{m-1}^i, v_m^i) participates in H if and only if $y'_{i-1} \neq y'_i$. In addition we let edges (v_0^1, v_1^1) and (v_{m-1}^1, v_m^1) participate in H. We denote the path that results from connecting the lines according to the rules above by S_1. An example is given in Figure 9.

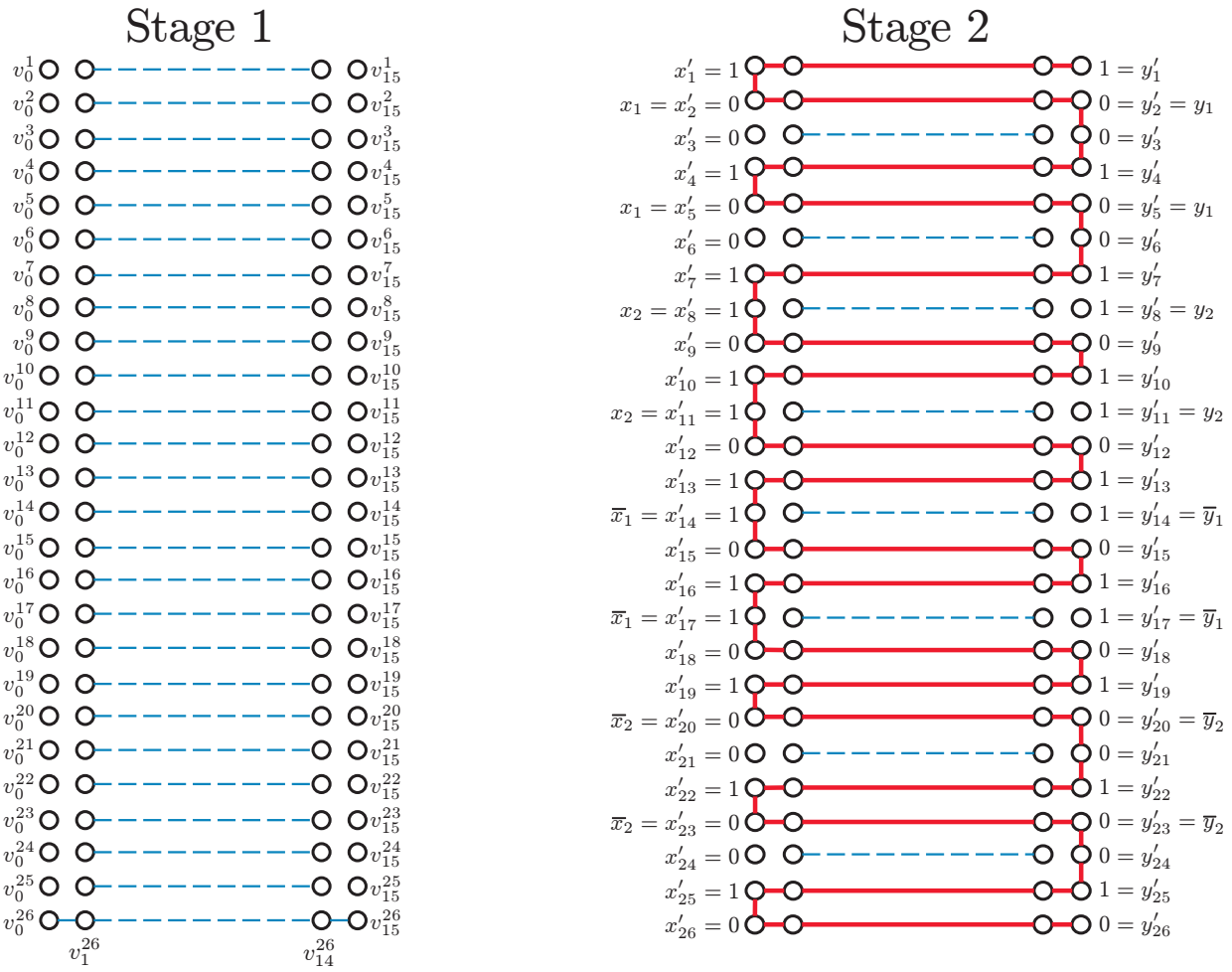

Figure 9: Example of the reduction using input strings $x = 01 = y$. Note that Γ is $12 \cdot 2 + 2 = 26$ and we use $d = 2$ and $p = 4$. In Stage 1, we add blue paths to H, that are displayed by blue dashed lines. In Stage 2, we create S_1, the red-colored path that looks like a snake.

Stage 3 Define S_2 as follows. We connect the lines not covered in Stage 3 (except the tree $\mathcal{T}$) and those nodes that are not covered by any path or line. In particular, on the left side of the graph, for $0 \leq i \leq 2b$, we do the following.

- If $x'_{2+6i} = 0$ (and thus $x'_{5+6i} = 0$ due to the definition of x') then edges (v_1^{3+6i}, v_0^{3+6i}), (v_0^{3+6i}, v_0^{6+6i}) and (v_0^{6+6i}, v_1^{6+6i}) are indicated to participate in H.

- If $x'_{2+6i} = 1$ (and thus $x'_{5+6i} = 1$ due to the definition of x'), edges $(v_m^{3+6i}, v_{m-1}^{3+6i})$, (v_1^{3+6i}, v_1^{6+6i}) and $(v_{m-1}^{6+6i}, v_m^{6+6i})$ will participate in H.

On the right side of the graph, for $0 \leq i \leq 2b$ we indicate the following edges to participate in H:

- $(v_m^{5+6i}, v_m^{2+6(i+1)})$ if $y'_{5+6i} = 0$ and $y'_{2+6(i+1)} = 0$.
- $(v_m^{5+6i}, v_{m-1}^{3+6(i+1)})$ if $y'_{5+6i} = 0$ and $y'_{2+6(i+1)} = 1$.
- $(v_{m-1}^{6+6i}, v_m^{2+6(i+1)})$ if $y'_{5+6i} = 1$ and $y'_{2+6(i+1)} = 0$.
- $(v_{m-1}^{6+6i}, v_{m-1}^{3+6(i+1)})$ if $y'_{5+6i} = 1$ and $y'_{2+6(i+1)} = 1$.

We denote the path that results from connecting lines according to the rules above by S_2. An example is given in Figure 10.

Stage 4 We include edges of the modified tree in a canonical way in H to form a path S_3 that covers all nodes in $\mathcal{T}$ and starts and ends at u_0^p and u_0^0 respectively, as follows. For all odd i such that $0 \leq i \leq m-1$, we include the edge (u_i^p, u_{i+1}^p) in H. For all $0 \leq l \leq p-1$ and all $0 \leq i \leq d^l$ we include the edges $(u_i^l, u_{i \cdot d+1}^{l+1})$ in H, and if i is odd, we also include the path connecting u_i^l to $u_{i \cdot d^{p-l}+1}$ in H. An example is given in Figure 11.

Stage 5 We now connect end points of the three paths. Let us investigate the six endpoints of the three paths.

- End points of S_3 are u_0^0 and u_0^p (see Figure 11).
- Path S_2 has both endpoints on the r-side (see Figure 10). Let us denote these endpoints by e_1 and e_2. Depending on the input strings, endpoint e_1 is either v_{m-1}^3 or v_m^2 and the other endpoint e_2 is either $v_{m-1}^{\Gamma-2}$ or $v_{m-1}^{\Gamma-3}$.
- The endpoints of S_1 are both on the r-side (see Figure 9): v_m^Γ and v_m^1.

Now we connect those endpoints in the following way.

- Connect S_1 and S_2, each at one endpoint on the r-side, by letting edge (e_1, v_m^1) participate in H.
- We connect the endpoint u_0^0 of S_3 to the endpoint v_m^Γ of S_1 by marking an edge (u_0^0, v_m^Γ) as participating.
- Connect the endpoint e_2 to v and v to the endpoint u_0^p of S_3 by including the corresponding edges of $G(\Gamma, 2, p)'$ in H.

An example is given in Figure 10.

If the strings are equal then the result is a Hamiltonian cycle since it contains the paths S_1, S_2, and S_3 that will be three disjoint paths (connected to be a cycle) that cover all nodes. Now we prove that if the strings are not equal, H will not be a Hamiltonian cycle. Let i be a position in which X^s and X^r differ. Let us consider the case that $x_i = 0$ and $y_i = 1$. Then the sequence $x'_{1+6i}, \ldots, x'_{6+6i}$ will be 100100 while the sequence $y'_{1+6i}, \ldots, y'_{6+6i}$ will be 110110. When we look

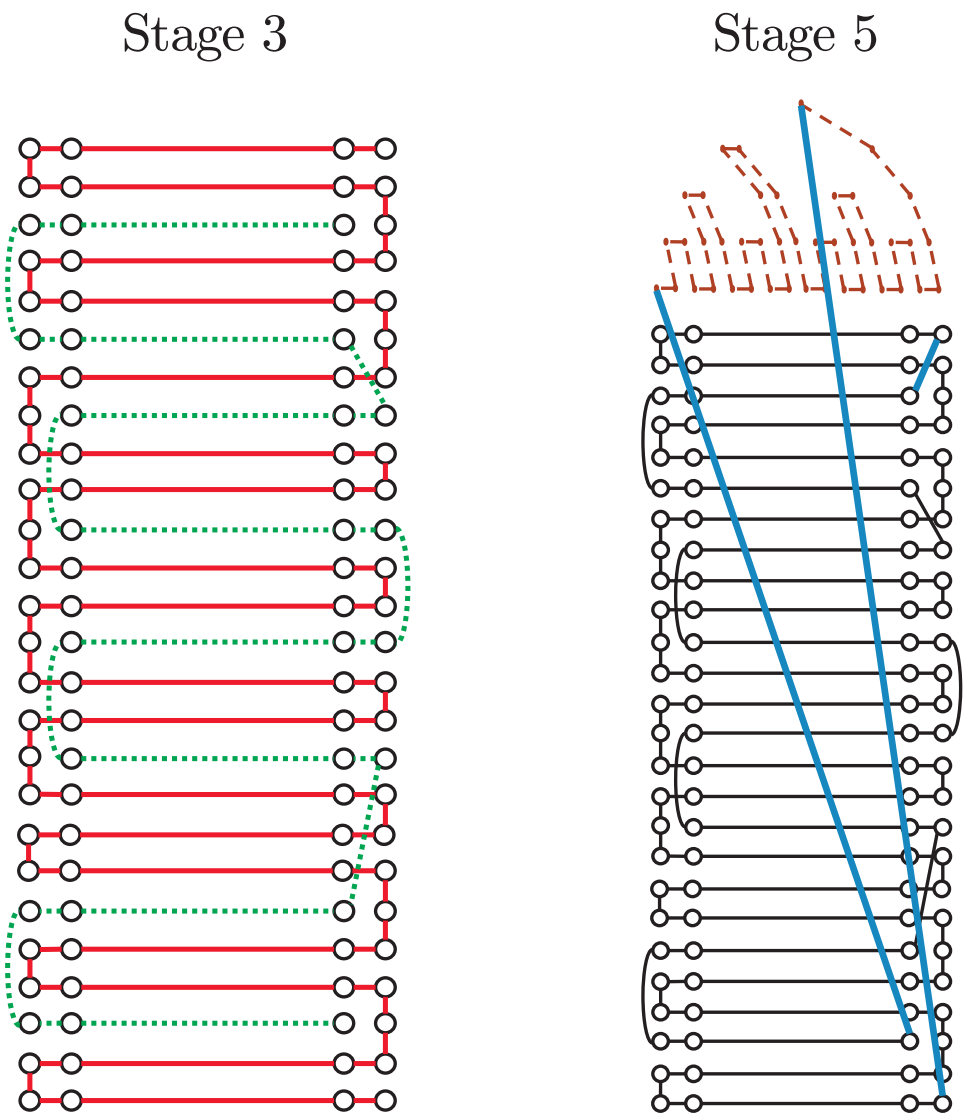

Figure 10: Continuation of the example started in Figure 9. In Stage 3, we add S_2 in dotted green (dotted lines). S_3 is displayed in dashed brown and added in stage four. Finally we connect S_1, S_2 and S_3 by bold blue edges to a Hamiltonian cycle in Stage 5.

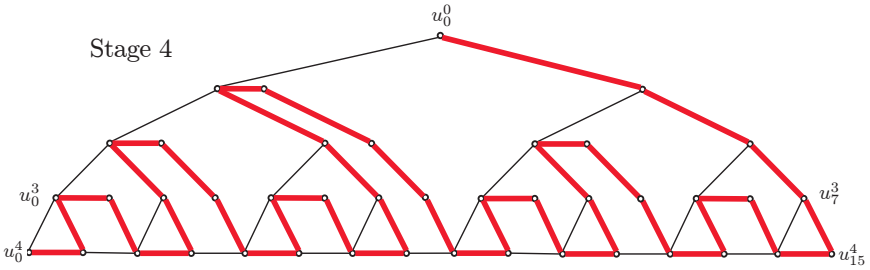

Figure 11: The modified tree for $p = 4$ and $d = 2$. The red bold edges form path S_3.

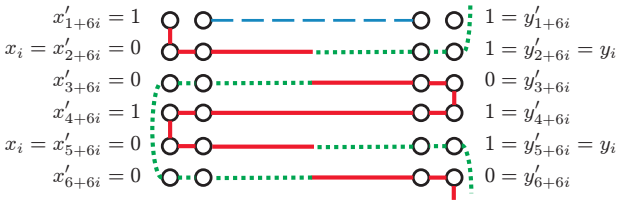

Figure 12: Example of the case that $x_i = 0$ and $y_i = 1$.

at the part of the graph H corresponding to this sequence (see Figure 12), we see that H can not be a cycle and thus not a Hamiltonian cycle: due to $y'_{2+6i} = 0$ and $x'_{2+6i} = 1$ there are no edges on the s- nor r-side of level $2 + 6i$ connecting the part of S_1 below level $2 + 6i$ to the part of S_1 above level $2 + 6i$. There will also be no edges of S_2 that accidentally connect those two parts to each other. The case that $x_i = 1$ and $y_i = 0$ is treated the same way: due to the construction of x' and y' the sequence $x'_{6b+1+6i}, \ldots, x'_{6b+6+6i}$ is 100100 and $y'_{6b+1+6i}, \ldots, y'_{6b+6+6i}$ will be 110110 and we can use exactly the same argument as before.

Now consider an algorithm $\mathcal{A}_{ham}$ for the Hamiltonian cycle verification problem. When $\mathcal{A}_{ham}$ terminates, vertex s determines its output for the equality problem by stating that both input strings are equal if and only if $\mathcal{A}_{ham}$ verified that H is a Hamiltonian cycle.

Hence a fast algorithm for the Hamiltonian cycle problem on $G(\Gamma, 2, p)'$ can be used to correctly solve the given instance of the equality problem on $G(\Gamma, 2, p)'$ faster. This contradicts the lower bound of the equality verification problem, which holds for all d (here we used $d = 2$).

6.2 Lower Bound of Spanning Tree and Path Verification Problems

The remaining two deterministic lower bounds follow from the lower bound of the Hamiltonian cycle verification problem, as follows.

Spanning Tree Verification Problem We reduce Hamiltonian cycle verification to spanning tree verification using $O(D)$ rounds using the following observation: H is a Hamiltonian cycle if and only if every vertex has degree exactly two and $H \setminus e$, for any edge e in H, is a spanning tree.

Therefore, to verify that H is a Hamiltonian cycle, we first check whether every vertex has degree exactly two in H. If this is not true then H is not a Hamiltonian cycle. This part needs

$O(D)$ rounds. Next, we check if $H \setminus \{e\}$, for any edge e in H, is a spanning tree. We announce that H is a Hamiltonian cycle if and only if $H \setminus \{e\}$ is a spanning tree.

Simple Path Verification Problem Similar to the above proof, we reduce Hamiltonian cycle verification to path verification using $O(D)$ rounds using the following observation: H is a Hamiltonian cycle if and only if every vertex has degree exactly two and $H \setminus e$ is a path (without cycles).

7 Hardness of Distributed Approximation

In this section we show a time lower bound of $\Omega(\sqrt{n/(B \log n)})$ for Monte Carlo randomized approximation algorithms of many problems (defined in Section 2.5), as in the following theorem.

Theorem 7.1. *For any polynomial function $\alpha(n)$, numbers p, $B \geq 1$, and $n \in \{2^{2p+1}pB, 3^{2p+1}pB, \ldots\}$, there exists a constant $\epsilon > 0$ such that any $\alpha(n)$-approximation ϵ-error distributed algorithm for any of the following problems requires $\Omega((\frac{n}{pB})^{\frac{1}{2}-\frac{1}{2(2p+1)}})$ time on some $\Theta(n)$-vertex graph of diameter $2p+2$ in the B model: minimum spanning tree [10, 29], shallow-light tree [28], s-source distance [8], shortest path tree [10], minimum routing cost spanning tree [35], minimum cut [7], minimum s-t cut, shortest s-t path and generalized Steiner forest [15].*

In particular, for graphs with diameter $D = 4$, we get $\Omega((n/B)^{1/3})$ lower bound and for graphs with diameter $D = \log n$ we get $\Omega(\sqrt{n/(B \log n)})$. Similar analysis also leads to a $\Omega(\sqrt{n/B})$ lower bound for graphs of diameter n^δ for any $\delta > 0$, and $\Omega((n/B)^{1/4})$ lower bound for graphs of diameter three using the same analysis as in [10].

The main tool used in this section is the randomized lower bound of network verification problems defined in Section 2.4 and proved in Section 5.

The main proof idea is similar to the proof that the Traveling Salesman Problem on general graphs cannot be approximated within $\alpha(n)$ for any polynomial computable function $\alpha(n)$ (see, e.g., [34]): We will define a weighted graph G' in such a way that if the subgraph H satisfies the desired property then the approximation algorithm must return some value that is at most $f(n)$, for some function f. Conversely, if H does not satisfy the property, the approximation algorithm will output some value that is strictly more than $f(n)$. Thus, we can distinguish between the two cases.

To highlight the main idea, we first prove the theorem for the minimum spanning tree problem in the next subsection. Proofs of other problems are in Subsection 7.2.

7.1 Lower Bound of Approximating the Minimum Spanning Tree

Recall that in the minimum spanning tree problem, we are given a connected graph G and we want to compute a minimum spanning tree (i.e., a spanning tree of minimum weight). At the end of the process each vertex knows which edges incident to it are in the output tree.

Let $\mathcal{A}_\epsilon$ be an $\alpha(n)$-approximation ϵ-error algorithm for the minimum spanning tree problem. We show that $\mathcal{A}_\epsilon$ can be used to solve the spanning connected subgraph verification problem using the same running time (thus the lower bound proved in Theorem 5.1 applies).

To do this, construct a weight function on edges in G, denoted by ω, by assigning weight 1 to all edges in H and weight $n\alpha(n)$ to all other edges. Note that constructing ω does not need any

communication since each vertex knows which edges incident to it are members of H. Call this weighted graph G'. Now we find the weight W of the $\alpha(n)$-approximated minimum spanning tree of G' using $\mathcal{A}_\epsilon$ and announce that H is a spanning connected subgraph if and only if W is less than $n\alpha(n)$.

We will show that the weighted graph G' has a spanning tree of weight less than $n\alpha(n)$ if and only if H is a spanning connected subgraph of G and thus the algorithm above is correct. Suppose that H is a spanning connected subgraph. Then, there is a spanning tree that is a subgraph of H and has weight $n-1 < n\alpha(n)$ since $\alpha(n) \geq 1$. Thus the minimum spanning tree has weight less than $n\alpha(n)$. Conversely, suppose that H is not a spanning connected subgraph. Then, any spanning tree of G' must contain an edge not in H. Therefore, any spanning tree has weight at least $n\alpha(n)$ as claimed.

7.2 Lower Bounds of Other Problems

We now prove the remaining lower bounds.

Shallow-light tree problem The lower bound for the shallow-light tree problem follows immediately from the lower bound of the MST problem when we set the length of every edge to be one and radius requirement to be n. In this case, the spanning tree satisfies the radius requirement and so the minimum-weight shallow-light tree becomes the minimum spanning tree.

s-source distance and shortest path tree problems We construct the graph G' as in Subsection 7.1 and the lower bounds follow in a similar way: H is a spanning connected subgraph if and only if the distance from s to every node is at most $n-1$ (i.e., $\mathcal{A}$ has approximated the distance to be at most $(n-1)\alpha(n)$), which is true if and only if the shortest path spanning tree contains only edges of weight one (i.e., the total weight of the shortest path spanning tree is at most $(n-1)\alpha(n)$).

Minimum routing cost spanning tree problem We modify the construction of G' in Subsection 7.1 as follows. We assign weight one to edges in H and $n^3\alpha(n)$ to other edges. Observe that if H is a spanning connected subgraph, the routing cost between any pair will be at most $n-1$ and thus the cost of the $\alpha(n)$-approximation minimum routing cost spanning tree will be at most $(n-1)\binom{n}{2}\alpha(n) < n^3\alpha(n)$. Conversely, if H is not a spanning connected subgraph, some pair of nodes will have routing cost at least $n^3\alpha(n)$ and thus the minimum routing cost spanning tree will cost at least $n^3\alpha(n)$.

Minimum cut problem We define $\bar{G}'$ by assigning weight one to all edges in $\bar{H} = (V, E(G) \setminus E(H))$ and $n\alpha(n)$ to all other edges and use the fact that $\bar{H}$ is a cut if and only if $\bar{G}'$ has a minimum cut of weight at most $n-1$, i.e., $\mathcal{A}$ outputs a value of at most $(n-1)\alpha(n)$.

Minimum s-t cut problem The reduction is the same as in the case of the minimum cut problem. Observe that s and t are *not* connected in H if and only if $\bar{H}$ is a s-t cut which in turn is the case if and only if $\bar{G}'$ has a minimum $s-t$ cut of weight $n-1$. Thus, the lower bound of s-t cut verification problem implies the lower bound of this problem.

Shortest s-t path problem We again construct G' as in Subsection 7.1. Observe that s and t are in the same connected component in H if and only if the distance between s to t in G' is at most $n-1$, i.e., $\mathcal{A}$ outputs a value of at most $(n-1)\alpha(n)$. The lower bound follows from the lower bound of s-t connectivity verification problem.

Generalized Steiner forest problem We will reduce from the lower bound of s-t connectivity. We have only one set $V_1 = \{s, r\}$. Construct G' as in Subsection 7.1. Observe that the minimum generalized Steiner forest will have weight at most $n-1$ if H is s-t connected and weight at least $n\alpha(n)$ otherwise. (Recall that G is assumed to be connected in the problem definition.)

8 Tightness of Lower Bounds of Verification Problems

We note that almost all lower bounds of verification problems stated so far are almost tight. To show this we will present deterministic $O(\sqrt{n}\log^* n + D)$-time algorithms for all verification problems except the least-element list verification problem. This upper bound almost matches the $\tilde{\Omega}(\sqrt{n})$ lower bounds shown in previous sections. Our main tool is the MST algorithm by Kutten and Peleg [21] and the connected component algorithm by Thurimella [33, Algorithm 5].

Recall that in the MST problem, we are given a weighted network G (that is the weight of each edge is known to the nodes incident to it) and we want to find a minimum spanning tree (for each edge e, nodes incident to it know whether e is in the MST or not.) Kutten and Peleg [21] showed that this problem can be solved by a $O(\sqrt{n}\log^* n + D)$-time distributed deterministic algorithm.

We also note the following connected component algorithm by Thurimella [33]. Given a subgraph H of G, the algorithm outputs a label $\ell(v)$ for each node v such that for any two nodes u and v, $\ell(u) = \ell(v)$ if and only if u and v are in the same connected component. Theorem 6 in [33] states that the distributed time complexity of this algorithm is $O(D + f(n) + g(n) + \sqrt{n})$ where $f(n)$ and $g(n)$ are the distributed time complexities of finding a MST and a $\sqrt{n}$ -dominating set, respectively. Due to [21] we have that $f(n) = g(n) = O(D + \sqrt{n}\log^* n)$.

We are now ready to present algorithms for our verification problems.

Spanning connected subgraph, spanning tree, cycle containment and connectivity verification problems We construct a weighted graph G' by assigning weight zero to all edges in H and weight one to other edges (for each edge e, nodes incident to it know its weight). Observe the followings.

- H is a spanning tree if and only if the MST of G' has cost zero.
- H is a spanning connected subgraph if and only if the MST of G' has cost zero.
- H contains no cycle if and only if all edges in H are in the MST of G', i.e., the cost of the MST of G' is $n - 1 - |E(H)|$ where $|E(H)|$ is the number of edges in H.
- H is connected if and only if there are $|V(H)| - 1$ edges in the MST that have cost zero, where $V(H)$ is the set of nodes incident to some edges in H. This is because all edges in a spanning forest of H can be used in the MST and there are less than $V(H) - 1$ such edges if and only if H is not connected.

Thus, we can verify these properties of H by finding a minimum spanning tree of G' using the $O(\sqrt{n}\log^* n + D)$-time algorithm of Kutten and Peleg [21].

Cut verification problem To verify if H is a cut, we simply verify if G after removing the edges in H, i.e. $H' = (V, E(G) \setminus E(H))$, is connected.

s-t connectivity verification problem We simply run Thurimella's algorithm (as explained above) and verify whether s and t are in the same connected component by verifying whether $\ell(s) = \ell(t)$.

Edge on all paths verification problem Observe that e lies on all paths between u and v if and only if u and v are disconnected in $H \setminus \{e\}$. Thus, we can use the s-t connectivity verification algorithm above to check this.

s-t cut verification problem To verify if H is a s-t cut, we simply verify s-t connectivity of G after removing the edges in H (i.e., $H' = (V, E(G) \setminus E(H))$).

e-cycle verification problem To verify if e is in some cycle of H, we simply verify s-t connectivity of $H' = H \setminus \{e\}$ where s and t are the end nodes of e.

Bipartiteness verification problem We run Thurimella's algorithm to find the connected components of H. We note that this algorithm can in fact output a rooted spanning tree of each connected component of H and make each node knows its level in such a tree. This level implies a natural two-coloring of nodes in H. Now all nodes check if their neighbors have a color different from their own color. They will have a different color if and only if H is bipartite.

9 Conclusion

We initiate the systematic study of verification problems in the context of distributed network algorithms and present a uniform lower bound for several problems. We also show how these verification bounds can be used to obtain lower bounds on exact and approximation algorithms for many problems. Our techniques exploit well-known bounds in communication complexity to show lower bounds in distributed computing. Our techniques give a general and powerful methodology for showing non-trivial lower bounds for various problems in distributed computing.

Several problems remain open. A general direction for extending all of this work is to study similar verification problems in special classes of graphs, e.g., a complete graph. A few specific open questions include proving better lower or upper bounds for the problems of shortest s-t path, single-source distance computation, shortest path tree, s-t cut, minimum cut. (Some of these problems were also asked in [7].) Also, showing randomized bounds for Hamiltonian path, spanning tree, and simple path verification remains open.

References

[1] L. Babai, P. Frankl, and J. Simon, *Complexity classes in communication complexity theory (preliminary version)*, in FOCS, 1986, pp. 337–347.

[2] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar, *An information statistics approach to data stream and communication complexity*, J. Comput. Syst. Sci., 68 (2004), pp. 702–732. Also in FOCS'02.

[3] A. Chattopadhyay and T. Pitassi, *The Story of Set Disjointness*, SIGACT News, 41 (2010), pp. 59–85.

[4] E. Cohen, *Size-Estimation Framework with Applications to Transitive Closure and Reachability*, J. Comput. Syst. Sci., 55 (1997), pp. 441–453. Also in FOCS'94.

[5] 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*, in STOC, 2011, pp. 363–372.

[6] D. P. Dubhashi, F. Grandioni, and A. Panconesi, *Distributed Algorithms via LP Duality and Randomization*, in Handbook of Approximation Algorithms and Metaheuristics, Chapman and Hall/CRC, 2007.

[7] M. Elkin, *Distributed approximation: a survey*, SIGACT News, 35 (2004), pp. 40–57.

[8] ——, *Computing almost shortest paths*, ACM Transactions on Algorithms, 1 (2005), pp. 283–323. Also in PODC'01.

[9] ——, *A faster distributed protocol for constructing a minimum spanning tree*, J. Comput. Syst. Sci., 72 (2006), pp. 1282–1308. Also in SODA'04.

[10] ——, *An Unconditional Lower Bound on the Time-Approximation Trade-off for the Distributed Minimum Spanning Tree Problem*, SIAM J. Comput., 36 (2006), pp. 433–456. Also in STOC'04.

[11] S. Frischknecht, S. Holzer, and R. Wattenhofer, *Networks cannot compute their diameter in sublinear time*, in SODA, 2012.

[12] J. A. Garay, S. Kutten, and D. Peleg, *A Sublinear Time Distributed Algorithm for Minimum-Weight Spanning Trees*, SIAM J. Comput., 27 (1998), pp. 302–316. Also in FOCS '93.

[13] M. R. Henzinger, P. Raghavan, and S. Rajagopalan, *Computing on data streams*, in External memory algorithms, J. M. Abello and J. S. Vitter, eds., American Mathematical Society, Boston, MA, USA, 1999, pp. 107–118.

[14] B. Kalyanasundaram and G. Schnitger, *The Probabilistic Communication Complexity of Set Intersection*, SIAM J. Discrete Math., 5 (1992), pp. 545–557.

[15] M. Khan, F. Kuhn, D. Malkhi, G. Pandurangan, and K. Talwar, *Efficient distributed approximation algorithms via probabilistic tree embeddings*, in PODC, 2008, pp. 263–272.

[16] L. Kor, A. Korman, and D. Peleg, *Tight Bounds For Distributed MST Verification*, in STACS, 2011, pp. 69–80.

[17] A. Korman and S. Kutten, *Distributed verification of minimum spanning trees*, Distributed Computing, 20 (2007), pp. 253–266. Also in PODC'06.

[18] F. Kuhn, T. Moscibroda, and R. Wattenhofer, *What cannot be computed locally!*, in PODC, 2004, pp. 300–309.

[19] F. Kuhn and R. Oshman, *The complexity of data aggregation in directed networks*, in DISC, 2011.

[20] E. Kushilevitz and N. Nisan, *Communication complexity*, Cambridge University Press, New York, NY, USA, 1997.

[21] S. Kutten and D. Peleg, *Fast Distributed Construction of Small k-Dominating Sets and Applications*, J. Algorithms, 28 (1998), pp. 40–66. Also in PODC'95.

[22] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann Publishers, 1 ed., 1991.

[23] N. Linial, *Locality in distributed graph algorithms*, SIAM J. Comput., 21 (1992), pp. 193–201.

[24] Z. Lotker, B. Patt-Shamir, and D. Peleg, *Distributed MST for constant diameter graphs*, Distributed Computing, 18 (2006), pp. 453–460. Also in PODC'01.

[25] M. Luby, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Comput., 15 (1986), pp. 1036–1053. Also in STOC'85.

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

[27] D. Nanongkai, A. Das Sarma, and G. Pandurangan, *A tight unconditional lower bound on distributed randomwalk computation*, in PODC, 2011, pp. 257–266.

[28] D. Peleg, *Distributed computing: a locality-sensitive approach*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[29] D. Peleg and V. Rubinovich, *A Near-Tight Lower Bound on the Time Complexity of Distributed Minimum-Weight Spanning Tree Construction*, SIAM J. Comput., 30 (2000), pp. 1427–1442. Also in FOCS'99.

[30] R. Raz and B. Spieker, *On the "log rank" – Conjecture in Communication Complexity*, in FOCS, 1993.

[31] A. A. Razborov, *On the Distributional Complexity of Disjointness*, Theor. Comput. Sci., 106 (1992), pp. 385–390. Also in ICALP'90.

[32] R. E. Tarjan, *Applications of path compression on balanced trees*, J. ACM, 26 (1979), pp. 690–715.

[33] R. Thurimella, *Sub-Linear Distributed Algorithms for Sparse Certificates and Biconnected Components*, J. Algorithms, 23 (1997), pp. 160–179. Also in PODC'95.

[34] V. V. Vazirani, *Approximation Algorithms*, Springer, July 2001.

[35] B. Y. Wu, G. Lancia, V. Bafna, K.-M. Chao, R. Ravi, and C. Y. Tang, *A polynomial-time approximation scheme for minimum routing cost spanning trees*, SIAM J. Comput., 29 (1999), pp. 761–778. Also in SODA'98.

[36] A. C.-C. Yao, *Probabilistic Computations: Toward a Unified Measure of Complexity*, in FOCS, 1977, pp. 222–227.

[37] ——, *Some complexity questions related to distributive computing (preliminary report)*, in STOC, 1979, pp. 209–213.