Derandomizing distributed algorithms with small messages: Spanners and dominating set

Author(s):
Ghaffari, Mohsen; Kuhn, Fabian

Publication date:
2018

Permanent link:
https://doi.org/10.3929/ethz-b-000315335

Originally published in:
Leibniz International Proceedings in Informatics (LIPIcs) 121, https://doi.org/10.4230/LIPIcs.DISC.2018.29

Derandomizing Distributed Algorithms with Small Messages: Spanners and Dominating Set

Mohsen Ghaffari
ETH Zurich, Switzerland
ghaffari@inf.ethz.ch

Fabian Kuhn
University of Freiburg, Germany
kuhn@cs.uni-freiburg.de

Abstract

This paper presents improved deterministic distributed algorithms, with $O(\log n)$-bit messages, for some basic graph problems. The common ingredient in our results is a deterministic distributed algorithm for computing a certain *hitting set*, which can replace the random part of a number of standard randomized distributed algorithms. This deterministic hitting set algorithm itself is derived using a simple method of conditional expectations. As one main end-result of this derandomized hitting set, we get a deterministic distributed algorithm with round complexity $2^{O(\sqrt{\log n \cdot \log\log n})}$ for computing a $(2k-1)$-spanner of size $\tilde{O}(n^{1+1/k})$. This improves considerably on a recent algorithm of Grossman and Parter [DISC'17] which needs $O(n^{1/2-1/k} \cdot 2^k)$ rounds. We also get a $2^{O(\sqrt{\log n \cdot \log\log n})}$-round deterministic distributed algorithm for computing an $O(\log^2 n)$-approximation of minimum dominating set; all prior algorithms for this problem were either randomized or required large messages.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Distributed Algorithms, Derandomization, Spanners, Dominating Set

Digital Object Identifier 10.4230/LIPIcs.DISC.2018.29

Related Version A full version of the paper is available at [15], `http://tr.informatik.uni-freiburg.de/reports/report285/report00285.pdf`.

1 Introduction and Related Work

We present improved deterministic distributed algorithms in the CONGEST model for graph problems including spanners and dominating set. Let us first recall the model definition.

The CONGEST model [28] of distributed computing. The network is abstracted as a simple n-node undirected graph $G = (V, E)$. There is one processor on each graph node $v \in V$, with a unique $\Theta(\log n)$-bit identifier $\mathrm{ID}(v)$, who initially knows only its neighbors in G. Communication happens in synchronous rounds. Per round, each node can send one, possibly different, $O(\log n)$-bit message to each of its neighbors. At the end, each node should know its own part of the output. For instance, when computing spanners, each node should know whether each of its edges is in the computed spanner (a computed subgraph of G, to be defined later) or not. We note that the variant of the model where we allow unbounded size messages is known as the LOCAL model [24, 28].

1.1 Our Contributions

1.1.1 Spanners

Graph spanners are a fundamental graph concept with a wide range of applications in distributed computing [4, 29]. For a graph $G = (V, E)$, a subgraph $H = (V, E')$ is an α-stretch spanner if each pairwise distance in H is at most an α factor larger than the same distance in G. Ideally, we want spanners with small stretch and small number of edges. It is known that any n-node graph admits a $(2k-1)$-stretch spanner with $O(n^{1+1/k})$ edges [4, 29], and this tradeoff is optimal conditioned on a widely-believed girth conjecture of Erdős [14].

Baswana and Sen [8] gave a randomized algorithm in the CONGEST model for computing a $(2k-1)$-stretch spanner with $O(kn^{1+1/k})$ edges in $O(k^2)$ rounds. Notice that $k \in [1, \log n]$. Hence, this is a $\text{poly}(\log n)$ round randomized algorithm with spanner size within a logarithmic factor of the optimal. There was a series of works that eventually led to a $\text{poly}(\log n)$ or even just $O(k)$ round deterministic algorithm with a similar spanner size [10–12] but all these algorithms use large messages. Currently, there are only three deterministic algorithms that work in the CONGEST model. One is the work of Barenboim, Elkin, and Gavoille [7], which runs in $\text{poly}(\log n)$ rounds, but has a considerably weaker stretch-size tradeoff: it computes a spanner with stretch $O(\log^{k-1} n)$ and size $O(n^{1+1/k})$ in $O(\log^{k-1} n)$ rounds. The other two results obtain a near-optimal stretch-size tradeoff but their round complexity is considerably higher. Derbel, Mosbah, and Zemmari [13] gave an algorithm with round complexity $O(n^{1-1/k})$ for computing a $(2k-1)$-stretch spanner with size $O(kn^{1+1/k})$. Finally, very recently, Grossman and Parter [18] gave an algorithm with round complexity $O(2^k n^{1/2-1/k})$ for computing a $(2k-1)$-stretch spanner with size $O(kn^{1+1/k})$.

Our first result considerably improves on this line of work, leading to a sub-polynomial round complexity for a nearly optimal stretch-size tradeoff:

▶ **Theorem 1.** *There is a distributed deterministic algorithm in the* CONGEST *model that computes a $(2k-1)$-stretch spanner with size $O(kn^{1+1/k} \log n)$ in $2^{O(\sqrt{\log n \log\log n})}$ rounds.*

1.1.2 Minimum Dominating Set

Minimum Dominating Set is another problem that has been central in the study of distributed algorithms for local problems, see e.g. [22]. Given a graph $G = (V, E)$, a set $S \subseteq V$ is a dominating set of G iff each node $v \in V$ is either in S or has a neighbor in S. Jia et al. [19] gave a randomized $O(\log \Delta)$-approximation in $O(\log n \log \Delta)$ rounds of CONGEST model. Kuhn and Wattenhofer [22] gave a randomized distributed algorithm that computes an $O(\sqrt{k}\Delta^{1/\sqrt{k}} \log \Delta)$-approximation in $O(k)$ rounds, e.g., an $O(\log^2 \Delta)$ approximation in $O(\log^2 \Delta)$ randomized rounds of CONGEST model. Later, Kuhn et al. [21] gave an $O(\log \Delta)$ randomized approximation in $O(\log n)$ rounds. Lenzen and Wattenhofer [23] pointed out that obtaining efficient deterministic algorithms for approximating minimum dominating set remains open. The only known result afterward is an algorithm of Barenboim et al. [7], which computes an $O(n^{1/k})$-approximation in $O(\log^{k-1} n)$ rounds; however this algorithm uses large messages. The complexity of deterministic CONGEST-model algorithms for approximating minimum dominating set remains open.

Our second result provides the first answer to this question, by providing a sub-polynomial round complexity for poly-logarithmic approximation.

▶ **Theorem 2.** *There exists a distributed deterministic algorithm in the* CONGEST *model that computes an* $O(\log^2 n)$ *approximation of minimum dominating set in* $2^{O(\sqrt{\log n \log\log n})}$ *rounds.*

We remark that while it might be possible to improve this round complexity to $2^{O(\sqrt{\log n})}$, improving it further and especially to $\text{poly}(\log n)$ would imply a major breakthrough in distributed graph algorithms: A result of Ghaffari, Harris, and Kuhn [17, Theorem 7.6] shows that obtaining a $\text{poly}(\log n)$ approximation of minimum dominating set within $\text{poly}(\log n)$ rounds is conditionally hard (even if we allow unbounded messages), because it would lead to a $\text{poly}(\log n)$-round deterministic algorithm for all locally checkable problems that admit $\text{poly}(\log n)$ round randomized algorithms. This includes problems such as Maximal Independent Set (MIS) and $(O(\log n), O(\log n))$-network decomposition. Getting a $\text{poly}(\log n)$-round deterministic algorithm for these would resolve several well-known open question of distributed graph algorithms including that of Linial from 1987 about polylogarithmic deterministic MIS [24], and many of the open problems in the book of Barenboim and Elkin [6].

1.1.3 Network Decompositions and Neighborhood Covers

Network decompositions, first introduced by Awerbuch et al. [3], have been a key tool in developing efficient (deterministic) distributed algorithms for a variety of distributed algorithms. Given an n-node graph $G = (V, E)$, a $(d(n), c(n))$-network decomposition of G partitions it into a $c(n)$ vertex-disjoint subgraphs, known as *blocks* of the decomposition (and indicated via different colors), such that in the subgraph induced by each block, each connected component (which is known as a *cluster* of this block) has a diameter at most $d(n)$. See Section 2 for the more formal definition. Awerbuch et al. [3] gave a deterministic algorithm with round complexity $2^{O(\sqrt{\log n \log\log n})}$ for computing a $(d(n), c(n))$-network decomposition with $d(n) = c(n) = 2^{O(\sqrt{\log n \log\log n})}$. This was later improved by Panconesi and Srinivasan [27] to a $2^{O(\sqrt{\log n})}$-round LOCAL algorithm for decomposition with $d(n) = c(n) = 2^{O(\sqrt{\log n})}$. While the algorithm of [3] works in the CONGEST model, that of [27] requires large messages. See also the work of Barenboim et al. [7, Corollary 5.4], where a generalized tradeoff of network decomposition in the CONGEST model is presented.

All of these decomposition algorithms [3, 7, 27] fail to work in the CONGEST model when we need a larger separation between the clusters of the same block, i.e., when their distance should be two or more hops. This is actually something that significantly limits the power of these network decompositions for CONGEST model algorithms (e.g., for the applications in spanners and dominating sets).

As our third contribution, we present a CONGEST model network decomposition algorithm that can be used to compute a decomposition such that clusters of the same block are at least k hops apart. The statement of the result is presented below as Theorem 3, the proof of which is deferred to the full version [15], due to space limitations.

▶ **Theorem 3.** *Let* $G = (V, E)$ *be an* n*-node graph and let* $k \geq 1$ *be an integer. There is a deterministic* CONGEST*-model algorithm that computes a strong diameter* k*-hop* $(k \cdot f(n), f(n))$*-decomposition of* G *in* $k \cdot f(n)$ *rounds, where* $f(n) = 2^{O(\sqrt{\log n \cdot \log\log n})}$.

In the above theorem, k-hop indicates that the clusters of the same block are at least k hops apart. See Section 2 for the more formal definition. The above theorem leads to the first efficient deterministic CONGEST model algorithm for neighborhood covers, another basic and central graph structure, which was introduced by Awerbuch and Peleg [5]. We refer to Section 2 for the related technical definition.

▶ **Corollary 4.** *Assume that we are given a strong diameter $2k$-hop (d, c)-decomposition of a graph G. One can compute a c-sparse k-neighborhood cover of diameter $d + k$ in $O(c(d + k))$ rounds in the* CONGEST *model on G. Consequently, for every $k \geq 1$, one can deterministically compute a $2^{O(\sqrt{\log n \cdot \log\log n})}$-sparse k-neighborhood cover of diameter $k \cdot 2^{O(\sqrt{\log n \cdot \log\log n})}$ of an n-node graph G in $k \cdot 2^{O(\sqrt{\log n \cdot \log\log n})}$ rounds of* CONGEST.

1.2 Our Method in a Nutshell, and Comparison with Prior Methods

Our spanner and minimum dominating set algorithms are developed also via network decompositions. We depart from the standard methodology in two parts. To outline these changes, we first review the standard methodology of algorithms that use network decompositions. We then comment on its shortcomings and outline how we go around each issue.

The standard method for (deterministic) algorithms via Network Decomposition. A standard technique in developing (deterministic) distributed algorithms for local graph problems (formally including Locally Checkable Labelings [26] and any other problem that can be formulated similarly using local constraints) is via the concept of $(d(n), c(n))$-network decompositions. The generic way to use them is to process the blocks sequentially in $c(n)$ phases. In the i^{th} phase, for each connected component of the i^{th} block, one gathers the whole topology of the that component (and perhaps some extra information about neighboring nodes) in an elected center of the component, make that node decide about all (local) decisions of the nodes of the component, and deliver this information back to the nodes. Since different components are disconnected from each other, their decisions do not influence each other and thus can be performed in parallel.

Shortcomings of the Generic Method via Network Decompositions for CONGEST. The method is perfect for the LOCAL model with large messages. However, when it comes to using small messages – i.e., in the CONGEST model – the method has two shortcomings:

Issue 1 – decompositions on power graphs. For many local problems, the constraints are not only about the direct neighbors of a node but a small neighborhood of a distance $r \geq 2$. For instance, as we will see, in the case of spanner computations this radius r can be as large as $O(\log n)$. In such cases, we need to ensure that connected components of each block are r hops away from each other, instead of just not being adjacent. This is (almost) the same as computing a network decomposition of G^r, which denotes the graph with an edge between two vertices if their distance is at most r hops[1]. The algorithm provided by Awerbuch et al. [3] for computing network decompositions does not seem to extend to computing a decomposition for G^r, because of the congestion that the algorithm creates[2]. We present a CONGEST-model algorithm for network decompositions of power graphs G^r; the formal statement is Theorem 3.

[1] Almost! Technically, we need that the components of each block are also connected in the base graph G so that we can run CONGEST model algorithms in each component independently, i.e., each edge of G has to pass messages for one component.

[2] We note that this is not a small technicality: The CONGEST model complexity of problems can be significantly different between G and even G^2. For instance, the problem of each node knowing the its degree in G and G^2 are very different. The former has a single-round CONGEST-algorithm, while there is no known $o(n)$-round algorithm for the latter.

Issue 2 – gathering topology in each component. The generic method of using network decompositions, each component is solved by gathering the whole topology of the component (and some neighborhood outside) and then solving the problem in a brute-force centralized manner. One can argue that this brute-force centralized computation is quite a stretch for the notion of having a *distributed* method of solving the problem.

The method we use to go around this issue is a derandomization of randomized distributed algorithms, which can typically solve the local problems that we are considering in $\text{poly}(\log n)$ rounds. We outline the method here. Most parts are generic and applicable to various problems, except the last part, which is specific to the constraints of each problem. We observe that for many problems, including spanners and dominating set, the corresponding efficient randomized algorithm can be made to work with only $\text{poly}(\log n)$ bits of randomness, using concepts such as k-wise independence. We refer to these bits as the *seed of randomness*. Then, derandomization is just a matter of determining a deterministic assignment to these $\text{poly}(\log n)$ bits while preserving certain properties of the output of the randomized algorithm. For that purpose, following an approach of Luby [25], we use the method of conditional expectations to fix the bits one by one. The only remaining piece of the algorithm is to check whether a bit should be 0 or 1. This requires us to be able to learn, or estimate, the expected number of unsatisfied local constraints. This last part will be done using a method specific for each problem, depending on its constraints.

Comparison with the methods of Censor-Hillel, Parter, and Schwartzman [9] and Grossman and Parter [18]

We note that this second part of our contribution as described in issue 2 above – namely, the method of conditional expectation applied on a random algorithm that uses only $\text{poly}(\log n)$ bits of randomness overall – is inspired by the work of Luby [25] in parallel algorithms and the recent work of Censor-Hillel, Parter, and Schwartzman [9] in distributed CONGEST and CONGESTED-CLIQUE model algorithms. Let us explain how our approach differs from that of [9], thus allowing us to improve on the bounds of [18].

Censor-Hillel et al. [9] give an $O(D \log^2 n)$-round CONGEST algorithm for maximal independent set (MIS), by derandomizing the randomized MIS algorithm of [16], using a method of conditional expectation close to Luby [25]. The key difference there is that (1) the complexity depends on the global diameter D, (2) for MIS, each of the constraints in the method of conditional expectation spans only the neighbors of one node and therefore, computing an upper bound on the score/cost function is much easier. In our case, we want a complexity that is considerably sublinear in the diameter, which calls for network decompositions. Moreover, for spanner and dominating set (and presumably many other local problems), the constraints span k-hop neighborhoods for some $k \geq 2$, instead of direct neighborhood. This causes two challenges: (A) we need a network decomposition of the power graphs, which prior to our work was not known in the CONGEST model, as explained above in issue 1. (B) Even computing each part of the cost function now spans k-hop neighborhood for some $k \geq 2$, and evaluating it with CONGEST-model messages requires a different method.

Censor-Hillel et al. [9] also give a derandomzied spanner algorithm in the CONGESTED-CLIQUE model, where all node-pairs can communicate with each other, exchanging $O(\log n)$ bits per round. This also follows a derandomization method inspired by that of Luby [25]. However, again there two differences, which limit that result from extending to our setting: (A)

this derandomization does not need to work with network decompositions and especially power-graph network decompositions, because everything is within one hop in the CONGESTED-CLIQUE and one can share the seed of randomness to all nodes, (B) computing the score/cost function, which spans k-hop neighborhoods, is much easier in the CONGESTED-CLIQUE, because this model does not suffer from the locality constraint.

In both cases above, both of the issues appear quite non-trivial to us. Indeed, Censor-Hillel et al. [9] comment that the best deterministic CONGEST algorithm for spanners takes barely sublinear time, $O(n^{1-1/k})$ rounds to be precise. That is much higher than the sub-polynomial time that we achieve. This $O(n^{1-1/k})$ bound was improved to nearly $O(\sqrt{n})$ – $O(n^{1/2-1/k} \cdot 2^k)$ rounds to be precise – in the simultaneous work of Grossman and Parter [18], using a special and well-crafted deterministic method for constructing spanners, and particularly without attempting a derandomization. We now show that the derandomization techniques can be extended and improved, along with the strengthened power-graph network decomposition, to achieve a round complexity $2^{O(\sqrt{\log n \cdot \log\log n})}$ rounds.

Some Other Related Work. Ghaffari, Harris, and Kuhn [17] also use some variant of a method of conditional expectation to obtain derandomized distributed algorithms, but for all of their results, locality is the main topic, and their algorithms use large messages. Kawarabayashi and Schwartzman [20] present distributed derandomizations for some other problems, including max cut and max k-cut. These work by turning a sequential process to a distributed process by going through the colors of a certain (defective) graph coloring one by one. However, those methods cannot extend to the problems that we consider as there the score/cost functions are very local (spanning single neighborhoods), whereas in our case, the constraints span up to $\log n$-neighborhood, which means a suitable coloring would require even up to polynomial many colors.

2 Model and Definitions

Mathematical Notation. For a graph $G = (V, E)$ and two nodes $u, v \in V$, we define $d_G(u, v)$ to be the hop distance between u and v. For an integer $k \geq 1$, we define $G^k = (V, E')$ to be the graph with an edge $\{u, v\} \in E'$ whenever $d_G(u, v) \leq k$. Given a node $v \in V$, we use $N_{G,k}(v) := \{u \in V : d_G(u, v) \leq k\}$ to denote the set of nodes within distance k of v in G. For a node set $S \subseteq V$, we use the shorthand notation $N_{G,k}(S) := \bigcup_{v \in S} N_{G,k}(v)$ and we drop the subscript G if it is clear from the context. Throughout, we use $\ln(\cdot)$ to refer to natural logarithm and $\log(\cdot)$ to refer to logarithms to base 2. Moreover, for a graph $G = (V, E)$, integers $a \geq 1$ and $b \geq 0$ and a node set $V' \subseteq V$, a set of nodes $S \subseteq V'$ is called a (a, b)-*ruling set* of G w.r.t. V' [3] if (A) for any two nodes $u, v \in S$, we have $d_G(u, v) \geq a$, and (B) $\forall u \in V' \setminus S$, there is a node $v \in S$ such that $d_G(u, v) \leq b$. If $V' = V$, S is simply called an (a, b)-ruling set of G.

Network Decomposition. A network decomposition of a graph G is given by a clustering of G and a coloring of the graph induced by contracting each cluster. We therefore first define the notion of a *cluster graph*.

▶ **Definition 5** (Cluster Graph)**.** Given a graph $G = (V, E)$ and an integer parameter $d \geq 1$, an (N, d)-cluster graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of G is a graph that is given by a set of $N \geq 1$ clusters $\mathcal{V} := \{C_1, \ldots, C_N\} \in 2^V$ such that (a) the clusters $C_1, \ldots, C_N$ form a partition of V, (b)

each cluster C_i induces a connected subgraph $G[C_i]$ of G, (c) each cluster C_i has a leader node $\ell(C_i)$ that is known by all nodes of C_i, and (d) inside each cluster, there is a rooted spanning tree $T(C_i)$ of $G[C_i]$ that is rooted at $\ell(C_i)$ and has diameter at most d. There is an edge $\{C_i, C_j\}$ between two clusters $C_i, C_j \in \mathcal{V}$ if there is edge in G connecting a node in C_i to a node in C_j. The identifier $\mathrm{ID}(C_i)$ of a cluster C_i is its leader's ID.

Given a cluster graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of G and an integer $k \geq 1$, we say that two clusters $C, C' \in \mathcal{V}$ are *k-separated* if for any two nodes u and v of G such that $u \in C$ and $v \in C'$, we have $d_G(u, v) > k$. A strong-diameter *k-hop network decomposition* of a graph G is then defined as follows.

▶ **Definition 6** (Network Decomposition)**.** Let $G = (V, E)$ be a graph and let $k \geq 1$, $d \geq 0$, and $c \geq 1$ be integer parameters. A *strong diameter k-hop (d, c)-decomposition* of G is a (N, d)-cluster graph $\mathcal{G}$ of G for some integer $N \geq 1$ together with a coloring of the clusters of $\mathcal{G}$ with colors $\{1, \ldots, c\}$ such that any two clusters with the same color are k-separated.

Sparse Neighborhood Covers. The notion of sparse neighborhood covers as introduced by Awerbuch and Peleg [5] is closely related to network decompositions.

▶ **Definition 7** (Sparse Neighborhood Cover)**.** Let $G = (V, E)$ be a graph and let $k \geq 1$, $d \geq 1$, and $s \geq 1$ be three integer parameters. A *s-sparse k-neighborhood cover of diameter d* is a collection of clusters $C \subseteq V$ such that (a) for each cluster C, we have a rooted spanning tree of $G[C]$ of diameter at most d, (b) each k-neighborhood of G is completely contained in some cluster, and (c) each node of G is in at most s clusters.

As we explain in the proof of Corollary 4, any $2k$-hop (d, c)-decomposition leads to a c-sparse k neighborhood cover of diameter $d + k$.

3 Hitting Set

In this section, we define an abstract problem, which we call the *hitting set* problem. This problem, which can be solved easily using randomized algorithms, captures a variety of the usual applications of randomness in distributed algorithms. In this section we provide a deterministic algorithm for solving this *hitting set* problem. In the later sections, we see how to use this deterministic subroutine to develop deterministic algorithms for other problems such as spanners and minimum dominating set, primarily by replacing their randomized parts with this deterministic hitting set subroutine.

Our main formulation of the hitting set problem (which is presented below in Definition 8 and solved in Lemma 9) is tailored to its usage in our spanner result. At the end of this section, in Lemma 10, we provide an alternative formulation and the corresponding deterministic algorithm, which are more suitable for our minimum dominating set result. The proofs are quite similar.

▶ **Definition 8** (The Hitting Set Problem)**.** Consider a graph $G = (V, E)$ with two special sets of nodes $L \subseteq V$ and $R \subseteq V$ with the following properties: each node $\ell \in L$ knows a set of vertices $R(\ell) \subseteq R$, where $|R(\ell)| = \Theta(p \log n)$, such that $dist_G(\ell, r) \leq T$ for every $r \in R(\ell)$. Here, p and T are two given integer parameters in the problem. Moreover, there is a T-round CONGEST algorithm that can deliver one message from each node $r \in R$ to all nodes $\ell \in L$ for which $r \in R(\ell)$. We emphasize that the same message is delivered to all nodes $\ell \in L$.

Given this setting, the objective in the hitting set problem is to select a subset $R^* \subseteq R$ such that (I) R^* dominates L – i.e., each node $\ell \in L$ has at least one node $r^* \in R^*$ such that $\ell \in R(r^*)$ – and (II) we have $|R^*| \leq |R|/p$.

▶ **Lemma 9.** *Given a 2T-hop (d, c)-decomposition of the graph G of the hitting set problem, there is a deterministic distributed algorithm that in $\tilde{O}(c(d+T))$ rounds solves the hitting set problem.*

Proof. The trivial randomized algorithm includes each node of R in R^* with probability $1/(2p)$. It is easy to verify that this satisfies the requirements (I) and (II), with high probability. In this proof, we develop a deterministic algorithm for solving the hitting set problem, effectively by derandomizing this randomized process. This derandomization has four aspects, which we discuss one by one.

Point 1 – Transforming the Requirements to One Cost Function. We try to capture the requirements (I) and (II) with one cost function. In particular, we define a *cost function* for any fixed set $R^* \subseteq R$ as follows. Consider the following indicator (random) variables: for each node $\ell \in L$, define $x_\ell = 1$ iff $R(\ell) \cap R^* = \emptyset$. Moreover, for each node $r \in R$, define $y_r = 1$ iff $r \in R^*$. Define the cost function as $Z = \sum_{\ell \in L} x_\ell + \sum_{r \in R} y_r$. Notice the value is clearly a function of the choice of $R^* \subseteq R$. Furthermore, it is easy to see that in the natural randomized algorithm that includes each node of R in R^* with probability $1/(2p)$, we have $\mathbb{E}[Z] \leq |R|/(2p) + 1/n^2$. This is because $\mathbb{E}[\sum_{r \in R} y_r] = \sum_{r \in R} \mathbb{E}[y_r] = \sum_{r \in R} 1/(2p) = R/(2p)$. Moreover, for each $\ell \in L$, we have $\mathbb{E}[x_\ell] = Pr[x_\ell = 1] = (1 - 1/(2p))^{\Theta(p \log n)} \leq 1/n^3$, which implies $\mathbb{E}[\sum_{\ell \in L} x_\ell] \leq 1/n^2$.

During the next three points presented below, we will describe a deterministic process for selecting R^* such that the related cost is at most $|R|/(2p) + 1/n$. Notice that this still does not mean that R^* satisfies (I). To take care of that issue, we perform the following clean up step, which has round complexity T, at the end: Suppose we have already chosen a subset $R^* \subseteq R$ such that the cost $Z = \sum_{\ell \in L} x_\ell + \sum_{r \in R} y_r$ of this selected set R^* is at most $|R|/(2p) + 1/n$. The number of nodes $\ell \in L$ for which $R(\ell) \cap R^* = \emptyset$ is $\sum_{\ell \in L} x_\ell$. By definition, these are exactly the vertices for which requirement (I) is not satisfied. For each such node ℓ, we mark one node $r \in R(\ell)$ arbitrarily and add the marked nodes to R^*. This can be done in T rounds by reversing the communication from R to L, now delivering one bit to each node $r \in R$ of whether any of the nodes $\ell \in L$ for which $r \in R(\ell)$ marked r or not. These marked nodes, which are added to R^*, increase the size of R^* by at most $\sum_{\ell \in L} x_\ell$. Thus, the total new size of R^* is at most $\sum_{r \in R} y_r + \sum_{\ell \in L} x_\ell \leq |R|/(2p) + 1/n \leq R/p$. Hence, now we have a set R^* that satisfies all the requirements (I) and (II).

Point 2 – Limited Independence Suffices. Next, we describe how we deterministically select a set R^* with cost at most $Z \leq |R|/(2p) + 1/n$. To be able to pick such a set R^* deterministically, it is helpful to have a randomized process that uses only a small number of random bits. For this reason, we first explain how to replace the fully random process of selecting R^* nodes with another random process that uses less randomness, in a certain sense to be formalized, but still provides the same guarantee on the expected cost. We will then derandomize this randomness-efficient random process.

Let us think of the decisions of whether a node $r \in R$ is included in R^* or not as a function $f : R \to \{0, 1, 2, \ldots, 2p-1\}$ where $f(r) = 0$ means $r \in R^*$ and all other values mean $r \notin R^*$. Notice that if for each $r \in R$, $f(r)$ is chosen uniformly at random

from $\{0, \dots, 2p-1\}$, then we have $\Pr[r \in R^*] = 1/(2p)$, as desired. Following standard terminology, we say that a family $\mathcal{F}$ of functions $f : R \to \{0, 1, 2, \dots, 2p-1\}$ is k-wise independent if for any set $S = \{s_1, s_2, \dots, s_k\} \subset R$ with $|S| = k$ and any choice of values $b_1, b_2, \dots, b_k \in \{0, 1, 2, \dots, 2p-1\}$, we have that

$$\Pr_{f \in_U \mathcal{F}}[f(s_1) = b_1 \& \dots f(s_k) = b_k] = (1/(2p))^k.$$

That is, upon selecting a function f uniformly at random from $\mathcal{F}$, the probability of the values of f over set S is exactly the same as in the fully random function. The advantage of k-wise independent functions is that the corresponding family is quite small and thus, we can choose one function in the family using considerably less randomness. This is made more clear in the next point. Moreover, they still provide many of the nice behaviors expected from truly random functions. In particular, using the extensions of standard Chernoff bound to functions with limited independence [30], we can see that if the selection function for choosing R^* out of R is $k = \Theta(\log n)$-wise independent (i.e., if it is chosen randomly from a k-wise independent family), then we still have a concentration within a constant factor what would be implied by the standard Chernoff bound. More concretely, we still have $Pr[x_\ell = 1] \leq 1/n^3$ for each $\ell \in L$. Hence, even with a k-wise independent selection function f, we have that the expected cost is small as desired, i.e., $\mathbb{E}[Z] \leq |R|/(2p) + 1/n^2$.

Point 3 – Defining a k-wise Independent Selection Process. To define a k-wise independent selection function in a manner that is suitable for our network decomposition, we use an independent function for the vertices of each cluster C of the decomposition. Hence, we have full independence among different colors and even among clusters of the same color. However, inside each cluster C, the selections are made using one k-wise independent function $g(C) : R \cap C \to \{1, 2, \dots, 2p\}$. One can easily see that such a combination of independent random functions, each of which is k-wise independent, is also a k-wise independent function.

To select a k-wise independent selection function for cluster C, we rely on classic constructions of k-wise independent functions. It is known [1] that there is a family $\mathcal{G}$ of $n^{O(k)}$ deterministic functions such that if we pick one function from $\mathcal{G}$ uniformly at random, we have a k-wise independent random function. This family can be known to all nodes of the cluster; they can all construct it by following the deterministic sequential construction of [1]. To randomly and uniformly sample one member of this family $\mathcal{G}$, which has $n^{O(k)}$ members, merely $O(k \log n)$ bits of randomness suffice. Hence, by using a random function defined via $O(k \log n) = O(\log^2 n)$ bits of randomness for each cluster, we can define a random selection function for vertices of R which ensures that $\mathbb{E}[Z] \leq |R|/(2p) + 1/n^2$.

Point 4 – Fixing the Bits of Randomness. We now fix the bits of randomness in the above random selection of R^*, in c phases. In the i^{th} phase, we decide about the vertices of R that are in the i^{th} color of network decomposition, whether to include each of them in R^* or not. This gradual process will be such that, at each point of time, the conditional expectation of the cost function, conditioned on the already decided vertices, is at most $|R|/(2p) + 1/n^2$. Hence, once we finish the process, a set R^* is selected with cost at most $|R|/(2p) + 1/n^2$.

Fix a color i. We fix the bits of randomness in each cluster of color i. Since clusters of this color are at least $2T$ hops apart in G, each variable x_ℓ or y_v in the cost function $Z = \sum_{\ell \in L} x_\ell + \sum_{r \in R} y_r$ is influenced by the randomness fixing of at most one cluster. Hence, each cluster C can fix its own randomness independent of the other clusters.

Let us focus on one cluster C in color i. We have a family of $\mathcal{G}$ of $n^{O(k)}$ deterministic functions for the selection of the R^*-nodes among $R \cap C$. We pick one function from $\mathcal{G}$ by fixing the corresponding bits of randomness one by one, in a manner that does not increase the conditional expectation of Z, given prior assignments. Imagine that all the functions in the family $\mathcal{G}$ are indexed with numbers from 1 to $n^{O(k)}$, and suppose that these indices are written as binary numbers with $O(k \log n)$ bits. Consider the process of fixing the first bit; the next bits are similar. Break the family $\mathcal{G}$ of $n^{O(k)}$ assignment functions into two subfamilies, $\mathcal{G}_0$ which are those that their function index starts with bit 0, and $\mathcal{G}_1$ which are those that their function index starts with bit 0. For each subfamily, we compute the conditional expectation of Z over the variables in $N^T(C)$ – i.e., the T-hop neighborhood of cluster C – when the assignment function is chosen uniformly at random from this subfamily. We then fix the first bit of randomness according to whichever leads to a smaller expectation, i.e., that is, we zoom in to one of subfamilies $\mathcal{G}_0$ and $\mathcal{G}_1$, in our search for a deterministic assignment function. We next explain why the expectation of Z over the variables in $N^T(C)$ can be computed in $O(d+T)$ time.

We first spend T rounds to deliver one message from each node $r \in R$ to all nodes $\ell \in L$ for which $r \in R(\ell)$. In this message, node r reports its color and cluster center ID, and whether node r has been put in R^* or not if the color of r was some $j < i$. Thus, each node ℓ in $N^T(C) \cap L$ can learn whether it is already hit or not, i.e., whether any of the nodes in $R(\ell)$ in the previous color clusters has been fixed to be in R^* or not. If there is already some such node $r \in R(\ell) \cap R^*$, then $x_\ell = 0$ and it will not change. If not, the expectation of x_ℓ can change by the assignments in C. In this case, node ℓ can exactly compute $\mathbb{E}[x_\ell] = Pr[x_\ell = 1]$ because it knows all the nodes in $R(\ell)$, those of colors less than i that their decisions have been made in the previous phases, the identifiers of those that are being decided in this phase, the colors and cluster identifiers of those with colors greater than i which will be decided in the next phases, and also the subfamily $\mathcal{G}_0$ or $\mathcal{G}_1$ in consideration. Similarly, each node $r \in R \cap C$ can compute $\mathbb{E}[y_r]$ because that only depends on the identifier of the node r and the subfamily $\mathcal{G}_0$ or $\mathcal{G}_1$ in consideration. Then, we can spend d rounds to perform a convergecast on the tree of cluster C to gather the summation of these expectations at the root[3].

Once these two expectations are gathered at the root of the cluster C, we go with the smaller one and zoom into the corresponding subfamily, among $\mathcal{G}_0$ or $\mathcal{G}_1$. This fixes the first bit of randomness in C but does not increase the conditional expectation of the cost function compared to when the assignment function was chosen from $\mathcal{G}$. We then proceed to the next bit. After going through all the $O(k \log n) = O(\log^2 n)$ bits, which takes $O(d \log^2 n)$ rounds, we have fixed all the bits and thus we have chosen a deterministic assignment for the R vertices of cluster C in a manner that did not increase the conditional expectation of the cost function. This finishes the process for one color. We then proceed to the next color and perform a similar process. After going through all colors, which takes $\tilde{O}(c(d+T))$ rounds, we have found a set $R^* \subseteq R$ such that the cost $Z = \sum_{\ell \in L} x_\ell + \sum_{r \in R} y_r$ of this selected set R^* is at most $|R|/(2p) + 1/n$. As described in point 1 above, this set R^* can be augmented to satisfy all the requirements of the hitting set problem, in T additional rounds. ◀

[3] We note that in the CONGEST model, we may not be able to convergecast the full precision of the expectation, but may need to truncate it to $\Theta(\log n)$ bits of precision. This would increase the expectation by at most $1/\operatorname{poly}(n)$. This is negligible even over all the at most n iterations that we perform such a convergecast and subfamily selection.

A Modified Variant of Hitting Set. We can use a similar method to solve a slightly modified variant of the hitting set problem, as stated in the following lemma. The proof is deferred to the full version. We use this variant in our dominating set algorithm.

▶ **Lemma 10** (An Alternative Hitting Set Lemma). *Let $H = (L \cup R, E)$ be a bipartite graph and let $p \geq 1$ be an integer parameter. Further assume that there is a spanning tree of diameter D that spans all nodes of H and that we can use the edges in E and the spanning tree edges for communication. There is a deterministic $\tilde{O}(D)$-time* CONGEST*-model algorithm that selects a subset $R^* \subseteq R$ of the nodes in R such that the following conditions hold:*

(a) *For all nodes $u \in L$, the number of neighbors in R^* is at most $O(\deg(u)/p + \log n)$.*

(b) *For all nodes in $u \in L$ with $\deg(u) \geq cp \log n$ for a sufficiently large constant $c > 0$, at least one neighbor of u is in R^*.*

4 Spanners

Here, we present the proof of Theorem 1, i.e., we develop a deterministic distributed algorithm for computing spanners by derandomizing the algorithm of Baswana and Sen [8], using our hitting set. We first briefly recall the algorithm of Baswana and Sen.

Baswana-Sen's Spanner Algorithm. The algorithm has k levels, where we gradually build, and sometimes dissolve clusters. At level i, each cluster induces a tree of depth at most $i-1$ rooted at the corresponding cluster center. Initially, each node is one cluster. In the i^{th} level for $i \in \{1, 2, \ldots, k-1\}$, each cluster of the previous level is active with probability $n^{-1/k}$ and inactive otherwise. This randomized decision is made by the corresponding cluster center. Then, inactive clusters get dissolved and their nodes either join other clusters or get dropped from the algorithm permanently. For each node v in an inactive cluster, if it has a neighbor in an active cluster, then v joins the cluster of one such neighbor u, and adds the edge $\{v, u\}$ to the tree of that cluster. If v has no neighbor in an active cluster, then v gets dropped from the rest of the algorithm. But just before that, for each inactive cluster $\mathcal{C}$ that contains a neighbor of v, node v adds to the spanner one edge to some neighbor in $\mathcal{C}$. Moreover, for each cluster of this level, we add the corresponding tree rooted in the cluster center to the spanner. This finishes level i, and we then proceed to the next level. In the very last level, all clusters are considered inactive and we act accordingly.

Properties of the Spanner algorithm of Baswana and Sen.

(1) Round Complexity: Clearly, the i^{th} level can be implemented in $O(i)$ rounds of the CONGEST model and thus the whole algorithm takes $O(k^2)$ rounds.

(2) Stretch: Eventually, all clusters are dissolved. For each edge $\{v, u\}$ in the graph, suppose without loss of generality that v gets dropped from the clustering no later than u. Then, an edge is added to the spanner from v to some node w in the cluster of u. If $w = u$, edge $\{v, u\}$ is in the spanner. Otherwise, there is a alternate route to go from v to u in the spanner by going to w and then using the cluster tree of u at that level; potentially going from w to its cluster center and then coming back to u. Since the tree has depth at most $i - 1 \leq k - 1$, the whole path has length at most $2k - 1$. That is, edge $\{v, u\}$ has stretch at most $2k - 1$.

(3) Spanner Size: The total number of cluster tree edges, over all levels, is $O(nk)$. Each node gets dropped in some level, when it has no active neighboring cluster, and then adds one edge connecting it to each (inactive) neighboring cluster to the spanner. If the node has

more than $\Theta(n^{1/k} \log n)$ neighboring clusters, w.h.p., it will have an active neighboring cluster. So the number of added edges per node is with high probability no more than $\Theta(n^{1/k} \log n)$. This is also true for the last level as there the total number of clusters is $\Theta(n^{1/k})$, w.h.p. Hence, the total number of edges in the spanner is $O(kn^{1+1/k} \log n)$, w.h.p.[4].

Derandomization – Abstracting the Properties of the Random Selection. The only part of this algorithm that relies on randomness is the step of selecting active clusters. As can be seen in the analysis, it suffices that this (random) selection satisfies the following two properties, per level: (1) nodes that have more than $d = \Theta(n^{1/k} \log n)$ neighboring clusters will have at least one active cluster, (2) if the number of clusters in this level is $R \geq \Theta(n^{1/k} \log n)$, the number of active clusters is at most $R \cdot n^{-1/k}$. The former ensures that the number of edges added per node in a level $i \in \{1, 2, \ldots, k-1\}$ is at most $\Theta(n^{1/k} \log n)$. The latter follows from Chernoff bound. Because of having this property in all levels, it follows that the total number of clusters at the last level is $O(n^{1/k} \log n)$. Hence, the number of added edges per node in that level is $O(n^{1/k} \log n)$.

Derandomization via Deterministic Hitting Set Computations. We can formulate the above two properties as a direct instance of the hitting set problem discussed in Definition 8, as follows: We set $p = n^{1/k}$ and $T = i + 1 \leq O(\log n)$. Moreover, we make each node that has at least $d = \Theta(n^{1/k} \log n)$ neighboring clusters be one node in L and each cluster center one node in R. Clearly, each node $\ell \in L$ can know $\Theta(p \log n)$ nodes of R that are within its $i+1 \leq k+1 \leq O(\log n)$ hops, these are the vertices in $R(\ell)$. We can also deliver one message from each $r \in R$ to all vertices $\ell \in L$ for which $r \in R(\ell)$ in T rounds. For that, we simply do a broadcast in the cluster centered at r and then pass it on to all neighboring nodes including ℓ. These provide all that we need to set up the hitting set problem. Moreover, we also use a $2T$-hop (d, c)-decomposition of graph G, for $d = c = 2^{O(\sqrt{\log n \cdot \log\log n})}$, which can be computing using Theorem 3 in $2^{O(\sqrt{\log n \cdot \log\log n})}$ rounds. We can now invoke the deterministic hitting set algorithm of Lemma 9, which runs in $2^{O(\sqrt{\log n \cdot \log\log n})}$ rounds. That provides a subset $R^* \subseteq R$ with size at most $R/p = R \cdot n^{-1/k}$ such that each node $\ell \in L$ has at least one node in $R^* \cap R(\ell)$. That is, each node that has more than $\Theta(n^{1/k} \log n)$ neighboring clusters will have at least one active cluster. These satisfy the two properties abstracted above, thus providing us with a deterministic selection of active clusters in each iteration of Baswana-Sen, hence completing the proof of Theorem 1.

5 Minimum Set Cover and Dominating Set

Consider a set cover instance $(X, \mathcal{S})$ consisting of a set X of elements and a set $\mathcal{S} \subseteq 2^X$ of subsets of X such that $\bigcup_{A \in \mathcal{S}} A = X$. The objective of the minimum set cover problem is to select a subset $\mathcal{C} \subseteq \mathcal{S}$ of the sets in $\mathcal{S}$ such that $\bigcup_{A \in \mathcal{C}} A = X$ and such that the cardinality of $\mathcal{C}$ is minimized. As standard (see e.g., [2]), we model the set cover instance $(X, \mathcal{S})$ as a distributed graph problem by defining a bipartite network graph that has a node u_x for each element $x \in X$ and a node v_A for each set $A \in \mathcal{S}$ and that contains an edge $\{u_x, v_A\}$ whenever $x \in A$. We also note that one can solve the distributed minimum dominating set

[4] With slightly more care, one can show that this number is actually $O(kn^{1+1/k})$, with high probability.

Algorithm 1: Distributed Set Cover Algorithm.

$\mathcal{C} := \emptyset$ *// start with an empty set cover;*
for *stage* $i := 1, 2 \ldots, \lceil \log n \rceil$ **do**
  for *phase* $j := 1, 2, \ldots, \lceil \log n \rceil$ **do**
    for *step* $c := 1, 2, \ldots, c(n)$ **do**
      $\mathcal{S}_{i,c} := \{A \in \mathcal{S} : \delta(A) \geq n/2^i \text{ and } A \text{ is in cluster of color } c\}$;
      $X_{i,j,c} := \{x \in X : s(x, c, n/2^i) \geq n/2^j\}$;
      Select $\mathcal{S}' \subseteq \mathcal{S}_{i,c}$ such that;
        a) $\forall x \in X_{i,j,c} : \exists A \in \mathcal{S}' : x \in A$;
        b) $\forall x \in X : x$ uncovered $\implies |\{A \in \mathcal{S}' : x \in A\}| = O(\log n)$;
      $\mathcal{C} := \mathcal{C} \cup \mathcal{S}'$ *// add* $\mathcal{S}'$ *to the set cover*

problem on a graph $G = (V, E)$ by using a distributed set cover algorithm and applying it to the corresponding set cover instance (where each node $v \in V$ represents an element and a set and where the set corresponding to a node u contains u, as well as all neighbors of u in G). The network graph of the set cover instance for the dominating set problem on G is given by the bipartite cover of G and a CONGEST-model algorithm on the bipartite cover of G can be run on the CONGEST model on G in the same time.

In the following, we assume that we are given a set cover instance $(X, \mathcal{S})$ and that $G = (V_X \cup V_{\mathcal{S}}, E)$ is the bipartite n-node graph corresponding to the given set cover instance. We further assume that for some $d(n) \geq 1$ and $c(n) \geq 1$, a strong diameter 2-hop $(d(n), c(n))$-decomposition of G is given. Recall that for $d(n) = c(n) = 2^{O(\sqrt{\log n \log\log n})}$, such a decomposition can be computed in $2^{O(\sqrt{\log n \log\log n})}$ rounds on G (cf. Theorem 3).

We first describe the algorithm to compute a small set cover of $(X, \mathcal{S})$. The algorithm can be seen as a distributed variant of the well-known sequential greedy algorithm. The algorithm starts with an empty set cover and it consists of a sequence of steps in which several sets of $\mathcal{S}$ are added to the set cover in parallel. Throughout the algorithm, we trace some properties of the subproblem that still has to be solved. For every set $A \in \mathcal{S}$, we use $\delta(A)$ to denote the number of uncovered elements of A (i.e., at the beginning of the algorithm, we have $\delta(A) = |A|$ and at the end, we need to have $\delta(A) = 0$). Further, for every element $x \in X$, for each of the colors $c \in \{1, \ldots, c(n)\}$ of the given 2-hop decomposition of G, and for some parameter $d \geq 1$, we define the degree-d, color-c support $s(x, c, d)$ of x as follows. If x is already covered, we have $s(x, c, d) = 0$, otherwise, $s(x, c, d)$ is defined to be the number of sets $A \in \mathcal{S}$ such that $x \in A$, A is in a cluster of color c, and $\delta(A) \geq d$. The algorithm consists of $\lceil \log n \rceil$ *stages* $i = 1, 2 \ldots, \lceil \log n \rceil$ and each stage consists of $\lceil \log n \rceil$ *phases* $j = 1, \ldots, \lceil \log n \rceil$. The algorithm guarantees that throughout stage $i \in \{1, \ldots, \lceil \log n \rceil\}$, for all sets $A \in \mathcal{S}$, it holds that $\delta(A) < n/2^{i-1}$, i.e., in each stage, the upper bound on the maximum remaining set size is halved. Further, for each stage $i \in \{1, \ldots, \lceil \log n \rceil\}$ and each phase $j \in \{1, \ldots, \lceil \log n \rceil\}$, it holds that $s(x, c, n/2^i) < n/2^{j-1}$ for all $x \in X$ and all $c \in \{1, \ldots, c(n)\}$. Further, each phase consists of $c(n)$ steps. The pseudocode of the whole set cover algorithm is given by Algorithm 1.

▶ **Lemma 11.** *For all* $i, j \in \{1, \ldots, \lceil \log n \rceil\}$ *and all cluster colors* $c \in \{1, \ldots, c(n)\}$*, throughout stage* i *and phase* j *of Algorithm 1, it holds that*

(a) *for every* $A \in \mathcal{S}$*, we have* $\delta(A) < n/2^{i-1}$*,*
(b) *for every* $x \in X$*, we have* $s(x, c, n/2^i) < n/2^{j-1}$*,*
(c) *at the end of step* c*, for every* $x \in X$*, we have* $s(x, c, n/2^i) < n/2^j$*.*

Proof. We prove (a)–(c) by induction on i, j, and c. First, note that (a) holds for $i = 1$ because the bipartite graph G representing the set cover instance has n nodes. Because there needs to be at least one set and at least one element in every set cover instance, we thus have $|\mathcal{S}| < n$ and $|X| < n$. Further note that if (a) is true for some stage i, then (b) holds for the given stage i and $j = 1$ for the same reason. Also note that (b) and (c) always hold for all covered elements x because in this case we defined $s(x, c, n/2^i)$ to be 0.

We next prove that (b) implies (c). Step c of stage i and phase j guarantees that for each element $x \in X_{i,j,c}$ (i.e., for each element for which $s(x, c, n/2^i) \geq n/2^j$), there is a set $A \in \mathcal{S}'$ such that $x \in A$. Consequently x is covered after the step and thus $s(x, c, n/2^i) = 0$. By condition (c), after all the $c(n)$ steps of stage i and phase j, we have $s(x, c, n/2^i) < n/2^j$ and thus if $j < \lceil \log n \rceil$, condition (b) also holds for stage i and phase $j + 1$. To also prove the induction step for condition (a), consider the end of phase $j = \lceil \log n \rceil$ of stage i. By (c), we have $s(x, c, n/2^i) < n/2^{\lceil \log n \rceil} \leq 1$ and thus $s(x, c, n/2^i) = 0$ for all $x \in X$ and all $c \in \{1, \ldots, c(n)\}$. This implies that there is no set $A \in \mathcal{S}$ left with $\delta(A) \geq n/2^i$. ◀

▶ **Lemma 12.** *Given a strong diameter 2-hop $(d(n), c(n))$-decomposition of G, Algorithm 1 can be implemented deterministically in $\tilde{O}(d(n) \cdot c(n))$ rounds in the* CONGEST *model on G.*

Proof. The algorithm consists of $O(\log n)$ stages, $O(\log n)$ phases per stage, and $c(n)$ steps per phase. The total number of steps is therefore $O(c(n) \log^2 n) = \tilde{O}(c(n))$. To prove the claim of the lemma, we thus need to show that each step can be implemented in $\tilde{O}(d(n))$ rounds in the CONGEST model on G. Consider some stage i, some phase j, and some step c in stage i and phase j. Recall that $\mathcal{S}_{i,c} \subseteq \mathcal{S}$ contains all sets $A \in \mathcal{S}$ that are in clusters of color c of the given network decomposition and for which $\delta(A) \geq n/2^i$ at the beginning of step c of phase j of stage i. Let $X_{i,c}$ be the set of uncovered elements of the sets in $\mathcal{S}_{i,c}$. Consider the subgraph $G_{i,c}$ of the set cover graph G that is induced by nodes corresponding to the elements in $X_{i,c}$ and the sets in $\mathcal{S}_{i,c}$. Note that for some element $x \in X_{i,c}$, $s(x, c, n/2^i)$ is the degree of the corresponding node in $G_{i,c}$. The algorithm needs to select a subset $\mathcal{S}'$ of the sets in $\mathcal{S}_{i,c}$ such that for each $x \in X_{i,c}$, the number of selected sets containing x is at most $O(\log n)$ and for each $x \in X_{i,j,c}$, there is at least 1 set containing x selected. On the graph $G_{i,c}$, this translates into selecting a subset of the nodes v_A corresponding to the sets $A \in \mathcal{S}_{i,c}$ such that for each $x \in X_{i,c}$, the corresponding node u_x has at most $O(\log n)$ neighbors selected and if u_x has degree at least $n/2^j$, it has at least 1 neighbor selected. From Lemma 11, we further know that all nodes u_x in $G_{i,c}$ have degree at most $n/2^{j-1}$ and all nodes v_A have degree at most $n/2^{i-1}$. Selecting the subset of sets $\mathcal{S}'$ therefore exactly corresponds to the solving the problem given by Lemma 10 on graph $G_{i,c}$ with parameter $p = n/(\gamma 2^j \log n)$ for an appropriate constant $\gamma > 0$. Further note that because we are given a 2-hop $(d(n), c(n))$-decomposition, the parts of the graph $G_{i,c}$ corresponding to different clusters of color c are disjoint. We can therefore solve the problem of selecting nodes in $\mathcal{S}_{i,c}$ separately for each cluster of color c. Because each such cluster has a spanning tree of diameter $d(n)$, Lemma 10 implies that each step can be implemented in $\tilde{O}(d(n))$ rounds. ◀

▶ **Lemma 13.** *Algorithm 1 computes a solution for a given set cover instance that is with an $O(\log^2 n)$-factor of an optimal solution.*

Proof. The algorithm always computes a valid solution (i.e., a solution that covers all the elements): For $i = j = \lceil \log n \rceil$, condition (c) of Lemma 11 implies that $s(x, c, 1) = 0$ for all $x \in X$ and all $c \in \{1, \ldots, c(n)\}$. This can only be true if all elements $x \in X$ are covered.

To prove the bound on the approximation ratio, we use a standard *dual fitting* argument (see e.g. [31, Chapter 13]). In the step that covers an element $x \in X$, we assign a dual variable $y_x > 0$ to x such that at the end of the algorithm $\sum_{x\in X} y_x = |\mathcal{C}|$. Consider some step c of stage i and phase j and assume that the sets in $\mathcal{S}'$ are added to the set cover $\mathcal{C}$. Let $X' \subseteq X$ be the set of elements that were uncovered before step c of stage i and phase j and which are covered by the sets in $\mathcal{S}'$. For all $x \in X'$, we set the dual variable y_x to $y_x := |\mathcal{S}'|/|X'|$. This clearly implies that at the end $\sum_{x\in X'} y_x = |\mathcal{S}'|$ and thus at the end $\sum_{x\in X} y_x = |\mathcal{C}|$. Note that for all sets $A \in \mathcal{S}'$, we have $\delta(A) \geq n/2^i$. Because for each uncovered element $x \in X$, there are at most $O(\log n)$ sets $A \in \mathcal{S}'$ for which $x \in A$, we have $|X| = \Omega(|\mathcal{S}'| \cdot n/(2^i \log n))$. Because by condition (a) of Lemma 11 for all $A \in \mathcal{S}$, we have $\delta(A) \leq n/2^{i-1}$, for all $x \in X' \cap A$, we have $y_x = O(\log n)/\delta(A)$. At the end of the algorithm, we thus get that for every set $A \in \mathcal{S}$,

$$\sum_{x\in A} y_x = O(\log n) \cdot \sum_{\ell=1}^{|A|} \frac{1}{\ell} = O(\log^2 n).$$

Dividing all y_x-variables by $O(\log^2 n)$ gives a feasible solution to the dual LP of the standard set cover LP relaxation. By LP duality, the obtained set cover is within an $O(\log^2 n)$ factor of the optimal solution. ◀

▶ **Theorem 14.** *A $O(\log^2 n)$-approximation for the distributed set cover problem can be computed deterministically in $2^{O(\sqrt{\log n \cdot \log\log n})}$ rounds in the* CONGEST *model.*

Proof. We can compute a 2-hop $(2^{O(\sqrt{\log n\cdot\log\log n})}, 2^{O(\sqrt{\log n\cdot\log\log n})})$-decomposition deterministically in $2^{O(\sqrt{\log n\cdot\log\log n})}$ rounds in the CONGEST model, using Theorem 3. Having this, the theorem then directly follows from Lemmas 13 and 12. ◀

References

1. Noga Alon and Joel H Spencer. *The probabilistic method*. John Wiley & Sons, 2004.
2. Matti Åstrand and Jukka Suomela. Fast distributed approximation algorithms for vertex cover and set cover in anonymous networks. In *Proceedings of the twenty-second annual ACM symposium on Parallelism in algorithms and architectures*, pages 294–302. ACM, 2010.
3. B. Awerbuch, AV Goldberg, M. Luby, and S. Plotkin. Network decomposition and locality in distributed computation. In *FOCS*, pages 364–369, 1989.
4. Baruch Awerbuch. Complexity of network synchronization. *Journal of the ACM (JACM)*, 32(4):804–823, 1985.
5. Baruch Awerbuch and David Peleg. Sparse partitions. In *Proc. IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 503–513, 1990.
6. Leonid Barenboim and Michael Elkin. Distributed graph coloring: Fundamentals and recent developments. *Synthesis Lectures on Distributed Computing Theory*, 4(1):1–171, 2013.
7. Leonid Barenboim, Michael Elkin, and Cyril Gavoille. A fast network-decomposition algorithm and its applications to constant-time distributed computation. *Theoretical Computer Science*, 2016.
8. Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Structures & Algorithms*, 30(4):532–563, 2007.

9 Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. Derandomizing local distributed algorithms under bandwidth restrictions. In *31 International Symposium on Distributed Computing*, 2017.

10 Bilel Derbel and Cyril Gavoille. Fast deterministic distributed algorithms for sparse spanners. In *International Colloquium on Structural Information and Communication Complexity*, pages 100–114. Springer, 2006.

11 Bilel Derbel, Cyril Gavoille, and David Peleg. Deterministic distributed construction of linear stretch spanners in polylogarithmic time. In *International Symposium on Distributed Computing*, pages 179–192. Springer, 2007.

12 Bilel Derbel, Cyril Gavoille, David Peleg, and Laurent Viennot. On the locality of distributed sparse spanner construction. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pages 273–282. ACM, 2008.

13 Bilel Derbel, Mohamed Mosbah, and Akka Zemmari. Sublinear fully distributed partition with applications. *Theory of Computing Systems*, 47(2):368–404, 2010.

14 Paul Erdős. Some problems in graph theory. In *STUDIA SIC MATH. HUNGAR.* Citeseer, 1966.

15 M. Ghaffari and F. Kuhn. Derandomizing distributed algorithms with small messages: Spanners and dominating set. Technical Report 285, U. of Freiburg, Dept. of Computer Science, 2018. URL: `http://tr.informatik.uni-freiburg.de/reports/report285/report00285.pdf`.

16 Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Pro. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2016.

17 Mohsen Ghaffari, David G Harris, and Fabian Kuhn. On derandomizing local distributed algorithms. *arXiv preprint arXiv:1711.02194*, 2017.

18 Ofer Grossman and Merav Parter. Improved deterministic distributed construction of spanners. In *31 International Symposium on Distributed Computing*, 2017.

19 Lujun Jia, Rajmohan Rajaraman, and Torsten Suel. An efficient distributed algorithm for constructing small dominating sets. *Distributed Computing*, 15(4):193–205, 2002.

20 Ken-Ichi Kawarabayashi and Gregory Schwartzman. Adapting local sequential algorithms to the distributed setting. *arXiv preprint arXiv:1711.10155*, 2017.

21 Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local computation: Lower and upper bounds. *J. ACM*, 63(2):17:1–17:44, mar 2016.

22 Fabian Kuhn and Roger Wattenhofer. Constant-time distributed dominating set approximation. In *Proc. ACM Symp. on Principles of Distributed Computing (PODC)*, pages 25–32, 2003.

23 Christoph Lenzen and Roger Wattenhofer. Minimum dominating set approximation in graphs of bounded arboricity. In *International Symposium on Distributed Computing*, pages 510–524. Springer, 2010.

24 Nathan Linial. Distributive graph algorithms global solutions from local data. In *Proc. IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 331–335. IEEE, 1987.

25 Michael Luby. Removing randomness in parallel computation without a processor penalty. *Journal of Computer and System Sciences*, 47(2):250–286, 1993.

26 Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995.

27 Alessandro Panconesi and Aravind Srinivasan. Improved distributed algorithms for coloring and network decomposition problems. In *Proc. ACM Symp. on Theory of Computing (STOC)*, pages 581–592. ACM, 1992.

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

29 David Peleg and Jeffrey D Ullman. An optimal synchronizer for the hypercube. *SIAM Journal on computing*, 18(4):740–747, 1989.

30 Jeanette P Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-hoeffding bounds for applications with limited independence. *SIAM J. on Discrete Math.*, 8(2):223–250, 1995.

31 Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.