

Distributed Distance-Bounded Network Design Through Distributed Convex Programming^{*†}

Michael Dinitz¹ and Yasamin Nazari²

- 1 Johns Hopkins University, Baltimore, MD, USA
mdinitz@cs.jhu.edu
- 2 Johns Hopkins University, Baltimore, MD, USA
ynazari@jhu.edu

Abstract

Solving linear programs is often a challenging task in distributed settings. While there are good algorithms for solving packing and covering linear programs in a distributed manner (Kuhn et al. 2006), this is essentially the only class of linear programs for which such an algorithm is known. In this work we provide a distributed algorithm for solving a different class of convex programs which we call “distance-bounded network design convex programs”. These can be thought of as relaxations of network design problems in which the connectivity requirement includes a distance constraint (most notably, graph spanners). Our algorithm runs in $O((D/\epsilon) \log n)$ rounds in the \mathcal{LOCAL} model and with high probability finds a $(1+\epsilon)$ -approximation to the optimal LP solution for any $0 < \epsilon \leq 1$, where D is the largest distance constraint.

While solving linear programs in a distributed setting is interesting in its own right, this class of convex programs is particularly important because solving them is often a crucial step when designing approximation algorithms. Hence we almost immediately obtain new and improved distributed approximation algorithms for a variety of network design problems, including Basic 3- and 4-Spanner, Directed k -Spanner, Lowest Degree k -Spanner, and Shallow-Light Steiner Network Design with a spanning demand graph. Our algorithms do not require any “heavy” computation and essentially match the best-known centralized approximation algorithms, while previous approaches which do not use heavy computation give approximations which are worse than the best-known centralized bounds.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases distributed algorithms, approximation algorithms, convex programming

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2017.5

1 Introduction

Distributed network design is a classical type of distributed algorithmic problem, going back at least to the seminal work on distributed MST by Gallager, Humblet, and Spira [16]. By “network design”, we mean the class of problems which can be phrased as “given input graph G , find a subgraph H which has some property P , and minimize the cost of H ”. Clearly different properties P , and different notions of cost, lead to very different problems. One important class of problems are *distance-bounded* network design problems, where the property P is that certain pairs of vertices are within some distance of each other in H (where distance refers to the shortest-path distance). The most well-known type of distance-bounded

* Supported in part by NSF awards 1464239 and 1535887.

† A full version of this paper is available at <https://arxiv.org/abs/1703.07417>.



© Michael Dinitz and Yasamin Nazari;
licensed under Creative Commons License CC-BY

21st International Conference on Principles of Distributed Systems (OPODIS 2017).

Editors: James Aspnes, Alysson Bessani, Pascal Felber, and João Leitão; Article No. 5; pp. 5:1–5:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

network design problems are problems involving *graph spanners*, in which the distance requirement is that the distance in H for all (or certain) pairs is within a certain factor (known as the stretch) of their original distance in H . But there are many other important versions of distance-bounded network design, such as the bounded diameter problem [12] and the shallow-light Steiner tree/network problems [19].

Many of these problems are NP-hard, so they cannot be solved optimally in polynomial time even in the centralized setting unless $P=NP$. Thus they have been studied extensively from an approximation algorithms point of view, where we design algorithms which approximate the optimal solution but which run in polynomial time. For many of these problems, a key step in the best-known centralized approximation algorithm is solving a linear programming relaxation of the problem, and then rounding the optimal fractional solution into a feasible integral solution. Interestingly, it is relatively common for the rounding to be “local”: if we are in a distributed setting and happen to know the optimal fractional LP solution, then the algorithm used to round this to an integral solution can be accomplished with a tiny amount of extra time (either 0 or a small constant number of rounds). So the bottleneck when trying to make these algorithms distributed is solving the LP, not rounding it.

Solving LPs in distributed settings has received only a small amount of attention, since it unfortunately turns out to be extremely challenging in general. Most notably, Kuhn, Moscibroda, and Wattenhofer [21] gave an efficient distributed algorithm (in the *LOCAL* model of distributed computation) for packing/covering LPs. Unfortunately, the LPs used for distance-bounded network design are not packing/covering LPs¹, and hence we are not able to use their techniques. Floréen et al. [15] also studied a special class of linear programs, namely min-max LPs, in distributed settings, which also cannot be used for our problems of interest. In this paper we show how to solve these LPs (and convex generalizations of them) in the *LOCAL* model of distributed computation, which almost immediately gives the best-known results for a variety of distance-bounded network design problems.

In particular, for many network design problems (DIRECTED k -SPANNER, BASIC 3-SPANNER, BASIC 4-SPANNER, LOWEST-DEGREE k -SPANNER, DIRECTED STEINER NETWORK WITH DISTANCE CONSTRAINTS with spanning demands, and SHALLOW-LIGHT STEINER NETWORK with spanning demands) we give approximation algorithms which run in $O(D \log n)$ rounds (where D is the maximum distance bound) and have the same approximation ratios as in the centralized setting. Previous distributed algorithms for these problems with similar round complexity have either used “heavy” computations (non-polynomial time algorithms) at the nodes (in which case they can often do *better* than the best computationally-bounded centralized algorithm), or give approximation bounds which are asymptotically worse than the best centralized bounds. See Section 1.2 for more discussion of previous work.

1.1 Our Results

We give two main types of results. First, we give a distributed algorithm that (approximately) solves distance-bounded network design convex programs with small round complexity. We then use this result to (almost immediately) get improved distributed approximation algorithms for a variety of network design problems.

¹ They can be turned into packing/covering LPs through a projection operation, but unfortunately this technique results in an exponential number of constraints, making [21] inapplicable. However, this technique has been used in the centralized setting for the fault-tolerant directed k -spanner problem [9].

1.1.1 Solving convex programs

Stating our main technical result (distributed approximations of distance-bounded network design convex programs) in full generality requires significant technical setup, so we provide an informal description here. See Section 4 for the full definitions and theorem statements (Theorem 13 in particular). But informally, a distance-bounded network design convex program is the following. We are given a graph $G = (V, E)$, a set $\mathcal{S} \subseteq V \times V$, and for each $(u, v) \in \mathcal{S}$ there is a set of “allowed” $u - v$ paths $\mathcal{P}_{u,v}$. Roughly speaking, we typically assume that the allowed paths are short, and define D to be maximum length of such paths. Informally, the integral problem is to find a subgraph H of G so that every $(u, v) \in \mathcal{S}$ is connected by at least one path from $\mathcal{P}_{u,v}$ in H , and the goal is to minimize some notion of “cost”. If our notion of “cost” is captured by an objective function $g : \mathbb{R}_{\geq 0}^{|E|} \rightarrow \mathbb{R}$ (which is typically linear, but which can be more general convex functions as long as they satisfy a “partitionability” constraint – see Section 4 for the details), then the natural relaxation of this problem is the following convex program, which has a variable x_e for every edge and a variable f_P for every allowed path.

$$\begin{aligned}
 \min \quad & g(x) \\
 \text{s.t.} \quad & \sum_{P \in \mathcal{P}_{u,v}: e \in P} f_P \leq x_e && \forall (u, v) \in \mathcal{S}, \forall e \in E \\
 & \sum_{P \in \mathcal{P}_{u,v}} f_P \geq 1 && \forall (u, v) \in \mathcal{S} \\
 & x_e \geq 0 && \forall e \in E \\
 & f_P \geq 0 && \forall (u, v) \in \mathcal{S}, \forall P \in \mathcal{P}_{u,v}
 \end{aligned}$$

Informally, the first type of constraint says that an allowed path is included only if all edges in it are included, and the second type of constraint required us to include at least one allowed path for each $(u, v) \in \mathcal{S}$. We call this type of convex program a *distance-bounded network design convex program*. It is clearly not a packing/covering LP due to the first type of constraint, and hence there is no known distributed algorithm to solve this kind of program. However, note that if the maximum length of any allowed path is constant, then there are only a polynomial number of such paths, and hence the size of the convex program is polynomial and so it can be solved in polynomial time in the centralized setting under reasonable assumptions on g (see [17] for details on solving convex programs in polynomial time).

Our main technical result is that we can approximately solve these optimization problems even in a distributed setting. For any path P let $\ell(P)$ denote the length of the path (the number of edges in it).

► **Theorem 1.** *For any constant $\epsilon > 0$, any distance-bounded network design convex program can be solved up to a $(1 + \epsilon)$ -approximation in $O(D \log n)$ rounds in the LOCAL model, where $D = \max_{(u,v) \in \mathcal{S}} \max_{P \in \mathcal{P}_{u,v}} \ell(P)$. Moreover, if the convex program can be solved in polynomial time in the centralized sequential setting, then the distributed algorithm uses only polynomial-time computations at every node*

The dependence on ϵ in the above theorem is hidden in the $O(\cdot)$ notation – see Theorem 13 for the full statement.

Our main technique is to use a distributed construction of *padded decompositions*, a specific type of network decomposition which we explain in detail in Section 3. Padded decompositions have been very useful for metric embeddings and approximation algorithms

(e.g., [18, 20]), but to the best of our knowledge have not been used before in distributed algorithms (with the exception of [10], which used a special case of them to give a distributed algorithm for the fault-tolerant 2-spanner problem). Very similar decompositions, such as the famous Linial-Saks decomposition [22], have been used extensively in distributed settings, but the guarantees for padded decompositions are somewhat different (and we believe that these decompositions may prove useful in the future when designing distributed approximation algorithms). In Section 3 we give a distributed algorithm in the \mathcal{LOCAL} model to construct padded decompositions. These padded decompositions allow us to solve a collection of “local” convex programs with the guarantees that a) most of the demands in \mathcal{S} are satisfied in one of the local programs, and b) the solutions of the local convex programs combine into a (possibly infeasible) global solution with cost at most the cost of the global optimum. Then by averaging over $O(\log n)$ of these decompositions we get a feasible global solution which is almost optimal.

1.1.2 Distributed approximation algorithms for network design

Solving convex programming problems in distributed environments is interesting in its own right, and Theorem 1 is our main technical contribution, but the particular class of convex programs that we can solve are mostly interesting as convex relaxations of interesting combinatorial optimization problems. Many of the problems are NP-hard, but there has been significant work (some quite recent) on designing approximation algorithms for them (see, e.g., [9, 6, 11, 5]). Almost all of these approximations depend on convex relaxations which fall into our class of “distance-bounded network design convex programs”. This means that as long as the rounding scheme can be computed locally, we can design distributed versions of these approximation algorithms by using Theorem 1 to solve the appropriate convex relaxation and then using the local rounding scheme.

We are able to use this framework to give distributed approximation algorithms for several problems. Most of them are variations of *graph spanners*, which were introduced by Peleg and Ullman [26] and Peleg and Schäffer [25], and are defined as follows.

► **Definition 2.** Let $G = (V, E)$ be a graph (possibly directed), and let $k \in \mathbb{N}$. A subgraph H of G is a k -spanner of G if $d_H(u, v) \leq k \cdot d_G(u, v)$ for all $u, v \in V$. The value k is called the *stretch* of the spanner.

Before stating our results, we first define the problems. In the BASIC k -SPANNER problem we are given an undirected graph G and a value $k \in \mathbb{N}$. A subgraph H of G is a feasible solution if it is a k -spanner of G , and the objective is to minimize the number of edges in H . For $k = 3, 4$, the best-known approximation algorithm for this problem is $\tilde{O}(n^{1/3})$ [5, 11]. If the input graph G (and the solution H) are directed, then this is the DIRECTED k -SPANNER problem, for which the best-known approximation is $\tilde{O}(\sqrt{n})$ [5]. If the objective is instead to minimize the maximum degree in H then this is the LOWEST-DEGREE k -SPANNER problem, for which the best-known approximation is $\tilde{O}(n^{(1-1/k)^2})$ [6].

The following theorem contains our results on distributed approximations of graph spanners. Informally, it states that we can give the same approximations in the \mathcal{LOCAL} model as are possible in the centralized model.

► **Theorem 3.** *There are algorithms in the \mathcal{LOCAL} model that w.h.p.² provide the following guarantees. For DIRECTED k -SPANNER, the algorithm runs in $O(k \log n)$ rounds and gives*

² By “with high probability” (or w.h.p.), we mean with probability at least $1 - 1/n^c$ for some $c \geq 1$.

an $\tilde{O}(\sqrt{n})$ -approximation. For BASIC 3-SPANNER and BASIC 4-SPANNER, the algorithms run in $O(\log n)$ rounds and gives an $\tilde{O}(n^{1/3})$ -approximation. For LOWEST-DEGREE k -SPANNER, the algorithm runs in $O(k \log n)$ rounds and gives an $\tilde{O}(n^{(1-1/k)^2})$ -approximation. All of these algorithms use only polynomial-time computations at each node.

We emphasize that our algorithms for these spanner problems both match the best-known centralized approximations and only use polynomial-time computations at each node. There is significant previous work (see Section 1.2) on designing distributed approximation algorithms for these and related problems that has only one of these two properties, but all previous approaches which use only polynomial-time computations necessarily do worse than the best centralized bound (or have much worse round complexity). At a high level, this is because previous approaches (most notably [2]) do not actually use the structure of the centralized algorithm: they only use the efficient centralized approximation as a black box. By going inside the black box and noticing that they all use a similar type of convex relaxation, we can simultaneously get low round complexity, best-known approximation ratios, and efficient local computation.

It turns out that we can use our techniques for an even broader question: DIRECTED STEINER NETWORK WITH DISTANCE CONSTRAINTS with a spanning demand graph. In this problem there is a set $\mathcal{S} \subseteq V \times V$ of demands, and for every demand $(u, v) \in \mathcal{S}$ there is a length bound $L(u, v)$. The goal is to find a subgraph H so that $d_H(u, v) \leq L(u, v)$ for all $(u, v) \in \mathcal{S}$, and the objective is to minimize the number of edges in H . The state of the art centralized bound for this problem is a $O(n^{3/5+\epsilon})$ -approximation [7], but if we further assume that every vertex $u \in V$ is the endpoint of at least one demand in \mathcal{S} (which we will refer to as a *spanning demand graph*) then it is straightforward to see that the centralized algorithm of [5] for DIRECTED k -SPANNER can be generalized to give a $\tilde{O}(\sqrt{n})$ -approximation. Our distributed version of this algorithm also generalizes, w.h.p. giving the following result.

► **Theorem 4.** *There is an approximation algorithm in the LOCAL model for DIRECTED STEINER NETWORK WITH DISTANCE CONSTRAINTS with a spanning demand graph with approximation ratio $\tilde{O}(\sqrt{n})$ which runs in $O((\max_{(u,v) \in \mathcal{S}} L(u, v)) \log n)$ rounds and uses only polynomial-time computations.*

Note that DIRECTED k -SPANNER and BASIC k -SPANNER are special cases of this problem, where there is a demand for every edge and the length bound is just k times the original distance. Interestingly, other network design problems which have proved important for distributed systems are also special cases, including the DISTANCE PRESERVER problem (when $L(u, v) = d_G(u, v)$ for all $(u, v) \in \mathcal{S}$), the PAIRWISE k -SPANNER problem (where $L(u, v) = k \cdot d_G(u, v)$ for all $(u, v) \in \mathcal{S}$), and the SHALLOW-LIGHT STEINER NETWORK problem (where $L(u, v) = D$ for all $(u, v) \in \mathcal{S}$, for some global parameter D). SHALLOW-LIGHT STEINER NETWORK in particular is a key component in state of the art systems for reliable Internet transport [1], although in that particular application the demand graph is not spanning. Extending our techniques to handle totally general demands by giving a distributed version of [7] is an extremely interesting open question.

1.2 Related Work

While distributed solving of convex programs is a natural question, there is little previous work in the LOCAL model. Possibly most related to our results is a line of work on solving *positive* linear programs (packing and covering LPs). This was introduced by [23], improved by [4], and then essentially optimal upper and lower bounds were given by [21].

Unfortunately, the convex programs we consider are not positive linear programs due to the “capacity” constraints in which some variables appear with positive coefficients while others have negative coefficients.

A special case of our result was proved earlier in [10], who showed how to solve the LP relaxation of BASIC 2-SPANNER in the \mathcal{LOCAL} model in $O(\log^2 n)$ rounds (they actually show more than this, by giving a distributed algorithm for the *fault-tolerant* version of BASIC 2-SPANNER, but that is not germane to our results). Our techniques are heavily based on [10], which is itself based on the ideas from [21]. In particular, [21] uses a Linial-Saks decomposition [22] to solve “local” versions of the linear program in different parts of the graph, and then combines these appropriately. To make this work for the BASIC 2-SPANNER LP relaxation, [10] had to use *padded decompositions*, which can be thought of as a variant of Linial-Saks with slightly different guarantees which, for technical reasons, are more useful for network design LPs. In this paper we extend these techniques further by giving a more general definition of padded decomposition which works for larger distance requirements, showing how to construct them in the \mathcal{LOCAL} model, and then showing that the basic “combining” idea from [10] can be extended to handle these more general decompositions and far more general constraints and objective functions.

The major type of combinatorial optimization problem which our techniques allow us to approximate are various versions of graph spanners. There are an enormous number of papers on spanners in both centralized and distributed models, but fewer papers which attempt to find the “best” spanner for the particular given input graphs (most papers on spanners give existential results and algorithms to achieve them, rather than optimization results). These optimization questions (e.g., BASIC k -SPANNER, DIRECTED k -SPANNER, and LOWEST-DEGREE k -SPANNER) have been considered quite a bit in the context of centralized approximation algorithms and hardness of approximation [9, 5, 11, 6, 8], but almost all of the known centralized results use linear programming relaxations, making them difficult to adapt to distributed settings. Hence there have been only two results on optimization bounds in distributed models: [10] and [2].

Barenboim et al. [2] provided a distributed algorithm using Linial-Saks decompositions that for any integer parameters k, α , gives an $O(n^{1/\alpha})$ -approximation for DIRECTED k -SPANNER in $\exp(O(\alpha)) + O(k)$ time. This is an extremely strong approximation bound, and in fact is better than even the best centralized bound. This is possible due to their use of very heavy (exponential time) local computation. Our algorithms, on the other hand, take polynomial time for local computations. Barenboim et al. [2] show that heavy local computations can be removed from their algorithm by using a centralized approximation algorithm for a variant of spanners known as *client-server k -spanners*, and in particular that an $f(k)$ -approximation for client-server k -spanner can be turned into an $O(n^{1/\alpha} f(k))$ -approximation algorithm running in $\exp(O(\alpha)) + O(k)$ rounds in the \mathcal{LOCAL} model for minimum k -spanner with only polynomial local computation. So in order to achieve the same asymptotic approximation ratio as the best-known centralized algorithm, the parameter α must be $\Omega(\log n)$ and hence the running time is polynomial in n , even though k might be a constant. It is essentially known (though not written anywhere) that a variety of other results with slightly different tradeoffs can be achieved through similar uses of Linial-Saks [13] or refinements of Linial-Saks such as [14]. However, since all of these approaches treat the centralized approximation algorithm as a black box, none of them can achieve the same approximation ratio as the centralized algorithm without suffering a much worse (usually polynomial) round complexity than the $O(k \log n)$ that we achieve.

2 Preliminaries and Notation

The distributed setting we will be considering is the *LOCAL* model [24], in which time passes in synchronous rounds and in each round every node can send an arbitrary message of unbounded size to each of its neighbors in the underlying graph $G = (V, E)$ (as always, we will let $n = |V|$ and $m = |E|$). This is in contrast to the *CONGEST* model, where nodes can only send a message of size $O(\log n)$ to each of their neighbors in each round. We are not focusing on the *CONGEST* model in this paper. We will assume that all nodes know n (or at least know a constant approximation of n). Usually in this model the communication graph is the same as the graph of computational interest; e.g., we will be trying to compute a spanner of the communication graph itself. But for some applications we will want the graph to be directed, in which case we make the standard assumption that communication is bidirectional: the graph for which we are trying to compute a convex relaxation / network design problem is directed, but messages can be sent in both directions across a link. In other words, the communication graph is just the undirected version of the given directed graph.

For any pair of nodes $u, v \in V$ we define $d(u, v)$ to be the distance between u and v in the communication graph (i.e. the length of a shortest path between u and v regardless of edge directions). We define $B(u, k)$ to be an undirected ball of radius k from u in the communication graph. More precisely, $B(u, k) = \{w \in V \mid d(u, w) \leq k\}$. If x is a vector then we use x_i to denote the i 'th component of x . Most of the time our vectors will be indexed by edges in a graph, in which case we will also use the notation $(x_e)_{e \in E}$.

Given a partition of the vertices V of a graph, we will refer to each part of the partition as a “cluster”. For any graph $G = (V, E)$ and set $S \subseteq V$, we let $E(S)$ denote the set of edges in the subgraph induced by S , i.e., $E(S) = \{(u, v) \in E \mid u, v \in S\}$. We will frequently need “restrictions” of vectors to induced subgraphs, so for any vector $x \in \mathbb{R}^m$, we define $x^S = (x_e^S)_{e \in E}$ to be the vector in \mathbb{R}^m where $x_e^S = 0$ if $e \notin E(S)$ and $x_e^S = x_e$ if $e \in E(S)$.

3 Padded decompositions

We will now define and give an algorithm to construct *padded decompositions*, which are one of the key technical tools that we will use when designing algorithm to solve distance-bounded network design convex programs. In this section all graphs are undirected and all distances are with respect to this undirected graphs (in fact, the definition and our algorithm work more generally for any metric space). Recall that $B(u, k)$ denotes the undirected ball of radius k from node u (in the communication graph), and $diam(C) = \max_{u, v \in C} d_G(u, v)$, which is often called the weak diameter. Intuitively, a (k, ϵ) -padded decomposition partitions a graph into clusters, where nodes in each cluster are not too far in the original graph, and balls of a radius k are preserved with probability at least $1 - \epsilon$.

► **Definition 5.** Given an *undirected* graph G , a (k, ϵ) -padded decomposition, where $0 < \epsilon \leq 1$, is a probability measure μ over the set of graph partitions (clusterings) that has the following properties:

- 1) For every $P \in \text{supp}(\mu)$,³ and every cluster $C \in P$, we have: $diam(C) \leq O((k/\epsilon) \log n)$.
- 2) For every $u \in V$, it holds that $\Pr(\exists C \in P \mid B(u, k) \subseteq C) \geq 1 - \epsilon$. That is to say, the probability that all nodes in $B(u, k)$ are in the same cluster is at least $1 - \epsilon$.

³ By $\text{supp}(\mu)$ we mean the set of partitions that have non-zero probability.

Algorithm 1: Sampling from a (k, ϵ) -padded decomposition of $G = (V, E)$.

- 1 Let $\pi : V \rightarrow [n]$ be an arbitrary bijection from V to $[n]$, and let $r = (\frac{2}{\epsilon})k$.
 - 2 **for** $v \in V$ **do**
 - 3 Sample z_v independently from a distribution with probability density function

$$p(z_v) = \binom{n}{n-1} \frac{e^{-z_v/r}}{r}.$$
 - 4 Set the radius $r_v = \min(z_v, r \ln n + k)$.
 - 5 **for** $u \in V$ **do**
 - 6 Node u joins cluster $C(v)$, such that

$$d(v, u) \leq r_v \wedge (\pi(v) < \pi(w) \forall w \neq v \text{ s.t. } d(w, u) \leq r_w). // \text{ Node } u \text{ joins}$$
 the cluster $C(v)$, with cluster center v , which is the first node in the permutation where $d(v, u) \leq r_v$.
-

This notion of padded decompositions is standard in metric embeddings and approximation algorithms [18, 20], but to the best of our knowledge has not yet been used in distributed algorithms. We first use a centralized algorithm (Algorithm 1) to sample from a (k, ϵ) -padded decomposition, and then describe how it can be implemented in the \mathcal{LOCAL} model. Algorithm 1 and its analysis are similar to a partitioning algorithm proposed in [3], which was shown to have a low probability of separating nodes in a close neighborhood. Due to space constraints, proofs can be found in Appendix A.

For any partition P constructed by Algorithm 1, each cluster is clearly $C(v)$ for some $v \in V$. We call this special node v the *center* of cluster $C(v)$. Later, we will use the center of each cluster for solving locally defined convex programs.

► **Lemma 6.** *Algorithm 1 partitions a given undirected graph $G = (V, E)$ into a partition P such that P is sampled from a (k, ϵ) -padded decomposition.*

We will now use an idea similar to the one used in [10] to make Algorithm 1 distributed. In [10] they only considered the special case of $k = 1$ and $\epsilon = 1/2$, which is why we cannot simply use their result as a black box.

► **Lemma 7.** *There is an algorithm in the \mathcal{LOCAL} model that runs in $O(\frac{k}{\epsilon} \ln n)$ rounds and samples from a (k, ϵ) -padded decomposition (so every node knows the cluster that it is in).*

Proof. Without loss of generality, we assume that all nodes have unique IDs⁴. The sequence of IDs in ascending order will determine the permutation π used in Algorithm 1, i.e. if $ID_u < ID_v$ then $\pi(u) < \pi(v)$. The algorithm proceeds as follows until all nodes have been assigned to a cluster: each node $u \in V$ chooses a radius r_u based on the distribution defined in Algorithm 1. Then every $u \in V$ simultaneously sends a message containing ID_u to all nodes in $B(u, r_u)$. After receiving all the messages, each node chooses the node with the smallest ID as the cluster center. Then Lemma 6 implies that the clusters satisfy the properties of a (k, ϵ) -padded decomposition. Since the radius that each node chooses is $O((k/\epsilon) \log n)$, and each node only communicates with nodes within its radius, the running time in the \mathcal{LOCAL} mode is $O((k/\epsilon) \log n)$. ◀

⁴ We assume this since nodes can each draw an ID from a suitably large space, so the probability of a collision is small enough that it does not affect the guarantees required by a padded decomposition.

4 Distributed distance bounded network design convex programming

In this section we prove Theorem 1, giving an algorithm similar to [10] which can almost optimally solve distance-bounded network design convex programs. We first make all definitions formal in Section 4.1, and in particular define formally the class of objective functions where our results hold. Then in Section 4.2 we give a distributed algorithm which solves these programs up to arbitrarily small error. All missing proofs can be found in Appendix B.

4.1 Distance bounded network design convex programs

We will first describe a general class of objective functions that our algorithm applies to. For a graph $G = (V, E)$ and a set $S \subseteq V$, we let $E(S)$ denote the set of edges in the subgraph of G induced by S . Recall that for a vector $x \in \mathbb{R}^m$ (where $m = |E|$), we define $x^S = (x_e^S)_{e \in E} \in \mathbb{R}^m$ to be the vector where $x_e^S = 0$ if $e \notin E(S)$ and $x_e^S = x_e$ if $e \in E(S)$.

► **Definition 8.** Given a graph $G = (V, E)$, a function $g : \mathbb{R}^m \mapsto \mathbb{R}$ is *convex partitionable* with respect to G if g is a non-decreasing⁵ and convex function with the following property: for all partitions $\sigma = \{\sigma_1, \dots, \sigma_\ell\}$ of nodes in V , there exists a non-decreasing function $h_\sigma : \mathbb{R}^\ell \mapsto \mathbb{R}$, s.t. $g(x) = h_\sigma(g(x^{\sigma_1}), g(x^{\sigma_2}), \dots, g(x^{\sigma_\ell}))$ for all $x = (x_e)_{e \in E}$ where $x_e = 0$ for any edge e with endpoints in different clusters of σ (equality does not need to hold for vectors x with nonzero values on edges between clusters).

Convex partitionable functions for graphs are a natural class of functions for distributed computing purposes. Moreover, this class includes many types of objective functions that are of interest in network design problems, including p -norms and linear functions. For example, if the function g is the p -norm with $p \in \mathbb{Z}_{\geq 0}$, then it is easy to verify that by setting the function h_σ to also be the p -norm for any partition σ of V , the conditions of Definition 8 are satisfied. Note, since we consider non-negative values, an unweighted sum is just the 1-norm, and the max function is the infinity norm, and hence they will also satisfy the conditions of Definition 8. Similarly, in case of linear functions, it is easy to see that conditions of Definition 8 are satisfied by setting h_σ to be the *unweighted* sum.

There are also other, less trivial examples. For example, it is not hard to show the p -norm of the *degree vector* (rather than just the edge vector) is also convex partitionable with respect to G . An important special case of this is the ∞ -norm of the degree vector, i.e., the maximum degree. Since we use this objective in some of our applications (i.e., for the LOWEST-DEGREE k -SPANNER problem), we give a short proof of this case in Appendix B. For an integral vector $x \in \mathbb{R}^m$ we can write $g(x) = \max_{v \in V} \deg(v)$. By generalizing this notation to all $x \in \mathbb{R}^m$, we can define fractional node degrees as $\deg(v) = \sum_{u:(v,u) \in E} x_{(v,u)}$ ⁶.

► **Lemma 9.** *Given a graph $G = (V, E)$, the function $g(x) = \max_{v \in V} (\sum_{u:(v,u) \in E} x_{(v,u)})$ is convex partitionable w.r.t. G .*

Now that this class of functions has been defined, we can formally define the class of distance-bounded network design convex programs.

⁵ Let $f(x_1, \dots, x_k)$ be a multivariate function. We will say f is nondecreasing if the following holds: if $x_i \leq x'_i$ for all $1 \leq i \leq k$, then $f(x_1, \dots, x_k) \leq f(x'_1, \dots, x'_k)$.

⁶ Here we are considering out-degree of nodes in a directed graph. It is easy to see that Lemma 9 also holds in cases of in-degree only or sum of out-degree and in-degree. The latter is what are interested in for Section 5.

► **Definition 10.** Let $\mathcal{S} \subseteq V \times V$ be a set of pairs in the graph $G = (V, E)$, and for any pair $(u, v) \in \mathcal{S}$ let $\mathcal{P}_{u,v}$ be a set of paths from u to v , which we sometimes call the set of “allowed” paths. Let g be a non-decreasing convex-partitionable function of $x = (x_e)_{e \in E}$ with $g(\vec{0}) = 0$. Then we call a convex program of the following form a *distance bounded network design CP*:

$$\begin{aligned}
\min \quad & g(x) \\
\text{s.t.} \quad & \sum_{P \in \mathcal{P}_{u,v}: e \in P} f_P \leq x_e && \forall (u, v) \in \mathcal{S}, \forall e \in E \\
& \sum_{P \in \mathcal{P}_{u,v}} f_P \geq 1 && \forall (u, v) \in \mathcal{S} \\
& x_e \geq 0 && \forall e \in E \\
& f_P \geq 0 && \forall (u, v) \in \mathcal{S}, \forall P \in \mathcal{P}_{u,v}
\end{aligned}$$

As we will see in Section 5, many network design problems use linear (or convex) programming relaxations that satisfy the conditions of Definition 10. A key parameter of such a program is the length of the longest allowed path $D = \max_{(u,v) \in \mathcal{S}} \max_{p \in \mathcal{P}_{u,v}} \ell(p)$ (where $\ell(p)$ is the length of path p).

4.2 Distributed Algorithm

In order to solve these convex programs in a distributed manner, we will first use padded decompositions to form a local problem using a simple distributed algorithm. Let P be a partition sampled from a (k, ϵ) -padded decomposition (in particular, obtained by Lemma 7), where $0 < \epsilon \leq 1$. Recall that for each cluster $C \in P$, $E(C) = \{(u, v) \in E \mid u, v \in C\}$. We define $G(C)$ to be the subgraph induced by C . We prove the following lemma in Appendix B.

► **Lemma 11.** *For each cluster C sampled from a (k, ϵ) -padded decomposition, there is a distributed algorithm running in $O(\frac{k}{\epsilon} \log n)$ rounds so that every cluster center knows $G(C)$.*

Let $\text{CP}(G)$ be a distance bounded network design CP defined on graph $G = (V, E)$. We will define local convex programs based on a partition P of G that is sampled from a (D, λ) -padded decomposition. The value of $0 < \lambda \leq 1$ will be set later based on the parameters of our distributed algorithm. For each $C \in P$, let $\text{CP}(C)$ be $\text{CP}(G)$ defined on $G(C)$, but where only demands corresponding to any pair $(u, v) \in \mathcal{S}$ in which $B(u, D)$ is fully contained in C are included. We denote the set of these demands by $N(C)$, more precisely, $N(C) = \{(u, v) \in \mathcal{S} \mid B(u, D) \subseteq C\}$. The objective will then be to minimize $g(x) = g((x_e)_{e \in E(C)})$. In other words $\text{CP}(C)$ is defined as follows:

$$\begin{aligned}
\min \quad & g(x) \\
\text{s.t.} \quad & \sum_{P \in \mathcal{P}_{u,v}: e \in P} f_P \leq x_e && \forall (u, v) \in N(C), \forall e \in E(C) \\
& \sum_{P \in \mathcal{P}_{u,v}} f_P \geq 1 && \forall (u, v) \in N(C) \\
& x_e \geq 0 && \forall e \in E(C) \\
& f_P \geq 0 && \forall (u, v) \in N(C), \forall P \in \mathcal{P}_{u,v}
\end{aligned}$$

There is a technical subtlety about computing the function g on each cluster, which is the fact that a solution $\langle x^C, f^C \rangle$ of $\text{CP}(C)$ is only defined on $G(C)$. While in practice x^C is a vector defined only on edges in $E(C)$, in our analysis, we will assume that x^C is a vector

Algorithm 2: Distributed algorithm for approximating distance bounded network design CPs.

- 1 Set $\lambda = \frac{\epsilon(1-\epsilon)}{(2-\epsilon)(1+\epsilon)}$ and $t = \left\lceil \frac{16(1-\frac{\epsilon}{2})(1+\epsilon) \ln n}{\epsilon^2} \right\rceil$.
 - 2 Sample from (D, λ) -padded decompositions t times by Lemma 7, and let P_i be the partition obtained in the i 'th run.
 - 3 For each cluster $C \in P_i$, the center of cluster C computes $G(C)$ (see Lemma 11).
 - 4 The center of each cluster $C \in P_i$ solves $CP(C)$ and sends the solution $\langle x^{C,i}, f^{C,i} \rangle$ to all nodes $u \in C$.
 - 5 **for** $e = (u, v) \in E$ **do**
 - 6 Let $I_{u,v} = \{i \mid \exists C \in P_i : u, v \in C\}$.
 // these are the iterations in which both endpoints are in same cluster
 - 7 $\tilde{x}_e \leftarrow \min(1, \frac{1+\epsilon}{t} \sum_{i \in I_{u,v}} x_e^{C_{u,i},i})$.
-

in $\mathbb{R}^{|E|}$ and $x_e^C = 0$ for all $e \notin E(C)$. This assumption does not impact the correctness of algorithm. The following lemma is similar to Lemma 3.8 in [10], and we show that it holds for our modified definition of local convex programs and for generalized objective functions that satisfy Definition 8.

► **Lemma 12.** *Let $\langle x^*, f^* \rangle$ be an optimal solution of $CP(G)$ and let $x^{*C} = (x_e^*)_{e \in E(C)}$. For each cluster $C \in P$, let $\langle \tilde{x}^C, \tilde{f}^C \rangle$ be an optimal solution of $CP(C)$. Then $g(\tilde{x}^C) \leq g(x^{*C})$.*

Proof. We argue that the vector $\langle x^{*C}, f^{*C} \rangle$, where $x_e^{*C} = x_e^*$ for all $e \in E(C)$ and $f_p^{*C} = f_p^*$ for all $p \in \mathcal{P}_{u,v}$, is a feasible solution to $CP(C)$. By definition of $N(C)$ we have that for any $(u, v) \in N(C)$ all paths in $\mathcal{P}_{u,v}$ also appear in $G(C)$, and therefore $\langle x^{*C}, f^{*C} \rangle$ satisfies both capacity and flow constraints of $CP(C)$ for pairs $(u, v) \in E(C)$ since they were satisfied in $CP(G)$. Since we assumed that $\langle \tilde{x}^C, \tilde{f}^C \rangle$ is an optimal solution of $CP(C)$, we get $g(\tilde{x}^C) \leq g(x^{*C})$. ◀

We now provide in Algorithm 2 a distributed algorithm for solving $CP(G)$. The high level idea is the following: we partition the graph t times, have cluster centers solve $CP(C)$ of their cluster using a sequential algorithm in each iteration, and then take an average over the solutions for each edge. Intuitively, for each edge, by averaging over local solutions for iterations in which the ball around that edge is in the same cluster, with high probability we get a feasible global solution. In the proof of Theorem 13, we will show that this solution gives a $(1 + \epsilon)$ -approximation solution to the LP, for an arbitrary $0 < \epsilon \leq 1$.

We assume that all nodes know the values of D and ϵ . Let $C_{u,i}$ denote the cluster that node u belongs to in the i -th iteration, and let $\langle x^{C_{u,i},i}, f^{C_{u,i},i} \rangle$ be the fractional CP solution of $C_{u,i}$, where $\langle x_e^{C_{u,i},i}, f_p^{C_{u,i},i} \rangle$ is the fractional CP value for $e = (u, v)$, and $p \in \mathcal{P}_{u,v}$. Since the objective is a function of edge vectors, what we mean by having a distributed solution to a distance bounded network design CP is that each node u will know the value x_e for all the edges e incident to u . It is not hard to see that the algorithm could be modified so that every node u can also know the flow value f_p for each path p .

► **Theorem 13.** *Algorithm 2 takes $O((D/\epsilon) \log n)$ rounds to terminate, and it will compute a solution of cost at most $(1 + \epsilon)CP^*$ to a bounded distance network design CP (Definition 10) with high probability, where CP^* is the optimal solution and $0 < \epsilon \leq 1$. Moreover, if the convex program can be solved sequentially in polynomial time, then all of the node computations are also polynomial time.*

Proof. Correctness: We first show that with high probability the values $\tilde{x}_e, e \in E$ form a feasible solution. Here we only need to show that a feasible solution for the flow values exist, and do not require nodes to compute these values. Let $I_u = \{i : \exists C \in P_i, B(u, D) \subseteq C\}$, i.e. I_u is the set of iterations in which $B(u, D)$ is contained in a cluster, and let $I_{u,v}$ be the set of iterations in which both u and v are in the same cluster. Since we need to implement Algorithm 2 in a distributed manner, we use $I_{u,v}$ in our implementation, while the analysis is based on I_u . We can do so since by definition we have $I_u \subseteq I_{u,v}$, for any $(u, v) \in E$.

For any $p \in \mathcal{P}_{u,v}$, we set the flow values to be $\tilde{f}_p = \frac{1}{|I_u|} \sum_{i \in I_u} f_p^{C_{u,i,i}}$, i.e. \tilde{f}_p is the average over local flows in iterations in which $B(u, k)$ is fully contained in a cluster. We will show that this gives a feasible flow. First we argue that enough flow is being sent. For all $(u, v) \in \mathcal{S}$, we have,

$$\sum_{p \in \mathcal{P}_{u,v}} \tilde{f}_p = \sum_{p \in \mathcal{P}_{u,v}} \frac{1}{|I_u|} \sum_{i \in I_u} f_p^{C_{u,i,i}} = \frac{1}{|I_u|} \sum_{i \in I_u} \sum_{p \in \mathcal{P}_{u,v}} f_p^{C_{u,i,i}} \geq \frac{1}{|I_u|} \sum_{i \in I_u} 1 \geq 1.$$

We have used the fact that for each $i \in I_u$ the solution corresponding to the CP of the cluster containing u satisfies the constraint that $\sum_{p \in \mathcal{P}_{u,v}: e \in p} f_p^{C_{u,i,i}} \geq 1$, because for each such i we know that $(u, v) \in N(C)$.

Next, we will argue that the capacity constraints are also satisfied. The second property of (D, λ) -padded decompositions implies that $\Pr(i \in I_u) \geq 1 - \lambda = 1 - \frac{\epsilon(1-\epsilon)}{(2-\epsilon)(1+\epsilon)} = \frac{1}{(1-\frac{\epsilon}{2})(1+\epsilon)}$ for each iteration $1 \leq i \leq t$. By linearity of expectations we have $E[|I_u|] \geq t(1-\lambda)$. Since each sampling is performed independently, by Chernoff bound for $\delta = \epsilon/2$, we get,

$$\begin{aligned} \Pr(|I_u| \leq t(1-\lambda)(1-\delta)) &= \Pr\left(|I_u| \leq \frac{t(1-\frac{\epsilon}{2})}{(1-\frac{\epsilon}{2})(1+\epsilon)}\right) \\ &= \Pr\left(|I_u| \leq \frac{t}{1+\epsilon}\right) \leq e^{-\frac{(\epsilon/2)^2(1-\lambda)t}{2}} \leq e^{-2 \ln n} = \frac{1}{n^2}. \end{aligned}$$

Hence by a union bound on all nodes we have that with high probability $|I_u| > t/(1+\epsilon)$. Therefore, for all $(u, v) \in \mathcal{S}, e \in E$, we have (w.h.p.),

$$\begin{aligned} \sum_{p \in \mathcal{P}_{u,v}: e \in p} \tilde{f}_p &= \sum_{p \in \mathcal{P}_{u,v}: e \in p} \frac{1}{|I_u|} \sum_{i \in I_u} f_p^{C_{u,i,i}} = \frac{1}{|I_u|} \sum_{i \in I_u} \sum_{p \in \mathcal{P}_{u,v}: e \in p} f_p^{C_{u,i,i}} \leq \frac{1}{|I_u|} \sum_{i \in I_u} x_e^{C_{u,i,i}} \\ &\leq \min\left(1, \frac{1}{|I_u|} \sum_{i \in I_{u,v}} x_e^{C_{u,i,i}}\right) \leq \min\left(1, \frac{1+\epsilon}{t} \sum_{i \in I_{u,v}} x_e^{C_{u,i,i}}\right) = \tilde{x}_e. \end{aligned}$$

Upper bound: We will now show that the upper bound holds. Let $\langle x^*, f^* \rangle$ be an optimal solution to CP(G). We have $\tilde{x}_e = \min(1, \frac{1+\epsilon}{t} \sum_{i \in I_e} x_e^{C_{u,i,i}})$, and for each $e = (u, v)$ and $1 \leq i \leq t$, we set $\tilde{x}_e^i = x_e^{C_{u,i,i}}$ if $i \in I_e$, and $\tilde{x}_e^i = 0$ otherwise. Note that $0 < (1+\epsilon)/t < 1$, and since g is a convex function and $g(\vec{0}) = 0$, by Jensen's inequality we have $g(\frac{1+\epsilon}{t}x) \leq \frac{1+\epsilon}{t}g(x)$. Then for $\tilde{x} = (\tilde{x}_e)_{e \in E}$ we can write:

$$\begin{aligned} g(\tilde{x}) &= g((\tilde{x}_e)_{e \in E}) \leq g\left(\frac{1+\epsilon}{t} \left(\sum_{i \in I_e} x_e^{C_{u,i,i}}\right)_{e \in E}\right) \leq \frac{1+\epsilon}{t} g\left(\left(\sum_{i \in I_e} x_e^{C_{u,i,i}}\right)_{e \in E}\right) \\ &\leq \frac{1+\epsilon}{t} g\left(\left(\sum_{i=1}^t \tilde{x}_e^i\right)_{e \in E}\right) \leq \frac{1+\epsilon}{t} g\left(\sum_{i=1}^t (\tilde{x}_e^i)_{e \in E}\right) \leq \frac{1+\epsilon}{t} \sum_{i=1}^t g((\tilde{x}_e^i)_{e \in E}). \end{aligned}$$

In the final inequality, since g is convex, we used Jensen's inequality to take the sum out of the function. It is now enough to show that in each iteration i , it holds $g((\tilde{x}_e^i)_{e \in E}) \leq g(x^*)$.

Let $\tilde{x}^i = ((\tilde{x}_e^i)_{e \in E})$, and let $P_i = \{C_1, C_2, \dots, C_\ell\}$ be the partition of V . Since g is a convex partitionable function w.r.t. G , there exists a nondecreasing and convex function $h : \mathbb{R}^m \mapsto \mathbb{R}$ for which we can write $g(\tilde{x}^i) = h_{P_i}(g(\tilde{x}^{i,C_1}), g(\tilde{x}^{i,C_2}), \dots, g(\tilde{x}^{i,C_\ell}))$, since $\tilde{x}_e^i = 0$ by definition for edges which go between clusters (for simplicity we are denoting \tilde{x}^{i,C_j} by \tilde{x}^{i,C_j}).

Recall that x^{*C} is the vector in which $x_e^{*C} = x_e^*$ for all $e \in E(C)$ and $x_e^{*C} = 0$ otherwise. By Lemma 12 we get that for all $C \in P_i$, $g(\tilde{x}^{i,C}) \leq g(x^{*C})$. Now we consider a vector \hat{x} , defined by setting $\hat{x}_e = x_e^*$ for all edge e with both endpoints in the same cluster, and $\hat{x}_e = 0$ otherwise. Since we assumed h_{P_i} to be nondecreasing, we get,

$$\begin{aligned} g(\tilde{x}^i) &= h_{P_i}(g(\tilde{x}^{i,C_1}), g(\tilde{x}^{i,C_2}), \dots, g(\tilde{x}^{i,C_\ell})) \leq h_{P_i}(g(x^{*C_1}), g(x^{*C_2}), \dots, g(x^{*C_\ell})) \\ &= h_{P_i}(g(\hat{x}^{C_1}, \hat{x}^{C_2}, \dots, \hat{x}^{C_\ell})) = g(\hat{x}) \leq g(x^*). \end{aligned}$$

For the last inequality we have used the fact that g is non-decreasing, and that for all $e \in E$, $\hat{x}_e \leq x_e^*$ (since either $\hat{x}_e = x_e^*$ or $\hat{x}_e = 0$). By plugging this into the above inequalities, we will get $g(\tilde{x}) \leq \frac{1+\epsilon}{t} \sum_{i=1}^t g(\tilde{x}^i) \leq (1+\epsilon)g(x^*)$, which implies the claim that Algorithm 2 gives a $(1+\epsilon)$ -approximation to the optimal solution.

Time Complexity: The decomposition step and sending the information within a cluster takes $O((D/\epsilon) \log n)$ rounds since the diameter of each cluster is $O((D/\lambda) \log n) = O((D/\epsilon) \log n)$. Since each decomposition is independent, we can do all of them in parallel, so steps 1-4 of the algorithm only take $O((D/\epsilon) \log n)$ rounds in total. Clearly the rest of the algorithm can be done in a constant number of rounds. Hence in total w.h.p. the algorithm will take $O((D/\epsilon) \log n)$ rounds. ◀

5 Distributed Approximation Algorithms for Network Design

In this section, we will focus on several network design problems which can be approximated by first solving a convex relaxation using Algorithm 2 and then locally rounding the solution. For that purpose, we will describe how each problem has a distance bounded network design CP relaxation (Definition 10), and will then show that existing rounding schemes are local. All missing proofs can be found in Appendix C.

5.1 Directed k -Spanner

Dinitz and Krauthgamer [9] introduced a linear programming relaxation for DIRECTED k -SPANNER which is just a distance-bounded network design CP with demands pairs $\mathcal{S} = E$, allowed paths $\mathcal{P}_{u,v}$ which are the directed paths from u to v of length at most k , and objective function $g(x) = \sum_{e \in E} x_e$. They showed that this LP can be solved in polynomial time (approximately if k is non-constant). We will denote this LP by $LP(G)$. Clearly, $LP(G)$ is a distance bounded network design CP with $D = k$. Hence, Theorem 13 implies that we can use Algorithm 2 to approximately solve this LP in $O(k \log n)$ rounds in the \mathcal{LOCAL} model.

We now provide in Algorithm 3 a distributed rounding scheme that gives an $O(n^{1/2} \log n)$ -approximation for DIRECTED k -SPANNER. This algorithm matches the best centralized approximation ratio known [5], and is just the obvious distributed version of the algorithm proposed in [5]. The difference is that here we truncate the shortest-path trees at depth k (as opposed to full shortest-path trees), and nodes choose whether to become a tree root independently (rather than chosen without replacement as in [5]).

The following lemma is essentially from [5], with the proof requiring only slight technical changes due to the slightly different algorithms. We sketch it for completeness in Appendix C.

Algorithm 3: Distributed rounding algorithm for k -spanner.

Input : Graph $G = (V, E)$, fractional solution $\langle x, f \rangle$ to $LP(G)$.

- 1 $E' = \emptyset, \forall v \in V : T_v^{in} = \emptyset, T_v^{out} = \emptyset.$
- 2 **for** $e \in E$ **do**
- 3 \lfloor Add e to E' with probability $\min(n^{1/2} \cdot \ln n \cdot x_e, 1)$.
- 4 **for** $v \in V$ **do**
 - 5 // Random tree sampling
 - 6 Choose p uniformly at random from $[0, 1]$.
 - 7 **if** $p < \frac{3 \ln n}{\sqrt{n}}$ **then**
 - 8 $T_v^{in} \leftarrow$ shortest path in-arborescence rooted at v truncated at depth k .
 - 9 $T_v^{out} \leftarrow$ shortest path out-arborescence rooted at v truncated at depth k .
- 9 Output $E' \cup (\cup_{v \in V} (T_v^{in} \cup T_v^{out}))$. // A node knows its portion of the output.

► **Lemma 14.** *Given a directed graph G , $LP(G)$ as defined, and a fractional solution LP^* to $LP(G)$, the output of Algorithm 3 has size $O(n^{1/2} \cdot (n + LP^*) \log n)$.*

It is easy to see that this algorithm can be implemented in the \mathcal{LOCAL} model.

► **Lemma 15.** *Algorithm 3 runs in $O(k)$ time in the \mathcal{LOCAL} model.*

We now immediately get our main result for DIRECTED k -SPANNER. Proof can be found in Appendix C.

► **Corollary 16.** *Algorithm 2 with $D = k$ along with the rounding scheme in Algorithm 3 yields an $O(n^{1/2} \ln n)$ -approximation w.h.p. to DIRECTED k -SPANNER that runs in $O(k \log n)$ time in the \mathcal{LOCAL} model and uses only polynomial-time computations at each node.*

5.2 Basic 3-Spanner and Basic 4-Spanner

If the input graph is undirected then stronger approximations are possible. In particular, for stretch 3 and 4, there are $\tilde{O}(n^{1/3})$ -approximations due to [5] (for stretch 3) and [11] (for stretch 4). Without going into details, both of these algorithms use the same LP relaxation as in DIRECTED k -SPANNER, but round the LP differently. So in order to give distributed versions of these algorithms, we only need to modify Algorithm 3 to use the appropriate rounding algorithm (and change some of the other parameters in the shortest-path arborescence sampling). Fortunately, both of these algorithms use rounding schemes which are highly local. Informally, rather than sample each edge independently with probability proportional to the (inflated) fractional value as in Algorithm 3, these algorithms sample a value independently at each *vertex* and then include an edge if a particular function of the values of the two endpoints (different in each of the algorithms) passes some threshold. Clearly this is a very local rounding algorithm: once we have solved the LP relaxation using Theorem 13, each node can draw its random value and then spend one more round to exchange a message with each of its neighbors to find out their values, and thus determine which of the edges have been included by the rounding. Thus the total running time is dominated by the time needed to solve the LP, which in these cases is $O(\log n)$ using Theorem 13.

5.3 Lowest-Degree k -Spanner

We now turn our attention to LOWEST-DEGREE k -SPANNER: Given a graph $G = (V, E)$ and a value k , we want to find a k -spanner that minimizes the maximum degree. We will use the relaxation and rounding scheme proposed by Chlamtáč and Dinitz [6]. The linear programming relaxation used in [6] is very similar to the Directed and Basic k -spanner LP relaxation described earlier, with the difference being that a new variable λ is added to represent the maximum degree, and so the objective is to minimize λ and constraints are added to force λ to upper bound the maximum fractional degree. The proof of the following result can be found in Appendix C.

► **Theorem 17.** *Given a graph $G = (V, E)$ (directed or undirected), and any integer $k \geq 1$ there is a distributed algorithm that w.h.p. computes an $\tilde{O}(\Delta^{(1-1/k)^2})$ -approximation to the LOWEST-DEGREE k -SPANNER problem, taking $O(k \log n)$ rounds of the LOCAL model and using only polynomial-time computations at each node.*

5.4 Directed Steiner Network with Distance Constraints

It is well-known that the centralized rounding of [5] for DIRECTED k -SPANNER is more general than is actually stated in their paper. In particular, the randomized rounding for “thin” edges gives the same guarantee even when each demand has a possibly different distance constraint. This fact was used, e.g., in [7] in their algorithms for DISTANCE PRESERVER, PAIRWISE k -SPANNER, and DIRECTED STEINER NETWORK WITH DISTANCE CONSTRAINTS. The difficulty in extending the algorithm of [5] is not in the LP rounding, but rather because the arborescence sampling technique used to handle thick edges in [5] (and in our Algorithm 3) assumes that n is a lower bound on the optimal cost. This assumption is true for DIRECTED k -SPANNER, but false for variants where there might be a tiny number of demands. However, it is easy to see that if we assume the demand graph is spanning (i.e., assume that every node is an endpoint of at least one demand) then the optimal solution must have at least $n/2$ edges, and hence we can again just use [5] to get a $\tilde{O}(\sqrt{n})$ -approximation for DIRECTED STEINER NETWORK WITH DISTANCE CONSTRAINTS as long as the demand graph is spanning.

While this is in the centralized setting, since our algorithm for DIRECTED k -SPANNER is just a lightly modified distributed version of [5] (the only difficulty in the distributed setting is solving the LP, which is why that is the main technical contribution of this paper), we can easily modify it to give the same approximation for DIRECTED STEINER NETWORK WITH DISTANCE CONSTRAINTS with spanning demand graphs. The only change is that we use $D = \max_{(u,v) \in \mathcal{S}} L(u, v)$ instead of k when solving the linear programming relaxation (using Theorem 13) and when truncating the shortest-path arborescences that we sample (note that we have to assume that D is global knowledge, which is reasonable for spanner problems and for SHALLOW-LIGHT STEINER NETWORK but may be less reasonable for other special cases of DIRECTED STEINER NETWORK WITH DISTANCE CONSTRAINTS). This implies Theorem 4, and all of the interesting special cases (SHALLOW-LIGHT STEINER NETWORK, DISTANCE PRESERVER, PAIRWISE k -SPANNER, etc.) which it includes.

References

- 1 Amy Babay, Emily Wagner, Michael Dinitz, and Yair Amir. Timely, reliable, and cost-effective internet transport service using dissemination graphs. In *37th IEEE International Conference on Distributed Computing Systems, (ICDCS)*, pages 1–12, 2017.
- 2 Leonid Barenboim, Michael Elkin, and Cyril Gavoille. A fast network-decomposition algorithm and its applications to constant-time distributed computation. *Theoretical Computer Science*, 2016.
- 3 Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *FOCS'96*, pages 184–193, 1996.
- 4 Yair Bartal, John W. Byers, and Danny Raz. Global optimization using local information with applications to flow control. In *FOCS*, pages 303–312, 1997.
- 5 Piotr Berman, Arnab Bhattacharyya, Konstantin Makarychev, Sofya Raskhodnikova, and Grigory Yaroslavtsev. Improved approximation for the directed spanner problem. In *ICALP Part I*, pages 1–12, 2011.
- 6 Eden Chlamtác and Michael Dinitz. Lowest-degree k -spanner: Approximation and hardness. *Theory of Computing*, 12(1):1–29, 2016.
- 7 Eden Chlamtác, Michael Dinitz, Guy Kortsarz, and Bundit Laekhanukit. Approximating spanners and directed steiner forest: Upper and lower bounds. In *SODA*, 2017.
- 8 Michael Dinitz, Guy Kortsarz, and Ran Raz. Label cover instances with large girth and the hardness of approximating basic k -spanner. *ACM Trans. Algorithms*, 12(2):1–16, 2016.
- 9 Michael Dinitz and Robert Krauthgamer. Directed spanners via flow-based linear programs. In *STOC'11*, pages 323–332, 2011.
- 10 Michael Dinitz and Robert Krauthgamer. Fault-tolerant spanners: better and simpler. In *PODC'11*, pages 169–178, 2011.
- 11 Michael Dinitz and Zeyu Zhang. Approximating low-stretch spanners. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, 2016.
- 12 Yevgeniy Dodis and Sanjeev Khanna. Design networks with bounded pairwise distance. In *STOC '99*, pages 750–759, 1999.
- 13 Michael Elkin. Personal Communication, 2017.
- 14 Michael Elkin and Ofer Neiman. Distributed strong diameter network decomposition: Extended abstract. In *PODC '16*, pages 211–216, 2016.
- 15 Patrik Floréen, Marja Hassinen, Joel Kaasinen, Petteri Kaski, Topi Musto, and Jukka Suomela. Local approximability of max-min and min-max linear programs. *Theory of Computing Systems*, 2011.
- 16 R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.
- 17 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988.
- 18 Anupam Gupta, Mohammad T. Hajiaghayi, and Harald Räcke. Oblivious network design. In *SODA '06*, pages 970–979, 2006.
- 19 M. Reza Khani and Mohammad R. Salavatipour. Improved approximations for buy-at-bulk and shallow-light k -steiner trees and $(k,2)$ -subgraph. *Journal of Combinatorial Optimization*, 31(2):669–685, Feb 2016.
- 20 Robert Krauthgamer, James R. Lee, Manor Mendel, and Assaf Naor. Measured descent: A new embedding method for finite metrics. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 434–443, 2004.
- 21 Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The price of being near-sighted. In *SODA '06*, pages 980–989, 2006.
- 22 Nathan Linial and Michael Saks. Low diameter graph decompositions. *Combinatorica*, 13(4):441–454, Dec 1993.

- 23 Christos H. Papadimitriou and Mihalis Yannakakis. Linear programming without the matrix. In *STOC '93*, pages 121–129, 1993.
- 24 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000.
- 25 David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.
- 26 David Peleg and Jeffrey D. Ullman. An optimal synchronizer for the hypercube. In *PODC'87*, pages 77–85, 1987.

A Proofs from Section 3

A.1 Proof of Lemma 6

The first property in Definition 5 is directly implied by the definition of r_v for all nodes $v \in V$. For the second property we consider an arbitrary node $u \in V$, and compute the probability that the ball $B(u, k)$ is not in any of the clusters in P . Consider an arbitrary value $1 \leq t \leq n$, let $v \in V$ be the node such that $t = \pi(v)$, and let $z = z_v$ be the real number sampled by v . Also, for any $x, y \in V$, let $\tilde{d}(x, y) = \min(d(x, y), r \ln n + k)$. Let us also order the clusters based on their center's position in the permutation, so that C_t is the cluster corresponding to $t = \pi(v)$ (i.e. v is the cluster center of C_t). We define X_t to be the event that if $B(u, k)$ is not in the first $t - 1$ clusters, then it is also not in any of the remaining clusters. We provide a recursive bound on X_t based on X_{t+1} . Then we will get the second property once we show $\Pr(X_0) \leq \epsilon$. We need to define the following events:

- A_t : $B(u, k)$ does not intersect with any of the clusters C_1, \dots, C_{t-1} .
- M_t^{cut} : $(\tilde{d}(v, u) - k \leq z < \tilde{d}(v, u) + k \mid A_t)$.
- M_t^{ex} : $(z < \tilde{d}(v, u) - k \mid A_t)$.
- X_t : $(\nexists j \geq t : B(u, k) \subseteq C_j \mid A_t)$.

In other words, conditional on the event that $B(u, k)$ is not in any of the first $t - 1$ clusters, either $B(u, k) \subseteq C_t$, or else one the following two events will occur: M_t^{cut} is the event that $B(u, k)$ partially intersects C_t , and M_t^{ex} is the event that $B(u, k)$ does not intersect C_t .

Now the event X_t occurs only when either M_t^{cut} occurs or both M_t^{ex} and X_{t+1} occur (i.e. when $B(u, k)$ is not in C_t or any of the next clusters). Hence we can write $\Pr(X_t) \leq \Pr(M_t^{cut}) + \Pr(M_t^{ex}) \Pr(X_{t+1})$. Recall that z is independently sampled from the density function $p(z_v) = \binom{n}{n-1} \frac{e^{-z_v/r}}{r}$, and thus M_t^{cut} can be written as follows:

$$\begin{aligned} \Pr(M_t^{cut}) &= \int_{\tilde{d}(v, u) - k}^{\tilde{d}(v, u) + k} p(z) dz = \binom{n}{n-1} \left(1 - e^{-2k/r}\right) e^{-(\tilde{d}(v, u) - k)/r} \\ &\leq \binom{n}{n-1} \frac{2k}{r} e^{-(\tilde{d}(v, u) - k)/r}. \end{aligned}$$

Similarly, we can write,

$$\Pr(M_t^{ex}) = \int_0^{\tilde{d}(v, u) - k} p(z) dz = \binom{n}{n-1} \left(1 - e^{-(\tilde{d}(v, u) - k)/r}\right).$$

We now inductively prove that $\Pr(X_t) \leq (2 - \frac{t}{n-1}) \binom{2k}{r}$. If $t < n$ is the last step, then $\Pr(X_t) = 0$, and thus this bound clearly holds. Assume that the bound is true for X_{t+1} , we

show that then it also holds for X_t . We have,

$$\begin{aligned} \Pr(X_t) &\leq \Pr(M_t^{cut}) + \Pr(M_t^{ex}) \Pr(X_{t+1}) \\ &\leq \binom{n}{n-1} \binom{2k}{r} \left(1 + \frac{n-t-2}{n-1} \left(1 - e^{-(\tilde{d}(v,u)-k)/r}\right)\right). \end{aligned}$$

Since $e^{-(\tilde{d}(v,u)-k)/r} \geq e^{-(\ln n)} \geq 1/n$, we get that $\Pr(X_t) \leq \left(2 - \frac{t}{n-1}\right) \binom{2k}{r}$. The second property is then implied by the fact that $\Pr(X_0) \leq \frac{2k}{r} = \frac{2k}{2k(1/\epsilon)} = \epsilon$.

B Proofs from Section 4

B.1 Proof of Lemma 9

Let $\sigma = \{\sigma_1, \dots, \sigma_\ell\}$ be a partition of nodes in V . For all $1 \leq i \leq \ell$, we have $g(x^{\sigma_i}) = \max_{v \in \sigma_i} (\sum_{u: (v,u) \in E} x_{(v,u)}^{\sigma_i})$. Then we can set $h_\sigma(y) = \max_{i \in [\ell]} (y_i)$, $y \in \mathbb{R}^\ell$, where y_i is the i -th coordinate of y . Let $\sigma(v) \in \sigma$ be the cluster that node v belongs to. For all $x = (x_{(u,v)})_{(u,v) \in E}$, where $x_{(u,v)} = 0$ for any $(u,v) \in E$ s.t. $\sigma(u) \neq \sigma(v)$, we have,

$$\begin{aligned} g(x) &= \max_{v \in V} \left(\sum_{u: (v,u) \in E} x_{(v,u)} \right) = \max_{\sigma_i \in \sigma} \left(\max_{v \in \sigma_i} \left(\sum_{u: (v,u) \in E} x_{(v,u)}^{\sigma_i} \right) \right) \\ &= \max_{\sigma_i \in \sigma} (g(x^{\sigma_i})) = h_\sigma(g(x^{\sigma_1}), g(x^{\sigma_2}), \dots, g(x^{\sigma_\ell})). \end{aligned}$$

It is also easy to see that the function h_σ is convex and non-decreasing. Hence h_σ satisfies the conditions in Definition 8.

B.2 Proof of Lemma 11

The first property of (k, ϵ) -padded decompositions implies that for all nodes $u \in C$, we have $d(u, v) = O((k/\epsilon) \log n)$, where v is the center of cluster C . Each node $u \in C$ that determines v as the center of the cluster it belongs to, will send the information of its incident edges to v . Since there is no bound on the size of the messages being forwarded, this can be done in $O((k/\epsilon) \log n)$ time.

C Proofs from Section 5

C.1 Proof of Lemma 14

Let $N_{s,t}$ be the subgraph of G induced by the nodes on paths in $\mathcal{P}_{s,t}$. Edge $e \in E$ is called a *thick* edge if $|N_{s,t}| \geq n^{1/2}$, and otherwise it is called a *thin* edge. The set E' in Algorithm 3 satisfies the spanner property for all thin edges (as argued in [5]), and the random tree sampling phase satisfies the spanner property for the thick edges. Each thick edge (s, t) is spanned if at least one node in $N_{s,t}$ performs the random tree sampling. This probability is at least $1 - (1 - \frac{3 \ln n}{n^{1/2}})^{n^{1/2}} \geq 1 - 1/n^3$. Then a union bound on all the edges (of size at most $O(n^2)$) implies that w.h.p. all thick edges are spanned. We now argue that the output is an $O(n^{1/2} \log n)$ -approximation algorithm: at most $O(n^{1/2} \log n)$ arborescences are chosen with high probability (each arborescence has $O(n)$ edges), and we argued that $|E'| = O(n^{1/2} \log n \cdot LP^*)$. Hence, the overall size of the output is $O(n^{1/2} \log n \cdot (n + LP^*))$.

C.2 Proof of Lemma 15

Each node v in G has received the fractional solutions x_e corresponding to all edges $e \in E$ incident to v . The randomized rounding step can be performed locally: the node with the smaller ID flips a coin, and exchanges the coin flip result with its corresponding neighbors. In order to form T_i^{in} and T_i^{out} , v performs a distributed BFS algorithms by forming a shortest path tree while keeping track of the distance from v . When the distance counter reaches k , the tree construction terminates.

C.3 Proof of Corollary 16

We first run Algorithm 2 to solve $LP(G)$ up to a constant factor (by setting $\epsilon = 1/2$), which takes time $O(k \log n)$ with high probability (Theorem 13). Since each cluster center can solve the local LP in polynomial time, all computations are polynomial time. We then use Algorithm 3 to round the fractional solutions of $LP(G)$, which takes $O(k)$ time. Since the size of a k -spanner is at least $\Omega(n)$, Algorithm 3 then outputs an $O(n^{1/2} \ln n)$ -approximation to the minimum (Lemma 14).

C.4 Proof of Theorem 17

It is easy to see that the LP relaxation proposed in [6] can be written as a distance bounded network design CP where the objective is $\max_{v \in V} (\deg(v)) = \max_{v \in V} \left(\sum_{u: \{v,u\} \in E} x_{\{v,u\}} \right)$ (we do not need to use their extra variable λ , since we can instead directly write the objective). Lemma 9 implies that this function is convex partitionable w.r.t. G , and hence the LOWEST-DEGREE k -SPANNER problem can be approximately solved (to within a constant factor) by using Algorithm 2 with $\epsilon = 1/2$. Next, we use the following rounding scheme proposed in [6]: each edge $e \in E$ is included in the spanner with probability $x_e^{1/k}$. It is clear that this can be done in a constant number of rounds, and hence the overall algorithm takes $O(k \log n)$ rounds (by Theorem 13) in the \mathcal{LOCAL} model. In [6], it was shown that this leads to a $\tilde{O}(\Delta^{(1-1/k)^2})$ -approximation solution of the problem.