Distributed Verification and Hardness of Distributed Approximation[*]

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

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 in 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 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.

[*]The extended abstract of this paper appears in 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 & College of Computing, Georgia Institute of Technology, Atlanta, GA, USA. 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).

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/decentralized 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 $O(\log n)$ [18], but not in time $\Omega(\sqrt{\log n/\log\log n})$ [13] (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 [20].

But, can we *verify* whether an existing 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 [20] 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 [20, 19]. 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 [4] and Dubhashi et al. [3], and the work of [10] 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 [16] 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 [7].

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 [7]. 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 [13] 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

[1]Such problems have been studied in the sequential setting, e.g., Tarjan[22] studied verification of MST.

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 ([20, 19]). 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. [11] 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 [12]). Note that distributed *construction* of MST has rather similar lower and upper bounds [21, 8]. 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.

1.3 Our Contributions

In this paper, our main contributions are two fold. 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. Section 4). These bounds apply to *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 ($D = O(\log n)$). (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 $\Omega(\sqrt{n/(B\log n)}+$

Diameter D	Previous lower bound for MST and shortest-path tree [7] (for exact algorithms, use $\alpha = 1$)	New lower bound for MST, shortest-path tree and all problems in Fig. 2.
n^δ, $0 < \delta < 1/2$	$\Omega(\sqrt{\frac{n}{\alpha B}})$	$\Omega(\sqrt{\frac{n}{B}})$
$\Theta(\log n)$	$\Omega(\sqrt{\frac{n}{\alpha B \log n}})$	$\Omega(\sqrt{\frac{n}{B \log n}})$
Constant ≥ 3	$\Omega((\frac{n}{\alpha B})^{\frac{1}{2} - \frac{1}{2D-2}})$	$\Omega((\frac{n}{B})^{\frac{1}{2} - \frac{1}{2D-2}})$
4	$\Omega((\frac{n}{\alpha B})^{1/3})$	$\Omega((\frac{n}{B})^{1/3})$
3	$\Omega((\frac{n}{\alpha B})^{1/4})$	$\Omega((\frac{n}{B})^{1/4})$

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 [7] while those in the second column are for these problems and many problems listed in Fig. 2. We note that these bounds almost match the $O(\sqrt{n}\log^* n + D)$ upper bound for the MST problem [8, 15] and are independent of the approximation-factor α.

D) (cf. Section 6). However, this bound applies to only deterministic algorithms. This result strengthens the lower bound result of MST verification by Kor et al. [11]. 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 [20], 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 problem, we also show deterministic lower bounds for other verification problems, including *Hamiltonian cycle* and *simple path*.

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 all the verification problems addressed here (cf. Appendix C).

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 5). 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 also ($\alpha = 1$). All these hardness bounds hold for 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 [7]) and subsumes some well-established exact bounds (e.g., Peleg and Rubinovich lower bound for MST [21]) as well as shows new strong bounds (both for exact and approximate computation) for many other problems (e.g., minimum cut), thus answering some questions that were open earlier (see the survey by Elkin [4]).

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 [7], where α is the approximation factor. [7] 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 [15, 6], for any approximation factor $\alpha > 1$.

2 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 [7, 17, 21]. However, while previous results [7, 21] rely on counting the number of states to analyze the *mailing problem* (along with some sophisticated techniques for the variant, called *corrupted mail problem*, in the case of approximation algorithm lower bounds) and use Yao's method [26] (with appropriate input distributions) to get lower bounds for randomized algorithms, our results are achieved using the following three steps of simple reductions, as follows.

(Section 3) First, we reduce the lower bounds of problems in the standard communication complexity model [14] to the lower bounds of the equivalent problems in the "distributed version" of communication complexity. Specifically, we relate the *communication* lower bound from the standard communication complexity model [14] 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 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 [7]. We show a reduction from the standard model to the distributed model, the proof of which relies on certain observations similar to those used in previous results (e.g., [21]).

(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*, 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 [1]). Following the result well known in communication complexity [14], we show that the distributed version of this problem has an $\Omega(\sqrt{n/(B\log n)})$ lower bound on graphs of small diameter. We then reduce this problem to the verification problems using simple reductions similar to those used in data streams [9]. The set disjointness function yields randomized lower bounds and works for many problems (see Fig. 2), but it does not reduce to certain other problems such as spanning tree. To show lower bounds for these and a few other problems, we use a different function f called *equality*. However, this reduction yields only deterministic lower bounds for the corresponding verification problems.

(Section 5) 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., [24]): We convert a verification problem to an optimization problem by introducing edge weight 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) [4] and some that improve over known ones (e.g., MST and shortest path tree) [7]. 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.

Fig. 2 summarizes these reductions that will be proved in this paper. For brevity, we focus on the proofs towards the lower bound of approximating minimum spanning tree in the main paper. Other results can be found in the Appendix.

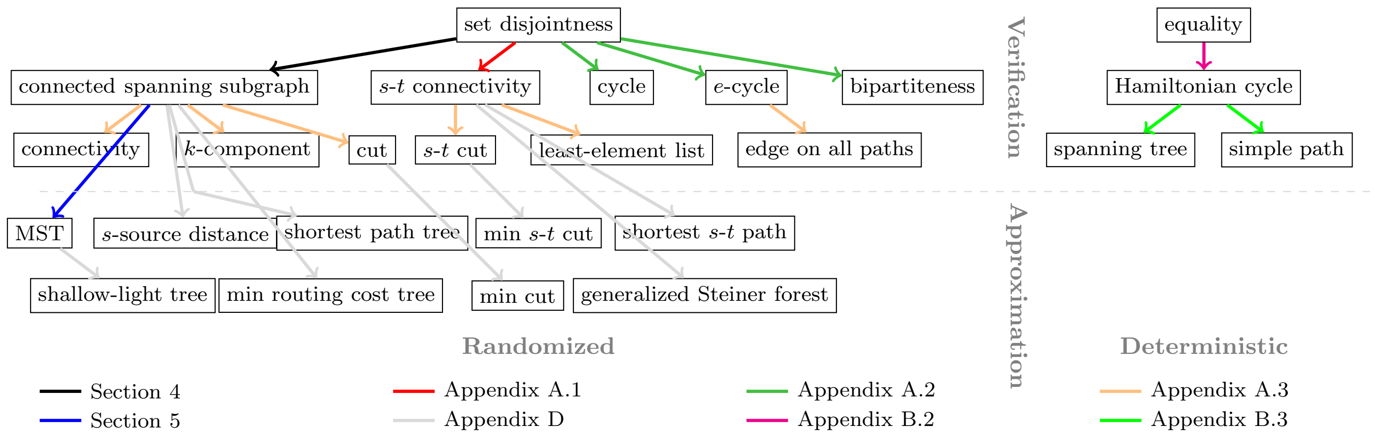


Figure 2: Problems and reductions between them to obtain randomized and deterministic lower bounds. For all problems, we obtain lower bounds as in Fig. 1

3 From Communication Complexity to Distributed Computing

Consider the following problem. There are two parties that have unbounded computational power. Each party receives a b-bit string, for some integer $b \geq 1$, denoted by $\bar{x}$ and $\bar{y}$ in $\{0,1\}^b$. They both want to together compute $f(\bar{x}, \bar{y})$ for some boolean function $f : \{0,1\}^b \times \{0,1\}^b \to \{0,1\}$. We consider two models of communication.

- *Direct communication:* This is the standard model in communication complexity. Two parties can communicate via a bidirectional edge of bandwidth one. We call the party receiving $\bar{x}$ *Alice*, and the other party *Bob*. At the end of the process, Bob will output $f(\bar{x}, \bar{y})$.
- *Communication through network $G(\Gamma, d, p)$:* Two parties are distinct vertices in a B model distributed network, called $G(\Gamma, d, p)$, for some parameters Γ, d, and p; the network has $\Theta(\Gamma d^p)$ vertices and a diameter of $\Theta(2p+2)$. (This network was first defined in [7] and described below.) We denote the vertex receiving $\bar{x}$ by s and the vertex receiving $\bar{y}$ by r. At the end of the process, r will output $f(\bar{x}, \bar{y})$.

We consider time lower bounds for *public coin randomized algorithms* under both models. In particular, we assume that all parties (Alice and Bob in the first model and all vertices in $G(\Gamma, d, p)$ in the second model) share a random bit string of infinite length. For any $\epsilon \geq 0$, we say that a randomized algorithm $\mathcal{A}$ is *ϵ-error* if for any input, it outputs the correct answer with probability at least $1-\epsilon$, where the probability is over all possible random bit strings. The running time of $\mathcal{A}$, denoted by $T_{\mathcal{A}}$, is the number of rounds in the worst case (over all inputs and random strings). Let $R_\epsilon^{cc-pub}(f)$ and $R_\epsilon^{G(\Gamma,d,p),s,r}(f)$ denote the best time complexity of ϵ-error algorithms in the models of direct communication and communication through graph $G(\Gamma, d, p)$, respectively. We are particularly interested in the case where we pick b to be Γ (for any choice of Γ). The rest of this section is devoted to showing 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.

Theorem 3.1. *For any Γ, d, p, B, $\epsilon \geq 0$, and function $f : \{0,1\}^\Gamma \times \{0,1\}^\Gamma \to \{0,1\}$, if*

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

then

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

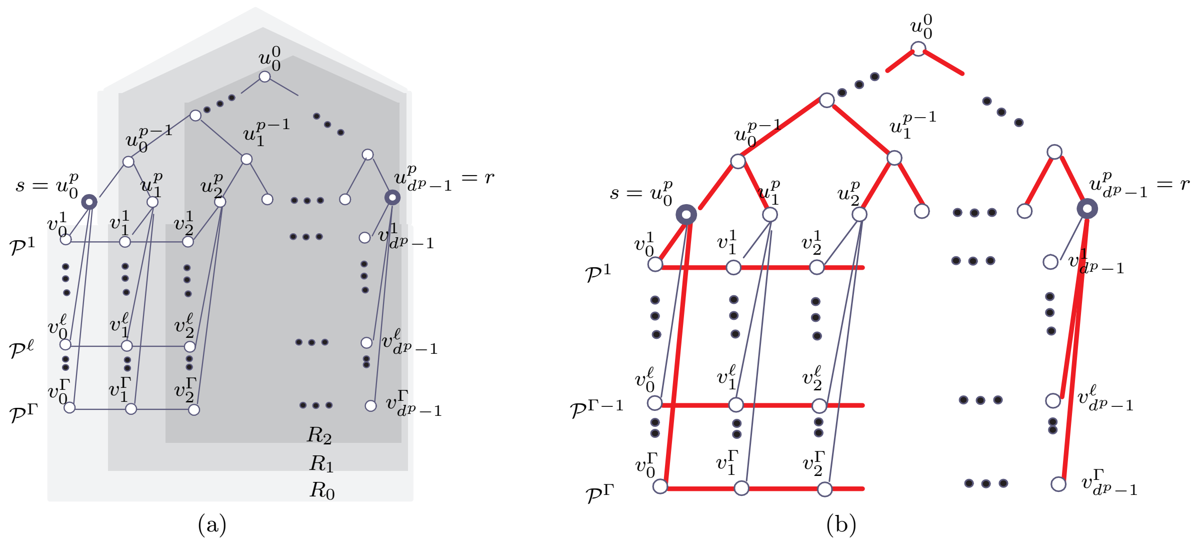


Figure 3: (a) The Graph $G(\Gamma, d, p)$ (here, $d = 2$). (b) Example of H for the spanning connected subgraph problem (marked with thick red edges) when $\bar{x} = 0...10$ and $\bar{y} = 1...00$.

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

The graph $G(\Gamma, d, p)$ **[7]:** 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 vertices, i.e., for $\ell = 1, 2, \ldots \Gamma$,

$$V(\mathcal{P}^\ell) = \{v_0^\ell, \ldots, v_{d^p-1}^\ell\} \quad \text{and}$$

$$E(\mathcal{P}^\ell) = \{(v_i^\ell, v_{i+1}^\ell) \mid 0 \le i < d^p - 1\}.$$

There is a tree, denoted by $\mathcal{T}$ having depth p where each non-leaf vertex has d children (thus, there are d^p leaf vertices). We denote the vertices 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 vertex u_j^p is connected to the corresponding path vertex v_j^ℓ by a *spoke* edge (u_j^p, v_j^ℓ). Finally, we set the two special vertices (which will receive input strings $\bar{x}$ and $\bar{y}$) as $s = u_0^p$ and $r = u_{d^p-1}^p$. Fig. 3(a) depicts this graph. The number of vertices in $G(\Gamma, d, p)$ its diameter are analyzed in [7], as follows.

Lemma 3.2. *[7] The number of vertices in $G(\Gamma, d, p)$ is $n = \Theta(\Gamma m)$ and its diameter is $2p + 2$.*

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

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

respectively. Define the i-left of the tree $\mathcal{T}$, denoted by $L_i(\mathcal{T})$, as the union of set $S = \{u_j^p \mid j \le d^p - 1 - i\}$ and all ancestors of all vertices in S in $\mathcal{T}$. Similarly, the i-right $R_i(\mathcal{T})$ of the tree $\mathcal{T}$ is the union of set $S = \{u_j^p \mid j \ge i\}$ 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 $i = 0$, the definition is slightly different; we set $L_0 = V \setminus \{r\}$ and $R_0 = V \setminus \{s\}$. See Fig. 3(a).

Let $\mathcal{A}$ be any *deterministic* distributed algorithm run on graph $G(\Gamma, d, p)$ for computing a function f. Fix any input strings $\bar{x}$ and $\bar{y}$ given to s and r respectively. Let $\varphi_\mathcal{A}(\bar{x}, \bar{y})$ denote the execution of $\mathcal{A}$ on $\bar{x}$ and $\bar{y}$. Denote the *state* of the vertex v at the end of round t during the execution $\varphi_{\mathcal{A}}(\bar{x}, \bar{y})$ by $\sigma_{\mathcal{A}}(v, t, \bar{x}, \bar{y})$. In two different executions $\varphi_{\mathcal{A}}(\bar{x}, \bar{y})$ and $\varphi_{\mathcal{A}}(\bar{x}', \bar{y}')$, a vertex reaches the same state at time t (i.e., $\sigma_{\mathcal{A}}(v, t, \bar{x}, \bar{y}) = \sigma_{\mathcal{A}}(v, t, \bar{x}', \bar{y}')$) 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*

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

is a vector of the states of the vertices of U at the end of round t of the execution $\varphi_{\mathcal{A}}(\bar{x}, \bar{y})$. We note the following crucial observation used in [21] and many later results.

Observation 3.3. *For any set $U \subseteq U' \subseteq V$, $C_{\mathcal{A}}(U, t, \bar{x}, \bar{y})$ can be uniquely determined by $C_{\mathcal{A}}(U', t-1, \bar{x}, \bar{y})$ 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 $\sigma_{\mathcal{A}}(v, t-1, \bar{x}, \bar{y})$ 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 $C_{\mathcal{A}}(U', t, \bar{x}, \bar{y})$. 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 $C_{\mathcal{A}}(U, t, \bar{x}, \bar{y})$. □

From now on, to simplify notation, when $\mathcal{A}$, $\bar{x}$ and $\bar{y}$ are clear from the context, we use C_{L_t} and C_{R_t} to denote $C_{\mathcal{A}}(L_t, t, \bar{x}, \bar{y})$ and $C_{\mathcal{A}}(R_t, t, \bar{x}, \bar{y})$, respectively. The lemma below states that C_{L_t} (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 $\bar{x}$ and $\bar{y}$. 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 $M_1^{R_{t-1}}, \ldots, M_{dp}^{R_{t-1}}$ sent by some vertices in R_{t-1} at time t such that*

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

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

Proof. We prove Eq. (2) only. (Eq. (1) is proved in exactly the same way.) Observe that all neighbors of all path vertices in R_t are in R_{t-1}. Similarly, all neighbors of all leaf vertices in $V(\mathcal{T}) \cap R_t$ are in R_{t-1}. Moreover, for any non-leaf tree vertex u_i^ℓ (for some 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}. For any $\ell < p$ and t, 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 Fig. 3(a), $u^{p-1}(R_1) = u_0^{p-1}$ and $u^{p-1}(R_2) = u_1^{p-1}$.) Finally, observe that for any i and ℓ, if u_{i-1}^ℓ is in R_t then all children of u_i^ℓ are in R_t (otherwise, all children of u_{i-1}^ℓ are not in R_t and so is u_{i-1}^ℓ, a contradiction). Thus, 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.

Proof of 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 graph $G(\Gamma, d, p)$. 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 only consider $T_\mathcal{A} < (d^p - 1)/2$, 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 $2dpT_\mathcal{A}$ communication bits, as follows.

Alice and Bob make $T_\mathcal{A}$ *iterations* of communications. Initially, Alice computes C_{L_0} which depends only on $\bar{x}$. Bob also computes C_{R_0} which depends only on $\bar{y}$. 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, Bob knows $C(R_{T_\mathcal{A}}, T_\mathcal{A}, \bar{x}, \bar{y})$. In particular, he knows the output of $\mathcal{A}$ (output by r) since he knows the state of r after $\mathcal{A}$ terminates. He thus outputs the output of r.

Since $\mathcal{A}_\epsilon$ is ϵ-error, the probability (over all possible shared random strings) that $\mathcal{A}$ outputs the correct value of $f(\bar{x}, \bar{y})$ is at least $1 - \epsilon$. Therefore, the communication protocol run by Alice and Bob is ϵ-error as well. Moreover, Alice and Bob communicates at most $2dpBT_\mathcal{A}$ bits. The theorem follows. □

4 Randomized Lower Bounds for Distributed Verification

In this section, we present randomized lower bounds for many verification problems for graph of various diameters, as shown in Fig. 1.

The general theorem is below. For brevity, in the main section of the paper, we prove the theorem only for the spanning connected subgraph verification problem. This will be useful later in proving many hardness of approximation results. In this problem, 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. Definitions of other problems and proofs of their lower bounds are in Appendix A.

Theorem 4.1. *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, connectivity, s-t connectivity, k-components, bipartiteness, cycle containment, e-cycle containment, cut, s-t cut, least-element list [2, 10], and edge on all paths.*

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 3 using the same analysis as in [7]. We note that the lower bound holds even in the public coin model where every vertex shares a random string. To prove the theorem, we need the lower bound for computing *set disjointness function.*

Definition 4.2 (Set Disjointness function)**.** Given two b-bit strings $\bar{x}$ and $\bar{y}$, the *set disjointness function*, denoted by $\mathtt{disj}(\bar{x},\bar{y})$, is defined to be 1 if the inner product $\langle \bar{x},\bar{y}\rangle$ is 0 (i.e., $x_i = 0$ or $y_i = 0$ for every $1 \le i \le b$) and 0 otherwise. We refer to the problem of computing $\mathtt{disj}$ function on $G(\Gamma, d, p)$ on Γ-bit input strings given to s and r by $\mathtt{DISJ}(G(\Gamma, d, p), s, r, \Gamma)$.

The following lemma is a consequence of Theorem 3.1 and the communication complexity lower bound of computing $\mathtt{disj}$.

Lemma 4.3. *For any Γ, d, p, there exists a constant $\epsilon > 0$ such that any ϵ-error algorithm solving $\mathtt{DISJ}(G(\Gamma, d, p),\ s,\ r,\ \Gamma)$ requires $\Omega(\min(d^p, \frac{\Gamma}{dpB}))$ time.*

Proof. If $R_\epsilon^{G(\Gamma,d,p),s,r}(\mathtt{disj}) \ge (d^p - 1)/2$ then $R_\epsilon^{G(\Gamma,d,p),s,r}(\mathtt{disj}) = \Omega(d^p)$ and we are done. Otherwise, Theorem 3.1 implies that $R_\epsilon^{cc-pub}(\mathtt{disj}) \le 2dpB \cdot R_\epsilon^{G(\Gamma,d,p),s,r}(\mathtt{disj})$. Now we use the fact that $R_\epsilon^{cc-pub}(\mathtt{disj}) = \Omega(\Gamma)$ for the function $\mathtt{disj}$ on Γ-bit inputs, for some $\epsilon > 0$ [**?**, **?**, **?**, **?**] (also see [14, Example 3.22] and references therein). It follows that $R_\epsilon^{G(\Gamma,d,p),s,r}(\mathtt{disj}) = \Omega(\Gamma/(dpB))$. ☐

The lower bound of spanning connected subgraph verification essentially follows from the following lemma.

Lemma 4.4. *For any Γ, $d \ge 2$ and p, there exists a constant $\epsilon > 0$ such that any ϵ-error distributed algorithm for spanning connected subgraph verification on graph $G(\Gamma, d, p)$ can be used to solve the $\mathtt{DISJ}(G(\Gamma, d, p),\ s,\ t,\ \Gamma)$ problem on $G(\Gamma, d, p)$ with 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 $\mathtt{DISJ}(G(\Gamma, d, p), s, t, \Gamma)$ problem with input strings $\bar{x}$ and $\bar{y}$. We use $\mathcal{A}$ to solve this instance of set disjointness problem 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, $1 \le i \le \Gamma$, 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, $1 \le i \le \Gamma$, vertex r indicates that the spoke edge $(r, v_{d^p-1}^i)$ participates in H if and only if $y_i = 0$. (See Fig. 3(b).)

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 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 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 $1 \le i \le \Gamma$ 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 $\bar{x}, \bar{y}$ 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. ☐

Using Lemma 4.3, we obtain the following result.

Corollary 4.5. *For any Γ, d, p, there exists a constant $\epsilon > 0$ such that any ϵ-error algorithm for 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 $\Omega(d^p)$ becomes

$$\Omega((n/(pB))^{\frac{1}{2}-\frac{1}{2(2p+1)}})\,.$$

Theorem 4.1 (for the case of spanning connected subgraph) follows.

5 Hardness of Distributed Approximation

In this section we show a time lower bound of $\Omega(\sqrt{n/(B\log n)})$ for approximation algorithms of many problems. For distributed approximation problems such as MST, we assume that a weight function $\omega : E \to \mathbb{R}^+$ associated with the graph assigns a nonnegative real weight $\omega(e)$ to each edge $e = (u, v) \in E$. Initially, the weight $\omega(e)$ is known only to the adjacent vertices, u and v. We assume that the edge weights are bounded by a polynomial in n (the number of vertices). It is assumed that B is large enough to allow the transmission of any edge weight in a single message.

We show the hardness of distributed approximation for many problems, as in the theorem below. For brevity, we only prove the theorem for the minimum spanning tree problem here. Definitions and proofs of other problems can be found in Appendix D.

Theorem 5.1. *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 [7, 21], shortest s-t path, s-source distance [5], s-source shortest path tree [7], minimum cut [4], minimum s-t cut, maximum cut, minimum routing cost spanning tree [25], shallow-light tree [20], and generalized Steiner forest [10].*

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

Recall the following standard notions of an *approximation algorithm.* We say that a randomized algorithm $\mathcal{A}$ is *α-approximation ϵ-error* if, for any input instance $\mathcal{I}$, algorithm $\mathcal{A}$ outputs a solution that is at most α times the optimal solution of $\mathcal{I}$ with probability at least $1-\epsilon$. Therefore, in the minimum spanning tree, an α-approximation ϵ-error algorithm should output a number that is at most α times the total weight of the minimum spanning tree, with probability at least $1-\epsilon$.

Proof of Theorem 5.1. (This proof is only for the case of minimum spanning 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.

To do this, construct a weight function on edges in G, denoted by ω, by assigning weight 1 to all edges in H and $n\alpha(n)$ to all other edges. Note that constructing ω does not need any communication since each vertex knows which edges incident to it are in H. Now we find the weight W of the minimum spanning tree using $\mathcal{A}_\epsilon$ and announce that H is a spanning connected subgraph if and only if W is less than $n\alpha(n)$.

Now we show that the weighted graph (G, ω) 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 $n-1 < n\alpha(n)$. 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 must contain an edge not in H. Therefore, any spanning tree has weight at least $n\alpha(n)$ as claimed. □

Our MST lower bound here matches the lower bound of exact MST algorithms and improves the lower bound of $\Omega(\sqrt{\frac{n}{\alpha B}})$ by Elkin [7]. Our lower bound for s-source distance complements the results in [5].

6 Deterministic Lower Bounds

We show the following lower bound of deterministic algorithms for problems listed in the theorem below. We note that our lower bound of spanning tree verification simplifies and generalizes the lower bound of *minimum* spanning tree verification shown in [11]. For brevity, problem definitions and proofs are placed in Appendix B.

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 problems requires $\Omega((\frac{n}{pB})^{\frac{1}{2}-\frac{1}{2(2p+1)}})$ time on some $\Theta(n)$-vertex graph of diameter $O(2p+2)$ in the B model: Hamiltonian cycle, spanning tree, and simple path verification.*

7 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.

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 [4].) Also, showing randomized bounds for Hamiltonian path, spanning tree, and simple path verification remains open.

References

[1] Arkadev Chattopadhyay and Toniann Pitassi. The Story of Set Disjointness. *SIGACT News*, 41(3):59–85, 2010.

[2] Edith Cohen. Size-Estimation Framework with Applications to Transitive Closure and Reachability. *J. Comput. Syst. Sci.*, 55(3):441–453, 1997. Also in FOCS'94.

[3] Devdatt P. Dubhashi, Fabrizio Grandioni, and Alessandro Panconesi. Distributed Algorithms via LP Duality and Randomization. In *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall/CRC, 2007.

[4] Michael Elkin. Distributed approximation: a survey. *SIGACT News*, 35(4):40–57, 2004.

[5] Michael Elkin. Computing almost shortest paths. *ACM Transactions on Algorithms*, 1(2):283–323, 2005.

[6] Michael Elkin. A faster distributed protocol for constructing a minimum spanning tree. *J. Comput. Syst. Sci.*, 72(8):1282–1308, 2006. Also in SODA'04.

[7] Michael Elkin. An Unconditional Lower Bound on the Time-Approximation Trade-off for the Distributed Minimum Spanning Tree Problem. *SIAM J. Comput.*, 36(2):433–456, 2006. Also in STOC'04.

[8] Juan A. Garay, Shay Kutten, and David Peleg. A Sublinear Time Distributed Algorithm for Minimum-Weight Spanning Trees. *SIAM J. Comput.*, 27(1):302–316, 1998. Also in FOCS '93.

[9] Monika R. Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. In James M. Abello and Jeffrey Scott Vitter, editors, *External memory algorithms*, pages 107–118. American Mathematical Society, Boston, MA, USA, 1999.

[10] Maleq Khan, Fabian Kuhn, Dahlia Malkhi, Gopal Pandurangan, and Kunal Talwar. Efficient distributed approximation algorithms via probabilistic tree embeddings. In *PODC*, pages 263–272, 2008.

[11] Liah Kor, Amos Korman, and David Peleg. Tight bounds for distributed MST verification. *In preparation*, 2010.

[12] Amos Korman and Shay Kutten. Distributed verification of minimum spanning trees. *Distributed Computing*, 20(4):253–266, 2007.

[13] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. What cannot be computed locally! In *PODC*, pages 300–309, 2004.

[14] Eyal Kushilevitz and Noam Nisan. *Communication complexity.* Cambridge University Press, New York, NY, USA, 1997.

[15] Shay Kutten and David Peleg. Fast Distributed Construction of Small k-Dominating Sets and Applications. *J. Algorithms*, 28(1):40–66, 1998.

[16] Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.

[17] Zvi Lotker, Boaz Patt-Shamir, and David Peleg. Distributed MST for constant diameter graphs. *Distributed Computing*, 18(6):453–460, 2006.

[18] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, 1986. Also in STOC'85.

[19] Nancy Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.

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

[21] David Peleg and Vitaly Rubinovich. A Near-Tight Lower Bound on the Time Complexity of Distributed Minimum-Weight Spanning Tree Construction. *SIAM J. Comput.*, 30(5):1427–1442, 2000. Also in FOCS'99.

[22] Robert Endre Tarjan. Applications of path compression on balanced trees. *J. ACM*, 26(4):690–715, 1979.

[23] Ramakrishna Thurimella. Sub-Linear Distributed Algorithms for Sparse Certificates and Biconnected Components. *J. Algorithms*, 23(1):160–179, 1997.

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

[25] Bang Ye Wu, Giuseppe Lancia, Vineet Bafna, Kun-Mao Chao, R. Ravi, and Chuan Yi Tang. A polynomial-time approximation scheme for minimum routing cost spanning trees. *SIAM J. Comput.*, 29(3):761–778, 1999. Also in SODA'98.

[26] Andrew Chi-Chih Yao. Probabilistic Computations: Toward a Unified Measure of Complexity. In *FOCS*, pages 222–227, 1977.

Appendix

A Randomized lower bounds

In this section, we show the randomized lower bounds as claimed in Theorem 4.1 for the following problems (listed in Fig. 2).

Definition A.1 (Problems with randomized lower bounds)**.** We define:

- ***s*-*t* connectivity verification problem**: 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. (Section A.1.)

- **cycle containment verification problem:** We want to verify if H contains a cycle. (Section A.2.)

- ***e*-cycle containment verification problem:** Given an edge e in H (known to vertices adjacent to it), we want to verify if H contains a cycle containing e. (Section A.2.)

- **bipartiteness verification problem**: We want to verify whether H is bipartite. (Section A.2.)

- **connectivity verification problem**: We want to verify whether H is connected. We also consider the **k-component verification problem** where we want to verify whether H has at most k connected components. (Note that k is not part of the input so 2-component and 3-component problems are different problems.) The connectivity verification problem is the special case where $k = 1$. (Section A.3)

- **cut verification problem**: We want to verify whether H is a cut of G, i.e., G is not connected when we remove edges in H. (Section A.3)

- ***s*-*t* cut verification problem**: 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. (Section A.3)

- **least-element list verification problem [2, 10]:** 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 $\{< v, d(u, v) > \mid v \text{ is the least element of } u\}$. (Section A.3)

  In the least-element list verification problem, each vertex knows its rank as an input, and some vertex u is given a set $S = \{< v_1, d(u, v_1) >, < v_2, d(u, v_2) >, \ldots\}$ as an input. We want to verify whether S is the least-element list of u. (Section A.3)

- **edge on all paths verification problem:** Given nodes u, v and edge e. We want to verify whether e lies on all paths between u, v in H. (Section A.3)

Lower bounds of the above problems are stated in Theorem 4.1 and the reductions are summarized in Fig. 2.

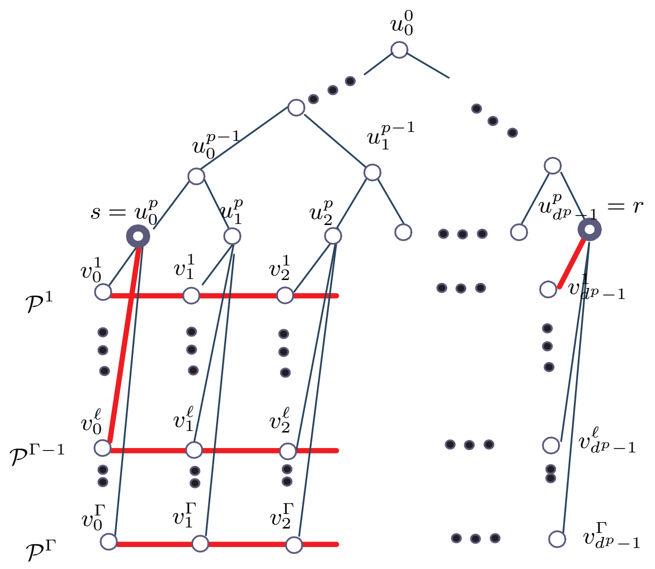


Figure 4: Example of H for s-t connectivity problem (marked with thick red edges) when $\bar{x} = 0...10$ and $\bar{y} = 1...00$.

A.1 Randomized lower bound of s-t connectivity verification

Similar to the lower bound of the spanning connected subgraph verification problem, the lower bounds of s-t connectivity follow from the following lemma.

Lemma A.2. *For any Γ, $d \geq 2$ and p, there exists a constant $\epsilon > 0$ such that any ϵ-error distributed algorithm for s-t connectivity verification problem on graph $G(\Gamma, d, p)$ can be used to solve the* DISJ$(G(\Gamma, d, p), s, r, \Gamma)$ *problem on $G(\Gamma, d, p)$ with the same time complexity.*

Proof. We use the same argument as in the proof of Lemma 4.4 except that we construct the subgraph H as follows.

s-t verification: 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 Fig. 4.)

Once algorithm $\mathcal{A}_{st}$ terminates, vertex r determines its output for the set disjointness problem by stating that both input strings are disjoint if and only if s-r connectivity verification algorithm verified that s and r are *not* connected in the given subgraph.

For the correctness of this algorithm, 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 the s-r connected subgraph candidate H, H is s-r connected if and only if the input strings $\bar{x}$ and $\bar{y}$ are *not* disjoint, i.e., there exists i such that $x_i = 1$ and $y_i = 1$. Hence the resulting algorithm has correctly solved the given instance of the set disjointness problem. $\square$

A.2 A randomized lower bound for cycle containment, e-cycle containment, and bipartiteness verification problem

Lemma A.3. *There exists a constant $\epsilon > 0$ such that any ϵ-error distributed algorithm for cycle containment, e-cycle containment, or bipartiteness verification problem on graph $G(\Gamma, d, p)$ can be used to solve the* DISJ$(G(\Gamma, d, p), s, r, \Gamma)$ *problem on $G(\Gamma, d, p)$ with the same time complexity.*

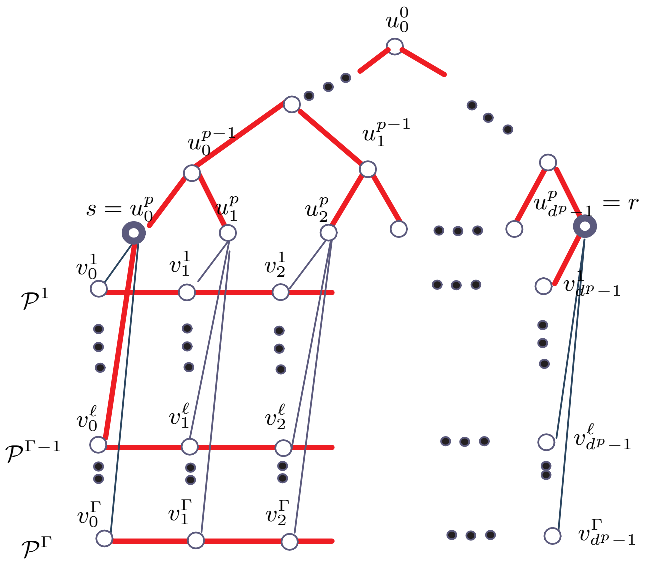


Figure 5: Example of H for the cycle and e-cycle containment and bipartiteness verification problem when $\bar{x} = 0...10$ and $\bar{y} = 1...00$.

Proof. cycle verification problem: We construct H in the same way as in the proof of Lemma A.2 *except* that the tree edges are participating in H (see Fig. 5).

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 and 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 $\bar{x}_i = \bar{y}_i = 1$. This causes a cycle in H consisting of some tree edges and path 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 A.4. *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 A.5. *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. The lower bound of bipartite follows from this claim.

Like in the previous proofs, we observe that if the input strings are not disjoint, then either H or H' are not bipartite. We 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). In follows that H' does not contain a cycle as well. Therefore, we have the following claim.

Claim A.6. *H and H' are both bipartite if and only if the input strings are disjoint.*

□

A.3 Randomized lower bounds of connectivity, k-component, cut, s-t cut, least-element list, and edge on all paths verification

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, we show that there is an algorithm $\mathcal{A}'(G', H')$ that verifies whether H' is a spanning connected subgraph in $O(\tau(n') + D')$ time, where n' and D' is the number of vertices in G' and its diameter, respectively. Thus, the lower bounds (which are larger than D) of the spanning connected subgraph problem apply to the connectivity verification problem as well.

To do this, 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 in $O(D)$ rounds and checking if H' is connected can be done in $O(\tau(n'))$ rounds by calling $\mathcal{A}(G, H)$ with $H = H'$ and $G = G'$. The total running time of $\mathcal{A}'$ is thus $O(\tau(n') + D)$.

k-component verification problem: The above argument can be extended to show the lower bound of k-component problem, as follows. Suppose again that we want to check if H is a spanning connected subgraph. Now we add $k-1$ virtual nodes adjacent to some node s in G. These nodes are added to H (denote the resulting subgraph by H') but will not be incident to any edges in H' and are simulated by s. Observe that the new graph, say G', has diameter $D' = D + 1$ and the number of nodes is $n' = n + k \leq 2n$. Moreover, H is a spanning tree of G if and only if H' has k connected component in G' (the spanning subgraph H of G plus $k-1$ single nodes). Therefore, if we can check if H has at most k (k constant) connected component in G' in $O(\tau(n'))$ time then we can also check if H is a spanning connected subgraph in G.

cut verification problem: We again reduce from the spanning connected subgraph problem. Recall the definition of the cut that H is a cut if and only if G is *not* connected when we remove edges in H. In other words, H is a cut if and only if $\bar{H}$ is not a spanning connected subgraph of G where $\bar{H}$ is the graph resulting from removing edges in H.

Thus, given a subgraph H', we verify if H' is a spanning connected subgraph as follows. Let H'' be the graph obtained by removing edges $E(H')$ of H' from G. Recall again that H' is a spanning connected subgraph if and only if H'' is not a cut. Thus, we verify if H'' is a cut. We announce that H' is a spanning connected subgraph if and only if H'' is verified not to be a cut.

s-t cut verification problem: The lower bound of s-t cut is proved similarly: H' is s-t connected if and only if H'' obtained by removing edges in H' from G 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. 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. On the other hand, if s and t are not connected then the distance between them will be at least 1 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 $e = uv$.

B Deterministic Lower Bounds

In this section, we show the randomized lower bounds as claimed in Theorem 6.1 for the following problems (also listed in Fig. 2).

Definition B.1 (Problems with deterministic lower bounds)**.** We define:

- **Hamiltonian cycle:** Given a graph G and subgraph H of G, 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 (ST) verification:** We would like to verify whether H is a spanning tree of G.
- **Simple path verification** Given a graph G, subgraph H of G verify that H is a simple path.

We first prove the lower bound of the first problem and later extend to other problems. To do this, we need a deterministic lower bound of computing the *equality function*, as follows.

B.1 A deterministic lower bound of computing equality function

Definition B.2 (Equality function)**.** Given two b-bit strings $\bar{x}$ and $\bar{y}$, the *equality function*, denoted by $\texttt{eq}(\bar{x}, \bar{y})$, is defined to be one if $\bar{x} = \bar{y}$ and zero otherwise. We refer to the problem of computing $\texttt{eq}$ function on $G(\Gamma, d, p)$ on Γ-bit input strings given to s and r as $\texttt{EQ}(G(\Gamma, d, p), s, r, \Gamma)$.

The following lemma follows from Theorem 3.1 and the communication complexity lower bound of computing $\texttt{eq}$.

Lemma B.3. *For any Γ, d, p, any deterministic algorithm solving $\texttt{EQ}(G(\Gamma, d, p), s, r, \Gamma)$ requires $\Omega(\min(d^p, \Gamma/dpB))$ time.*

Proof. We use the fact that $R_0^{cc-pub}(\texttt{eq}) = \Omega(\Gamma)$ for the function $\texttt{eq}$ on Γ-bit inputs (see, e.g., [14, Example 1.21] and references therein). Thus by Theorem 3.1, $R_0^{G(\Gamma,d,p),s,r}(f) = \Omega(\min d^p, \Gamma/dpB)$ implying the lemma. ☐

Corollary B.4. *For any Γ, d, p and $b = \Theta(\Gamma)$, any deterministic algorithm solving $\texttt{EQ}(G(\Gamma, d, p), s, r, b)$ requires $\Omega(\min(d^p, \Gamma/dpB))$ time.*

B.2 A deterministic lower bound for Hamiltonian cycle verification

Lemma B.5. *Any distributed algorithm for Hamiltonian cycle verification on a graph $G(\Gamma, 2, p)'$ (as defined in the proof) can be used to solve the $\texttt{EQ}(G(\Gamma, 2, p), r, s, b)$, problem on $G(\Gamma, 2, p)$, where $\Gamma = 2 + 12b$, with the same time complexity.*

Proof. We construct $G(\Gamma, 2, p)'$ from $G(\Gamma, 2, p)$ by adding edges and vertices to $G(\Gamma, 2, p)$. Since $d = 2$ thus $\mathcal{T}$ will be a binary tree. Let $m = d^p - 1$. First we 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. So far we only modified left/right parts of the graphs that are far away from the middle – this modification does not change the argument in the proof causing the lower bound for the equality problem. Now we add edges (u_i^p, u_{i+1}^p) for all $0 \le i \le m-1$. Thus we shorten the distance between each pair of nodes u_i^p and u_{i+1}^p from ≤ 3 to 1 (red edges in Figure 6). 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 paths/nodes in Figure 6). This will affect the lower bound by at most a constant factor as well since it will increase the amount of messages that can be sent through the graph within a certain time by at most 3 times. Furthermore 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 affect the lower bound only by a constant as well. Thus the lower bound for the equality problem remains valid on $G(\Gamma, 2, p)'$.

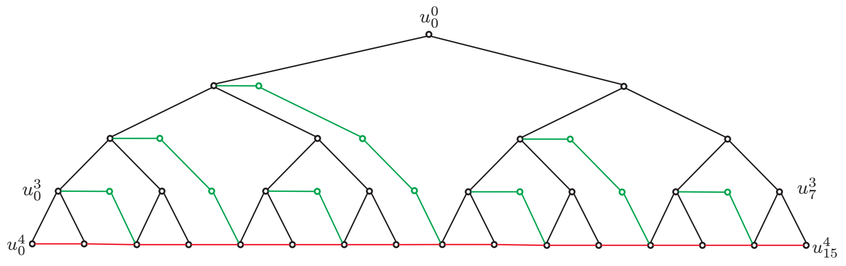


Figure 6: Example of the modification of the tree-part of G in the case $p = 4$. In red: the new edges (u_i^p, u_{i+1}^p), in green, the new paths/nodes connecting u_i^l to $u_{i \cdot d^{p-l}+1}^p$.

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

$$\bar{x}' := 1x_1 01 x_1 01 x_2 01 x_2 01 \ldots 01 x_b 01 x_b 01 \overline{x_1} 01 \overline{x_1} 01 \ldots 01 \overline{x_b} 01 \overline{x_b} 010,$$

$$\bar{y}' := 1y_1 01 y_1 01 y_2 01 y_2 01 \ldots 01 y_b 01 y_b 01 \overline{y_1} 01 \overline{y_1} 01 \ldots 01 \overline{y_b} 01 \overline{y_b} 010$$

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

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 left over lines with each other. These three paths will cover all nodes. The final stage is to connect all three paths with each other. If the input strings are equal the resulting graph H is an Hamiltonian cycle. If the input strings are not equal, things are getting messed up and we can show that the result is not a Hamiltonian cycle. Observe that in the case the strings are equal all three paths will look like disjoint snakes when using the graph layout of Figure 3(a). The formal description of the five stages will be accompanied by a small example in Figures 7 and 9. $\bar{x} = 01 = \bar{y}$.

First stage: 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 $j \in \{2, \ldots, m-2\}$ for $j \in \{1, \ldots, m-1\}$ 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 7).

Second stage: define path S_1 – 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, $1 \leq i \leq \Gamma$, 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, $1 \leq i \leq m^K$, 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 $2 \leq i \leq m$ each edge (v_0^i, v_1^i) participates in H if and only if $x'_{i-1} \neq x'_i$. Similarly for $2 \leq i \leq m^K$ 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 7.

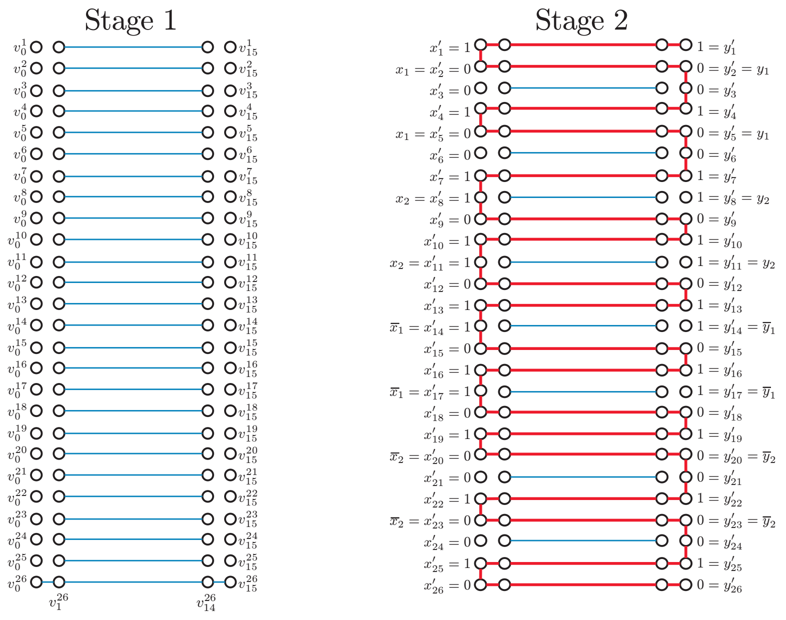


Figure 7: Example of the reduction using input strings $\bar{x} = 01 = \bar{y}$, thus Γ is $12 \cdot 2 + 2 = 26$ and we use $d = 2$ and $p = 4$. In stage one, we add lines to H, that are displayed in blue. In stage two we create S_1, the red-colored path that looks like a snake.

Third stage: define S_2 – we connect the other lines (but not the highways) and those nodes that are not covered by any path/line yet. On the s-side of the graph, for $0 \leq i \leq 2b$, whenever

- $x'_{2+6i} = 0$ (and thus $x'_{5+6i} = 0$ due to the definition of $\bar{x}'$), 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.
- $x'_{2+6i} = 1$ (and thus $x'_{5+6i} = 1$ due to the definition of $\bar{x}'$), 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 r-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 9.

Fourth stage: We include edges of the modified tree in a canonical way to H such that the path S_3 looks like a snake – for all odd i in $0 \le i \le m-1$ we include the edge (u_i^p, u_{i+1}^p) in H. For all $0 \le l \le p-1$ and all $0 \le i \le d^l$ we include the edges $(u_i^l, u_{i \cdot d+1}^{l+1})$ in H – 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 8.

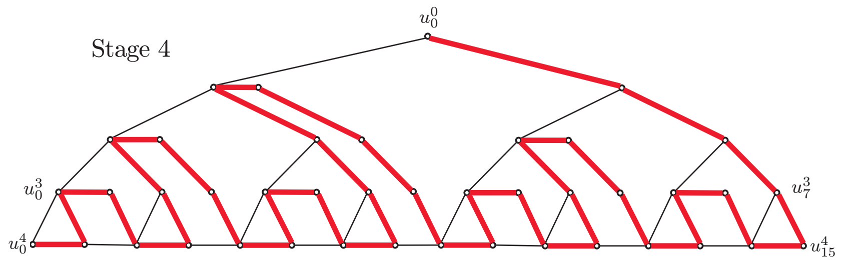


Figure 8: The modified tree for $p = 4$ and $d = 2$. In red/bold: path S_3.

Fith stage: connect the endpoints!

Lets investigate the six endpoints of the three paths:

- one endpoint of the snake S_3 is u_0^0, another endpoint is u_0^p.
- snake S_2 has both endpoints on the r-side. Lets 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, 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: 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 S_3 by its endpoint u_0^0 to the endpoint v_m^Γ of S_1.
- 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 give in Figure 9.

If the strings are equal the result is a Hamiltonian cycle since in this case the snakes where chosen to be three disjoint paths that cover all nodes.

Now we need to 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. Lets 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 at the part of the graph H corresponding to this sequence (see Figure 10), 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

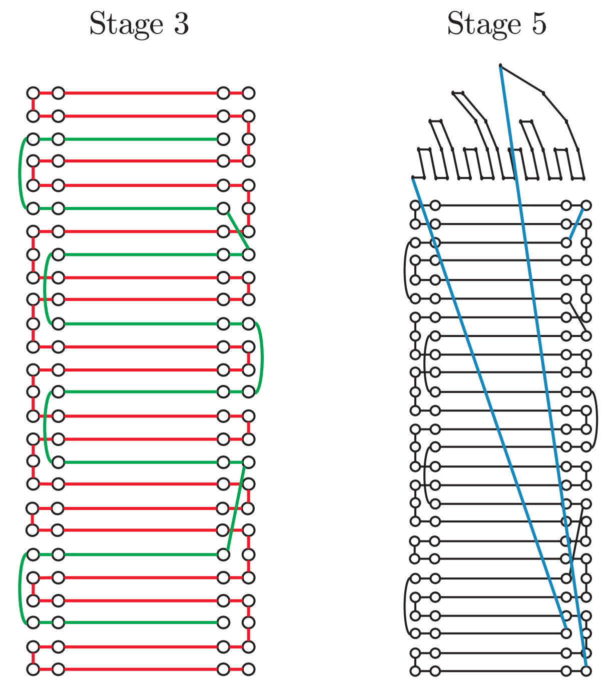


Figure 9: Continuation of the example started in 7. In stage three, we add S_2 in green. S_3 is displayed in brown and added in stage four. Finally we connect S_1, S_2 and S_3 to a Hamiltonian cycle in stage five.

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.

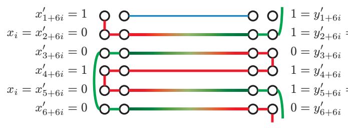


Figure 10: Example of the case that $x_i = 0$ and $y_i = 1$.

The case that $x_i = 1$ and $y_i = 0$ is treated the same way, due to the construction of $\bar{x}'$ and $\bar{y}'$ 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 an 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 $G(\Gamma, 2, p)'$ and thus on $G(\Gamma, 2, p)$ faster. A contradiction to the lower bound for the equality problem, which holds for all d (we used $d = 2$). $\square$

Combined with Corollary B.4, we now have

Theorem B.6. *For every Γ and corresponding p, any distributed algorithm for solving Hamiltonian cycle problem on the graph $(\Gamma, 2, p)'$ in the B model for $B \geq 3$ requires $\Omega(\min(d^p, \Gamma/dpB))$ time.*

Using the same analysis as in Section 4, we obtain:

Corollary B.7. *For any $p \geq 1$, $B \geq 3$, and $n \in \{2^{2p+1}pB, 3^{2p+1}pB, \ldots\}$, any distributed algorithm for the Hamiltonian cycle problem in the B model for $B \geq 3$ requires $\Omega(\frac{n}{pB}^{\frac{1}{2}-\frac{1}{2(2p+1)}})$ time on some n-vertex graph of diameter $2p+2$.*

B.3 A deterministic lower bound for spanning tree and path verification problems

Lemma B.8. *For any $p \geq 1$, $B \geq 3$, and $n \in \{2^{2p+1}pB, 3^{2p+1}pB, \ldots\}$, any $B \geq 3$, any deterministic algorithm for spanning tree verification requires $\Omega(\frac{n}{pB}^{\frac{1}{2}-\frac{1}{2(2p+1)}})$ time in some family of n-vertex graph of diameter $2p+2$.*

Proof. 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. □

Lemma B.9. *For any $p \geq 1$, $B \geq 3$, and $n \in \{2^{2p+1}pB, 3^{2p+1}pB, \ldots\}$, any $B \geq 3$, any deterministic algorithm for path verification requires $\Omega((\frac{n}{pB}^{\frac{1}{2}-\frac{1}{2(2p+1)}}))$ time in some family of n-vertex graph of diameter $2p+2$.*

Proof. 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). □

C Tightness of lower bounds

We note that 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 the s-t connectivity, k-component, connectivity, cut, s-t-cut, bipartiteness, edge on all path, and simple path verification problems. Algorithms for all other problems stated in this paper can be found using the reductions given in Figure 2.

In particular, one can use the MST algorithm by Kutten and Peleg [15] and the connected component algorithm by Thurimella [23][Algorithm 5] to verify these properties.

Deterministic algorithms almost matching the deterministic lower bounds: We need to give upper bounds for the k-spanning tree and path verification problems.

Path verification problem: compute a breath first search-tree T on $G \setminus H$ in time $O(D)$ connect the tree T to H by a single edge of G. The resulting subgraph of G is a spanning tree of G if and only if H is a path.

k-spanning tree verification problem: We construct a weighted graph G' by assigning weight zero to all edges in $H^k := H$ and one to other edges. We then find a minimum spanning T^k tree of H using the $O(\sqrt{n}\log^* n + D)$-time algorithm in [15]. Now we create $H^{k-1} := H^k \setminus T^k$. H^{k-1} is a $k-1$-spanning tree if and only if H^k is a k-spanning tree. If all T^j were spanning trees, after k iterations we are left with H^0 which contains no nodes and no edges if and only if H^k was a k-spanning tree.

Deterministic algorithms almost matching the randomized lower bounds: We need to give upper bounds for the s-t connectivity, cycle, connectivity, k-components, cut, s-t cut, bipartiteness and edge on path-verification problems.

s-t connectivity verification problem: To do this, we run the connected component algorithm by Thurimella [23][Algorithm 5] where, 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. [23][Theorem 6] states that the distributed time complexity of [23][Algorithm 5] is $O(D + f(n) + g(n) + \sqrt{n})$ where $f(n)$ and $g(n)$ are the distributed time complexities of finding an MST and a $\sqrt{n}$ -dominating set, respectively. Due to [15] we have that $f(n) = g(n) = O(F + \sqrt{n}\log^* n)$. We can now verify whether s and t are in the same connected component by verifying whether $\ell(s) = \ell(t)$.

cycle verification problem: Assign weight 0 to all edges of H and weight 1 to all edges of $G \setminus H$. Compute a minimum spanning tree of G using [15]. H contains no cycle if and only if all edges E_H of H are in the minimum spanning tree, i.e., $W = n - 1 - |E(H)|$ where $|E(H)|$ is the number of edges in H.

edge on all path verification problem: If and only if u and v are disconnected in $H \setminus \{e\}$, then e is on all paths between u and v. We can use the s-t connectivity verification algorithm from above to check that.

cut verification problem: To verify if H is a cut, we simply verify if G after removing the edges E_H of H is connected.

s-t cut verification problem: To verify if H is an s-t cut, we simply verify s-t connectivity of G after removing the edges E_H of 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. It is thus left to verify s-t connectivity.

k-components verification problem: We simply put weight 1 on edges in H and 2 on other edges and find the MST using an algorithm in [15]. Observe that H has at most k connected component if and only if there are at most $k-1$ edges of weight 2 in the MST, i.e., the MST has weight at most $n - 1 + (k - 1)$.

connectivity verification problem: Same as above for $k = 1$.

spanning connected subgraph verification problem: One can use the above described algorithm to verify that H is connected. Verifying that the vertices V_G of G are the same as the vertices V_H of H completes the algorithm.

bipartiteness verification problem: Compute a minimum spanning tree T_H of H in time $O(\sqrt{n}\log^* n + D)$ using [15]. Now 2-color this tree. Now all nodes check if their neighbors have a different color then themselves. They will have a different color if and only if H is bipartite.

D Details of hardness of approximation

In this section, we show the randomized lower bounds as claimed in Theorem 5.1 for the following problems (listed in Fig. 2).

Problems: Given a connected graph G with a weight function ω on edges (where, for each edge e, $\omega(e)$ is known to nodes incident to e), we consider the following problems.

- In the **minimum spanning tree** problem [7, 21], 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 a node outputs this weight.

- Given two nodes s and t, the **shortest s-t path** problem is to find the shortest path between s and t. In the end of the process a node outputs the length of the shortest path.

- Given a node s, the **s-source distance** problem [5] is to find the distance from s to every node. In the end of the process, every node knows its distance from s. The **s-source shortest path tree** problem [7] is to find the shortest path spanning tree rooted at s, i.e., the shortest path from s to any node t has the same weight as the unique path from s to t in such a tree.

- A set of edges E' is a **cut** if G is not connected when we delete E'. The **minimum cut** problem [4] is to find a cut of minimum weight. A set of edges E' is a s-t cut if s and t are not connected when we delete E'. The **minimum s-t cut** problem is to find an s-t cut of minimum weight. In the end of the process, a node outputs the weight of the minimum cut and minimum s-t cut. The **maximum cut** problem is to find a cut of maximum weight.

- The **minimum routing cost spanning tree** problem [25] is defined as follows. We think of the weight on an edge as the cost of routing messages through this edge. The routing cost for a pair of vertices in a given spanning tree is the sum of the weights of the edges in the unique tree path between them. The routing cost of the tree itself is the sum over all pairs of vertices of the routing cost for the pair in the tree. Our goal is to find a spanning tree with minimum routing cost.

- The **generalized Steiner forest** problem [10] is defined as follows. We are given k disjoint subsets of vertices $V_1, ..., V_k$. The goal is to find a minimum weight subgraph in which each pair of vertices belonging to the same subsets are connected.

- Given a network with two cost functions associated to edges: weight and length. Given a root node r and the desired radius ℓ, a **shallow-light tree** [20] is the spanning tree whose radius (defined by length) is at most ℓ and the total weight is minimized (among trees of the desired radius).

We recall the following standard notion of an approximation algorithm which we defined earlier in Section 5. For any minimization problem $\mathcal{X}$, we say that an algorithm $\mathcal{A}$ is an α-approximation if, for any input instance $\mathcal{I}$, algorithm $\mathcal{A}$ outputs a solution that is at most α times the optimal solution of $\mathcal{I}$.

Therefore, in the minimum spanning tree, minimum cut, minimum s-t cut, and shortest s-t path problems stated above, 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 s-source shortest path tree 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 $\ell \leq \alpha \cdot distance(s, v)$.

Additionally, we say that a randomized algorithm $\mathcal{A}$ is α-approximation ϵ-error if, for any input instance $\mathcal{I}$, algorithm $\mathcal{A}$ outputs a solution that is at most α times the optimal solution of $\mathcal{I}$ with probability at least $1 - \epsilon$.

Proof of Theorem 5.1. The proof idea for these problems is similar to the proof that the general case Traveling Salesman Problem cannot be approximated within $\alpha(n)$ for any polynomial computable function $\alpha(n)$ (see, e.g., [24]): 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 outputs some value that is strictly more than $f(n)$. Thus, we can distinguish between the two cases. We have already used this technique to proof the lower bound of the MST problem in Section 5. We now show lower bounds of other problems using the same technique.

The lower bound for shallow-light tree follows immediately 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.

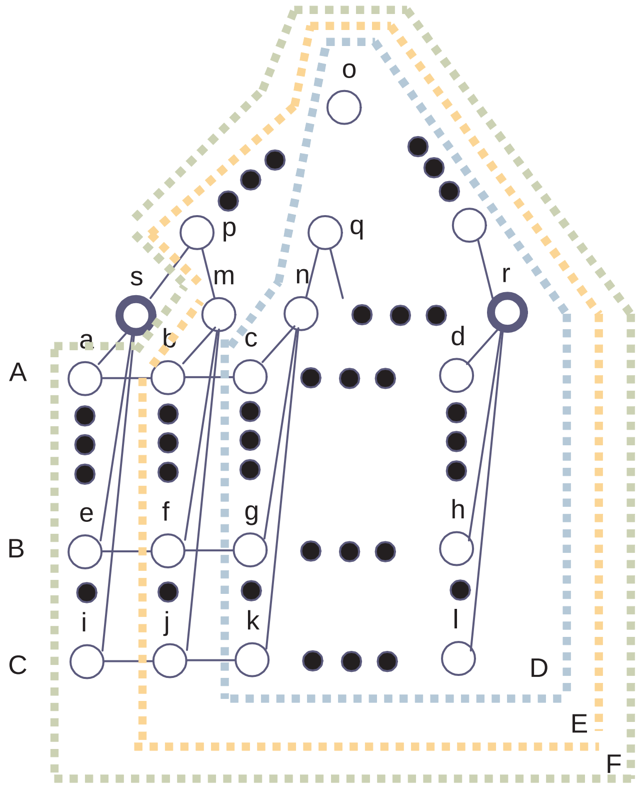
The lower bound of s-source distance and shortest path spanning tree follow in a similar way: H is a spanning connected subgraph if and only if the distance from s to every node is at most $n-1$ (i.e., $\mathcal{A}$ have approximate distance at most $(n-1)\alpha(n)$) if and only if the shortest path spanning tree contain only edges of weight one (i.e., the total weight of the shortest path spanning tree is at most $(n-1)\alpha(n)$).

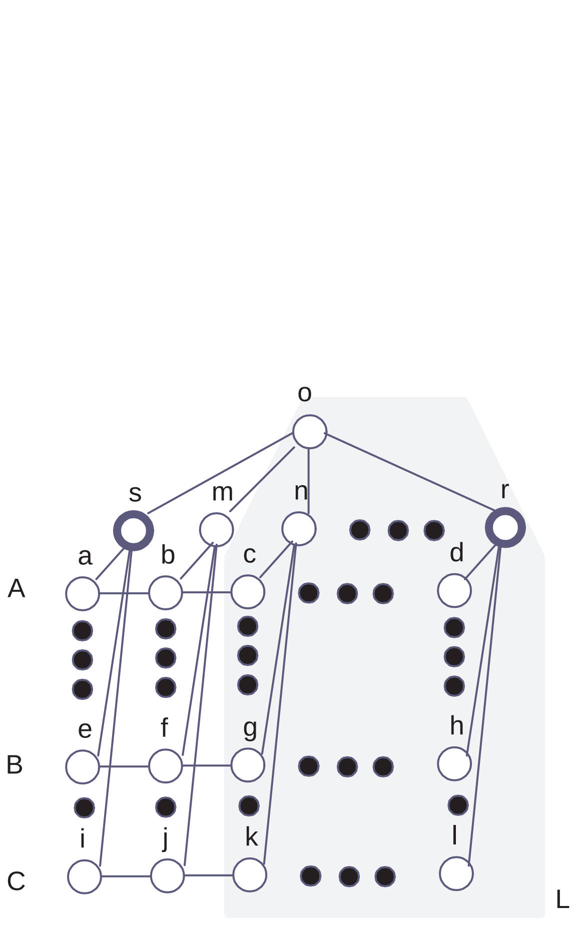
For the lower bound of the shortest s-t path, observe that s and t are connected in H if and only if the distance from s to t in G' is at most $n-1$, i.e., $\mathcal{A}$ outputs a value of at most $(n-1)\alpha(n)$. The lower bound follows from the lower bound of s-t connectivity verification problem.

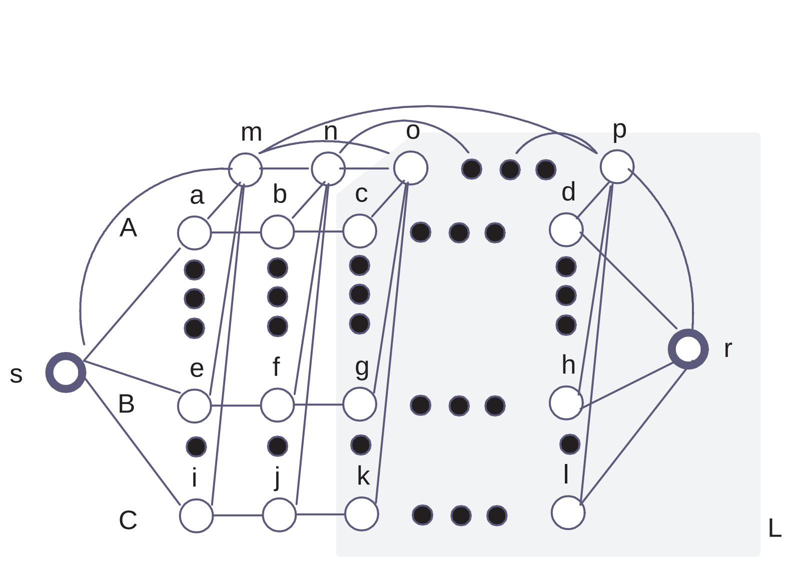
For the lower bound of the minimum cut, first observe that H is a spanning connected component if and only if $\bar{H}$, obtained by deleting all edges $E(H)$ of H from G, is not a cut. (Recall that G is assumed to be connected in the problem definition.) Therefore, verifying if $\bar{H}$ is a cut also has the same lower bound. Now, we define $\bar{G}'$ by assigning weight one to all edges in $\bar{H}$ 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 minimum cut of weight at most $n-1$, i.e., $\mathcal{A}$ outputs value at most $(n-1)\alpha(n)$. The same argument applies to s-t cut: s and t are *not* connected in H if and only if $\bar{H}$ is an s-t cut if and only if $\bar{G}'$ has minimum $s-t$ cut of weight $n-1$.

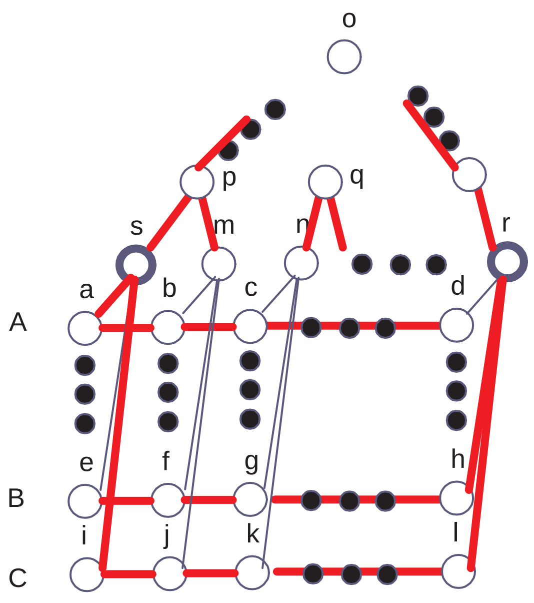
For the minimum routing cost spanning tree problem, we assign weight 1 to edges in H and $n^3\alpha(n)$ to other edges. Observe that if H is a spanning connected subgraph, the routing cost between any pair will be at most $n-1$ and thus the cost of the $\alpha(n)$-approximation minimum routing cost spanning 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 $n^3\alpha(n)$.

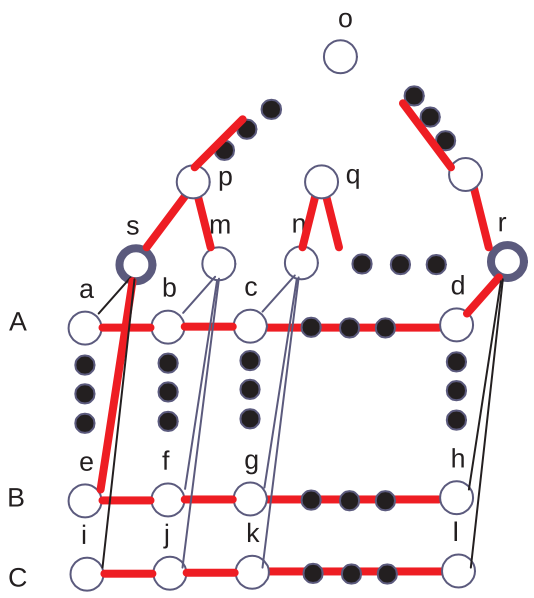
For the generalized Steiner forest problem, we will reduce from the lower bound of s-r connectivity. We will have only one set $V_1 = \{s, r\}$. We assign weight 1 to edges in H and $n\alpha(n)$ to other edges. Observe that the minimum generalized Steiner forest will have weight at most $n-1$ if H is s-t connected and at least $n\alpha(n)$ otherwise. (Recall that G is assumed to be connected in the problem definition.) $\square$

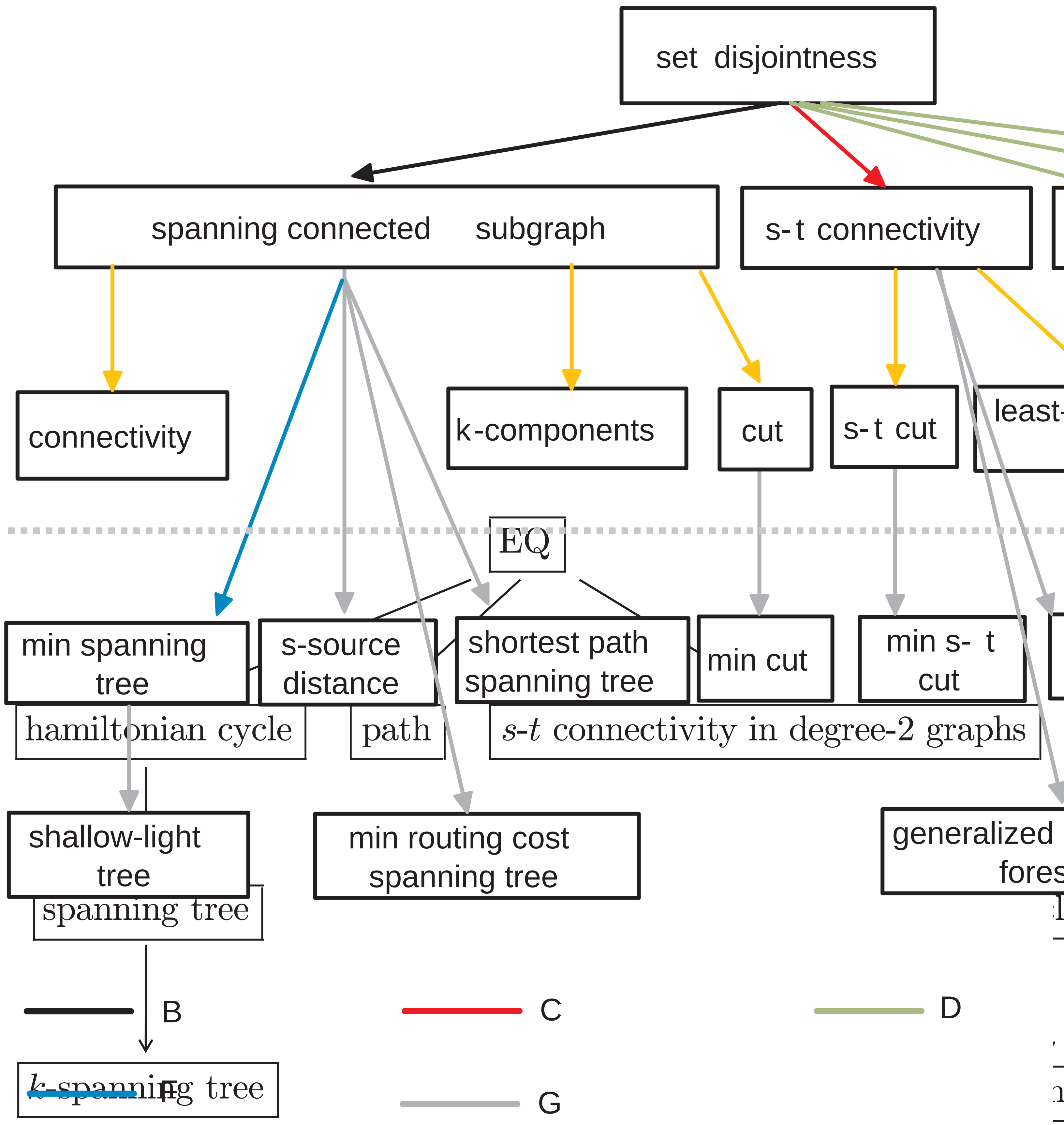

o
p
q
s
m
n
r
a
b
c
d
A
e
f
g
h
B
i
j
k
l
C
D
E
F

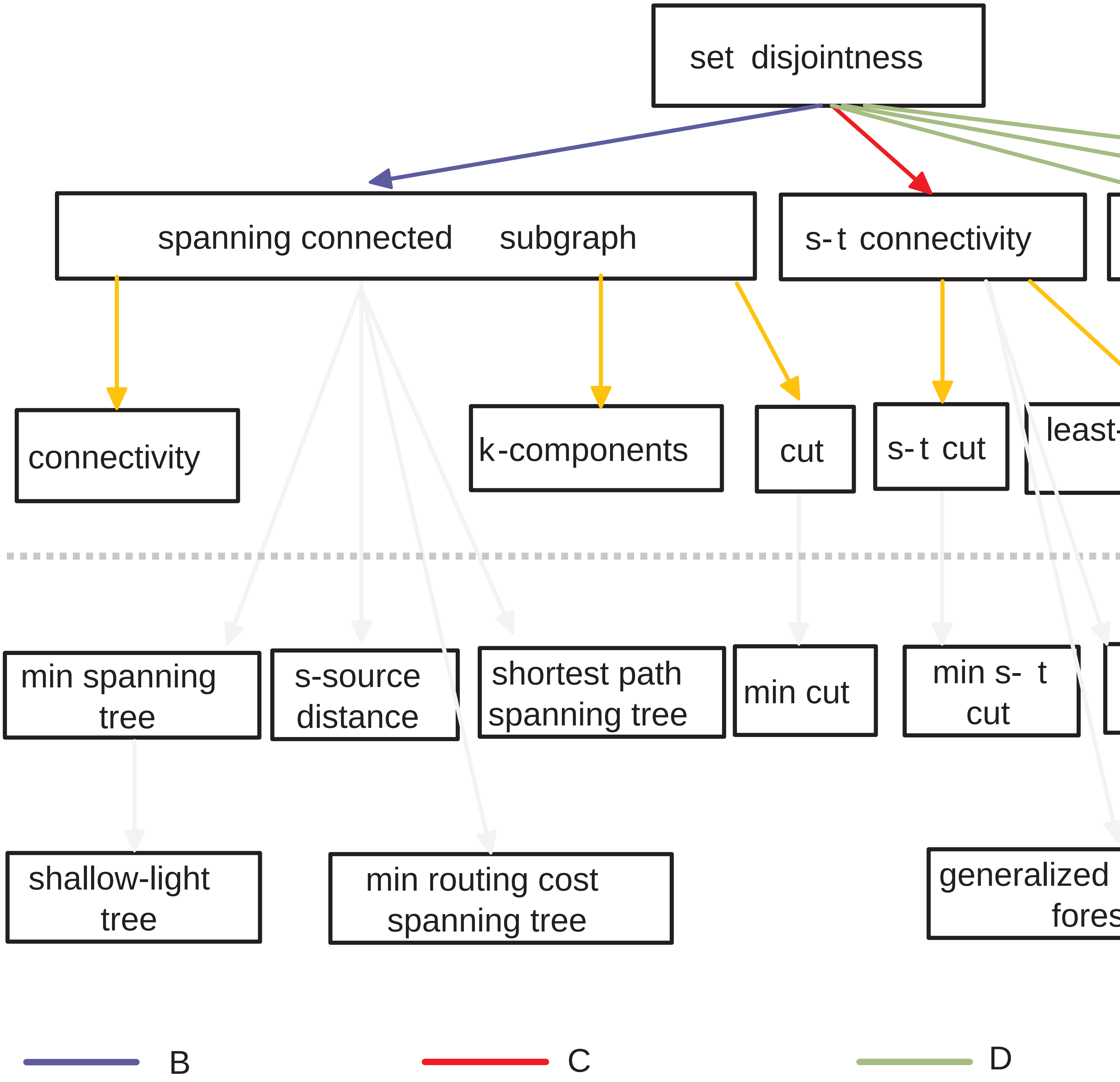

o
s
m
n
r
a
b
c
d
A
e
f
g
h
B
i
j
k
l
C
L

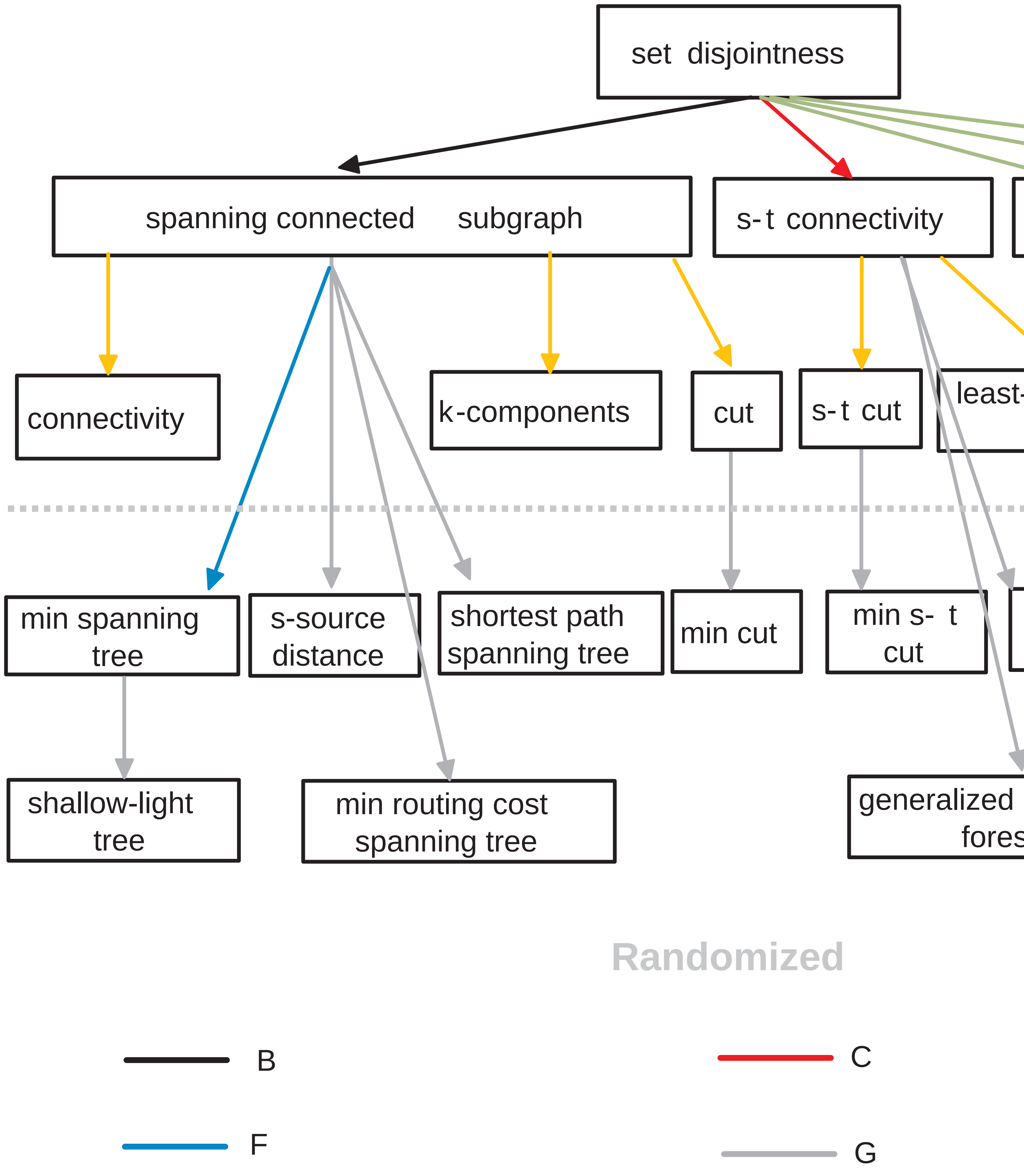

m
n
o
p
a
b
c
d
A
e
f
g
h
s
r
B
i
j
k
l
C
L

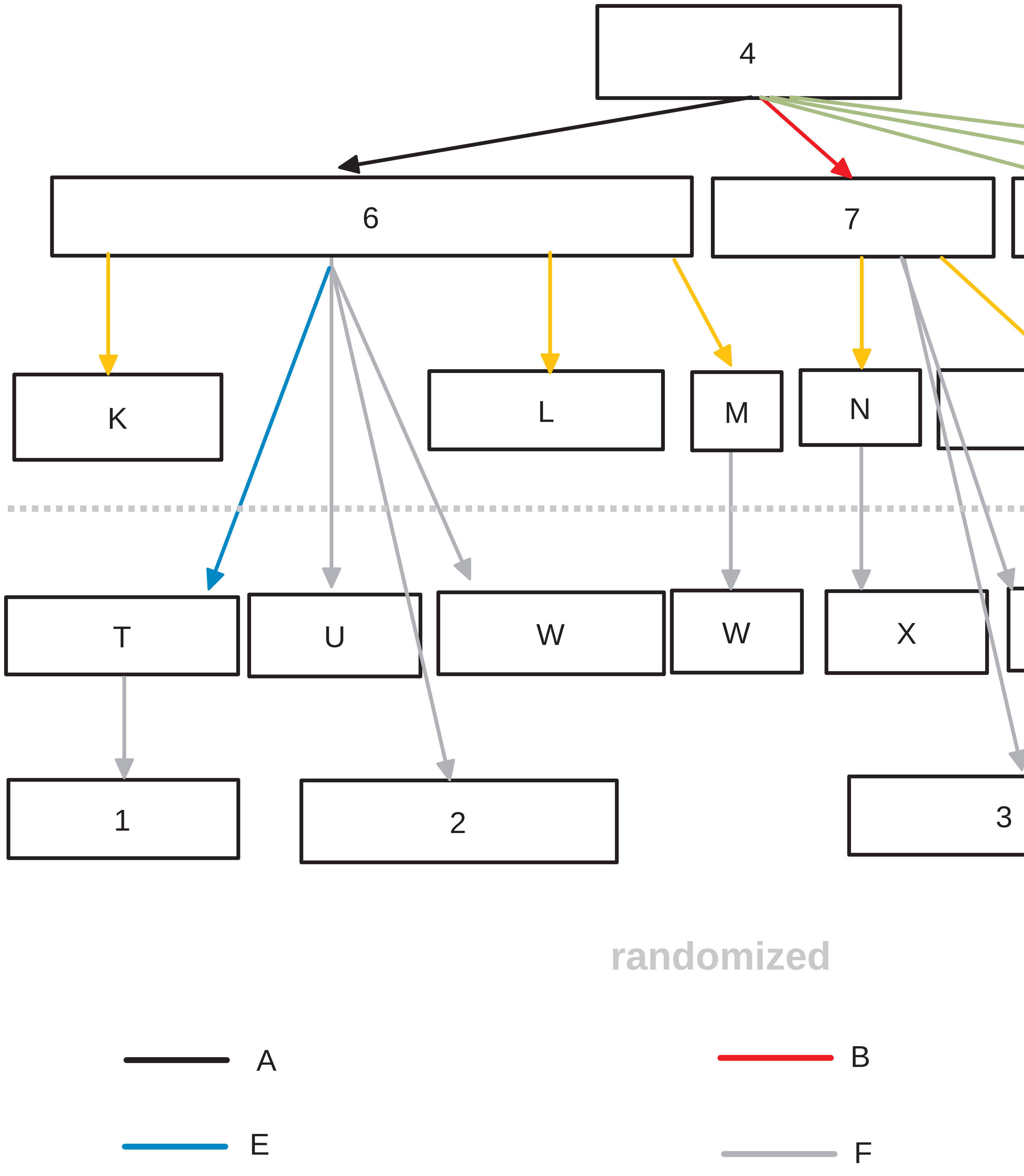

o
p
q
s
m
n
r
a
b
c
d
A
e
f
g
h
B
i
j
k
l
C

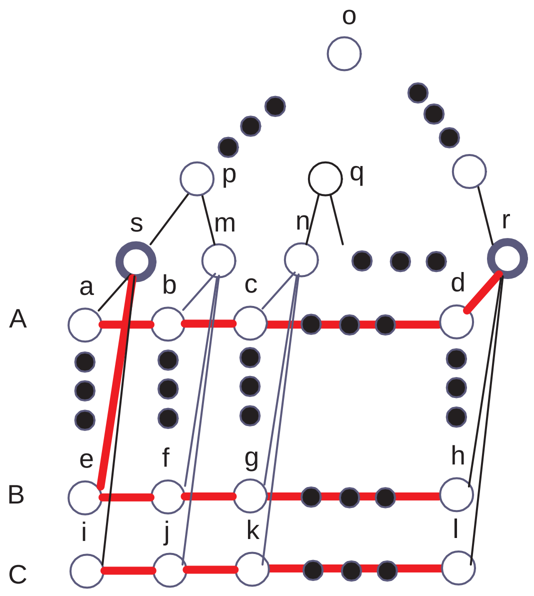

o
p
q
s
m
n
r
a
b
c
d
A
e
f
g
h
B
i
j
k
l
C


set disjointness
spanning connected subgraph
s-t connectivity
connectivity
k-components
cut
s-t cut
least-
EQ
min spanning tree
s-source distance
shortest path spanning tree
min cut
min s-t cut
hamiltonian cycle
path
s-t connectivity in degree-2 graphs
shallow-light tree
min routing cost spanning tree
generalized fores
spanning tree
k-spanning tree
B
C
D
F
G


set disjointness
spanning connected subgraph
s-t connectivity
connectivity
k-components
cut
s-t cut
least-
min spanning tree
s-source distance
shortest path spanning tree
min cut
min s- t cut
shallow-light tree
min routing cost spanning tree
generalized
fores
B
C
D


set disjointness
spanning connected subgraph
s-t connectivity
connectivity
k-components
cut
s-t cut
least-
min spanning tree
s-source distance
shortest path spanning tree
min cut
min s- t cut
shallow-light tree
min routing cost spanning tree
generalized
fores
Randomized
B
C
F
G


4
6
7
K
L
M
N
T
U
W
W
X
1
2
3
randomized
A
B
E
F



o
p
q
s
m
n
r
a
b
c
d
A
e
f
g
h
B
i
j
k
l
C