Approximation Algorithms for Optimal Hopsets

Michael Dinitz[*] Ama Koranteng[*] Yasamin Nazari[†]

Abstract

For a given graph G, a *hopset* H with hopbound β and stretch α is a set of edges such that between every pair of vertices u and v, there is a path with at most β hops in $G \cup H$ that approximates the distance between u and v up to a multiplicative stretch of α. Hopsets have found a wide range of applications for distance-based problems in various computational models since the 90s. More recently, there has been significant interest in understanding these fundamental objects from an existential and structural perspective. But all of this work takes a worst-case (or existential) point of view: How many edges do we need to add to satisfy a given hopbound and stretch requirement for *any input graph*?

We initiate the study of the natural optimization variant of this problem: given a *specific graph instance*, what is the minimum number of edges that satisfy the hopbound and stretch requirements? We give approximation algorithms for a generalized hopset problem which, when combined with known existential bounds, lead to different approximation guarantees for various regimes depending on hopbound, stretch, and directed vs. undirected inputs. We complement our upper bounds with a lower bound that implies Label Cover hardness for directed hopsets and shortcut sets with hopbound at least 3.

1 Introduction

A hopset H with hopbound β and stretch α for a given (directed or undirected) graph G is a set of (possibly weighted) edges such that between every pair of vertices u and v in G there is a path with at most β hops in $G \cup H$ that approximates the distance between u and v up to multiplicative stretch α (our results hold for a more generalized version of the problem formally defined in Definition 2). A related object is shortcut sets, which preserve *reachability* via low-hop paths, rather than distances. Hopsets were formally introduced 25 years ago by Cohen [Coh00], and they were used to compute approximate single-source shortest paths in undirected graphs in parallel settings. More recently, they have been shown to be useful for a variety of different problems and settings, and have been studied extensively. Examples of these applications include low parallel depth single-source shortest paths algorithms [KS97, Coh00, MPVX15, EN19b], distributed shortest-paths computation [EN19a, EN18, CHDKL21], and dynamic shortest-paths [BR11, HKN14, GWN20, Che18, ŁN22] with implications for fast static flow-based algorithms [Mad10, BPGS21, CKL$^+$22] and dynamic clustering [CFG$^+$24], faster construction of related objects such as distributed or massively parallel distance sketches [EN18, DN19, DM24], parallel reachability [UY90], and work-efficient algorithms for reachability and directed shortest paths [CFR20a, CFR20b, CF23, Fin18, JLS19].

In addition to their wide range applications, hopsets and shortcut sets are also studied as fundamental objects in their own right. There has been a surge of recent work that, rather than focusing on running time, focuses on finding the best *existential* (extremal) upper or lower bounds for hopsets and shortcut sets [KP22b, BW23, Hes03, HP21, KP23, WXX24]. Namely, the main goal is to find the smallest value $\gamma(n, \beta, \alpha)$ such that *every* graph G on n nodes admits a hopset with hopbound β and stretch α.

Another line of work has explored the connections between hopsets and many other fundamental objects such as spanners [BLP20, ABP18], emulators [EN20, HP19], and distance preservers [KP22a, HXX25, BHT23].

[*]Johns Hopkins University. Supported in part by NSF award 2228995.

[†]Vrije Universiteit Amsterdam. This publication is part of the project VI.Veni.232.038 of the research programme "Dynamic graph algorithms: distances and clustering" which is financed by the Dutch Research Council (NWO).

The two main existing lines of work in this direction are on finding efficient algorithms for constructing hopsets in various computational models for specific applications, or on existential bounds and structural properties. Such results are very useful, since they give us a *worst-case bound* on how many edges we must add to an arbitrary graph if we want a hopset of a certain quality and how fast can this be done for our particular application. The focus in these results has been on improving these existential bounds and developing fast algorithms for a variety of different settings and parameters; see Section 1.2 for a discussion of known results of this form.

A complementary type of problem with a very different flavor is *optimization*: *given* a graph G, hopbound β, and stretch bound α, can we design an algorithm to efficiently find the smallest hopset *for* G (rather than in the worst case over all graphs)? If the minimization problem is NP-hard, then can we *approximate* the smallest hopset for G? Note that the existential and optimization versions of these problems are complementary: good existential bounds guarantee that no matter the graph G, there will be a reasonably small hopset; conversely, good optimization results guarantee that we can find an approximately minimum hopset, even if the best hopset for G is significantly smaller than for the worst-case graph. An existential bound might convince someone to use a hopset for some application—since they know they will never need to add too many edges—but once they commit to using a hopset for that application, they might naturally want to find the best hopset for their particular input. Additionally, in many distributed and parallel settings, adding hopset edges can be seen as a preprocessing step (that can take considerable time), after which (approximate) distance queries can be performed in fewer distributed/parallel rounds (often corresponding to the hopbound). Thus in such cases, we may be willing to spend more preprocessing time in order to add the fewest number of edges.

This natural complementarity between the existential and optimization versions has led to significant study of *both* versions for a number of related objects, most notably graph spanners (subgraphs that approximately preserve distances). For spanners, while the vast majority of work has been on the existential questions (see, e.g., [ADD+93] for the fundamental tradeoff between stretch and size), there has also been significant work on the optimization versions (see, e.g., [KP94, KP98, Kor01, EP07, DK11, BBM+11, DKR16, DNZ20, GKL23, CD16, CDKL20, GLQ21, CDK12, CDR19]). Similarly, there has also been work on optimization versions of other related objects such as reachability preservers [AB24], 2-hop covers [CHKZ03], and diameter reduction [DZ10].

Despite both the recent importance of hopsets and the extensive study of optimization for related objects, the optimization version of hopsets has not yet been considered. In this paper we initiate this line of research, introducing these optimization variants and proving both upper and lower bounds on their approximability.

1.1 Our Results and Techniques

There are many variants of this problem, divided along three main axes: whether the graph is directed or undirected, what the desired hopbound β is, and what the required stretch is (and in particular whether it is arbitrary or whether it is in some particularly nice regime like $1+\epsilon$ for small ϵ or $2k-1$ for integer k). A summary of our results for these variants can be found in Table 1.

1.1.1 Upper Bounds

While this list may appear a bit complex, it turns out that almost all of the results are generated by trading off several approximation algorithms that we design and analyze with the known existential bounds for the given regime. In particular, we consider three main approximation algorithms: one based on rounding an LP relaxation, one based on randomly sampling stars, and one based on defining and analyzing an appropriate variant of junction trees [GKL23, GKL24b, CDKL20]. Instead of just analyzing the approximation ratio of each algorithm as a function of n (the traditional point of view), we analyze the approximation ratios as functions of n, OPT, β and the *local neighborhood size* [DK11, BBM+11]. Importantly, none of these algorithms have performance that depends on whether the graph is undirected or directed, or what the allowed stretch is.

Undirected/Directed	**Hopbound β**	**Stretch**	**Approximation**	**Theorem**
Directed	$\widetilde{O}(n^{2/5})$	Individual	$\widetilde{O}(\beta^{1/3} \cdot n^{2/3+\epsilon'})$	5.19
Directed	$\widetilde{\Omega}(n^{2/5})$	Individual	$\widetilde{O}(n^{1+\epsilon'}/\sqrt{\beta}\,)$	5.21
Directed	$\geq 20 \log n$	$1+\epsilon$	$\widetilde{O}(n^{3/4+\epsilon'} \cdot \epsilon^{-\frac{1}{4}})$	5.24
Directed	2	Individual	$O(\ln n)$	4.3
Undirected	$\widetilde{O}(n^{\frac{1}{2}-\frac{1}{2\eta}})$	$1+\epsilon$	$\widetilde{O}(\sqrt{\beta} \cdot n^{\frac{1}{2}+\frac{1}{2\eta}+\epsilon'})$	5.27
Undirected	$\widetilde{O}(k^{-1/2} \cdot n^{\frac{1}{2}-\frac{1}{2k}})$	$2k-1$	$\widetilde{O}(\sqrt{k\beta} \cdot n^{\frac{1}{2}+\frac{1}{2k}+\epsilon'})$	5.30

Table 1: Comparison of hopset results, where $\epsilon' > 0$ is an arbitrarily small constant, $\epsilon \in (0,1)$, and $\eta > \beta^{1/(\ln\ln\beta - \frac{1}{2}\ln\ln\ln\beta)}$. For $\beta \geq 3$, $\eta \geq 6$. "Individual" stretch means arbitrary distance bounds, possibly different for each individual demand pair. All results are for pairwise demands, with edge lengths being nonnegative integer and upper bounded by $\texttt{poly}(n)$.

Demands	**Edge Lengths**	**Edge Costs?**	**Stretch**	**Approximation**	**Paper**
Pairwise	Arbitrary	Y	2	$O(\ln n)$	[DK11]
All-Pairs	Arbitrary	N	Arbitrary	$\widetilde{O}(n^{1/2})$	[BBM+11]
Pairwise	Integer, $\leq \texttt{poly}(n)$	Y	Individual	$\widetilde{O}(n^{4/5+\epsilon})$	[GKL23]
Pairwise	Vector	Y	Indiv. Vectors	$\widetilde{O}(\lvert D\rvert^{1/2+\epsilon} \cdot \tau^{m-1})$	[GKL24b]

Table 2: Comparison of related results for directed spanners. The result of [GKL24b] allows for *edge resource consumption vectors* as opposed to edge lengths. D is the set of demands, $\epsilon > 0$ is an arbitrarily small constant, τ is the largest resource budget, and m is the size of the resource consumption vectors. "Individual" stretch means arbitrary distance bounds, possibly different for each individual demand pair. All variants require nonnegative edge lengths.

To get the results in Table 1, we trade these three algorithms off not only with each other based on the described parameters, but also with the known existential bounds. Unsurprisingly, there are different existential bounds known for each setting. These different existential bounds are what lead to each of the different rows of Table 1.

We also note that there is a simple reduction from hopsets to the multicriteria spanner problem, introduced by [GKL24b]. For multicriteria spanners, instead of edges being associated with just edge lengths—as with traditional spanners—edges in the multicriteria spanner problem each have a resource consumption vector, where each entry corresponds to the consumption of the respective resource on that edge. The simple reduction to multicriteria spanners implies a $\widetilde{O}(\lvert D\rvert^{1/2+\epsilon} \cdot \beta)$-approximation for hopsets, where D is the set of vertex demand pairs given in the input (see Table 2).

Technical Overview. Technical components of our main approximation algorithms have appeared before, in particular, in the literature on approximation algorithms for spanners. The LP rounding algorithm we use is a variant of the one introduced for spanners by [BBM+11], our star sampling is a variant of the arborescence sampling used by [DK11], and our junction tree algorithm is a variant of the junction trees used by [CDKL20] and [GKL23]. But using them for hopsets involves overcoming a number of technical difficulties.

First, and most notably, the performance bound on arborescence sampling used by [DK11, BBM+11] fundamentally depends on the fact that for traditional graph spanners, the required connectivity implies that $\mathsf{OPT} \geq n-1$, and hence polynomial size bounds of the form n^{γ} can be interpreted as $n^{\gamma-1} \cdot \mathsf{OPT}$, i.e., as $n^{\gamma-1}$-approximations. But for hopsets, since we are adding edges rather than removing them, this is not the case: the only bound we have is the trivial bound of $\mathsf{OPT} \geq 1$ (or else we are already done). Hence bounds that are sublinear for spanners (e.g., the $O(n^{1/2})$-approximation of [BBM+11]) turn into *superlinear* approximation ratios if translated to hopsets in the obvious way. These are still nontrivial bounds, in the sense that the trivial bound for hopsets is an $O(n^2)$-approximation ($\mathsf{OPT} \geq 1$ and we can just add all pairwise edges to get a solution of size $O(n^2)$), but superlinear approximation ratios are not particularly satisfying. This is why we need different algorithms for various regimes of OPT. By combining the star sampling and randomized LP rounding we get an approximation bound that is comparatively better for cases where β is small and OPT is large. On the other hand, our junction tree algorithm performs better when OPT is smaller.

Second, while the LP relaxation that we use for our rounding algorithm is a natural variant of the standard spanner LP relaxation (see [DK11, BBM+11, DNZ20]), the hopset variant turns out to be significantly more difficult to solve. Most notably, solving the equivalent LP relaxations for spanners (either the flow version of [DK11] or the fractional cut version of [DNZ20]) requires using the ellipsoid method with a separation oracle for a problem known as Restricted Shortest Path, in which we are given two distance metrics (think of one metric as being our original edge lengths and one metric as being the fractional values given by the LP to edges) and are asked to find the shortest path in the second metric subject to a distance constraint in the first metric. Unfortunately, Restricted Shortest Path is NP-hard, but it turns out that one can use approximation algorithms for Restricted Shortest Path to solve the LP up to any desired accuracy (see [DK11]). But for our hopset LP, the equivalent separation oracle has *three* metrics (the original edge lengths, the number of hops, and the LP values), and we need to find the shortest path in the third metric subject to upper bounds in the first two metrics. So we design a PTAS for this more complex problem in order to solve our LP.

Third, for similar reasons our junction tree argument is more complex. Junction trees were originally introduced for network design problems where the only constraints are on connectivity, not distances; see, e.g., [CEGS11, FKN12]. By reducing to a "layered" graph, Chlamtáč et al. [CDKL20] showed that junction tree-based schemes are also useful in distance-constrained settings, and this layering technique was pushed significantly further by [GKL23]. When trying to use these ideas for hopsets, though, we have the same difficulty as when solving the LP: we have essentially *two* distance constraints for each demand, corresponding to the number of hops and the allowed distance. To overcome this, we give a "two-stage" layering reduction, where we first construct a layered graph that allows us to get rid of the hop requirement without changing the distance requirements. Then we can use the further layered graph of [GKL23] to get rid of the distance requirement, transforming it into a pure connectivity problem. In fact, we can use [GKL23] as a black-box for this second step.

The black-box nature of our use of [GKL23] has an interesting corollary: even without caring about junction trees, the layered graph reduction we design will immediately let us use the known results for pairwise spanners [GKL23] (by considering them on the *weighted transitive closure*) to get a non-trivial approximation guarantee for our full problem. However this reduction introduces additional factors in β, which we improve by white-boxing their approach, which will in turn give us a better trade-off in combination with the other algorithms.

1.1.2 Lower Bounds

To complement our upper bounds, we show that (at least for directed graphs) we cannot hope to get approximation ratios that are subpolynomial. In particular, we give an approximation-preserving reduction from the famous Label Cover problem (or more precisely, its minimization variant Min-Rep [Kor01]) to the problem of computing the smallest hopset in a directed graph with hopbound at least 3, for any stretch value. This gives the following theorem.

Theorem 1.1. *Assuming that $NP \not\subseteq DTIME(2^{polylog(n)})$, for any constant $\epsilon > 0$, and for any $\beta \geq 3$, there is no polynomial-time algorithm that can approximate directed* Generalized β-Hopset *(for any stretch value; see Definition 2) or the minimum shortcut set on directed graphs with approximation ratio better than $2^{\log^{1-\epsilon} n}$.*

Our reduction is quite similar to known hardness reductions used for spanners [Kor01, EP07, DKR16, CD16]. The main difficulty is that these previous reductions, since they are for spanners, only have to reason about *subgraphs* of the graph created by the reduction. That is, given an instance of Min-Rep, they create some graph G and argue that if the Min-REP optimum is large then any subgraph of G that is a spanner must be large. But for hopsets, not only are the edges of G "free," we also need to argue about hopset edges that *are not* part of G. This requires some changes in the reductions and analysis. The full reduction and details can be found in Section 6.

1.2 Related Work

In earlier applications, sparse $(1+\epsilon)$-approximate hopsets for undirected graphs were used to obtain low parallel depth single-source shortest paths algorithms [KS97, Coh00, MPVX15, EN19b]. Similarly, fast hopset constructions for undirected graphs were studied in many other settings such as distributed [EN19a, EN18, CHDKL21, EN18, DN19, DM24] and dynamic settings [BR11, HKN14, GWN20, Che18, ŁN22].

Another line of work focuses on fast computation of hopsets for directed graphs and shortcut sets that preserve pairwise reachability, rather than distances, while reducing the diameter. These have gained attention particularly due to their application in parallel reachability and directed shortest path computation [UY90, CFR20a, CFR20b, CF23, Fin18, JLS19].

More recently, hopsets and shortcut sets have been studied from an extremal (or existential) point of view. In particular, for $(1+\epsilon)$-hopsets, there are almost (up to $1/\epsilon$ factors) matching upper bounds [EN19a, HP19] and lower bounds [ABP18] in *undirected graphs.* One trade-off that is widely used in $(1+\epsilon)$-approximate single-source shortest paths applications is that for any graph, there exists a $(1+\epsilon)$-hopset of size $n^{1+o(1)}$ with hopbound $n^{o(1)}$. On the other hand, [ABP18] showed that there are instances in which this trade-off is tight. For larger stretch, better size/hopbound trade-offs are known.

In directed graphs there are polynomial gaps between existential lower bounds and upper bounds, both for approximate hopsets and shortcut sets. A widely used folklore sampling approach implies an $\widetilde{O}(n)$ size for exact hopset and shortcut sets, with hopbound $O(\sqrt{n})$ [1]. A breakthrough result of [KP22b] went beyond this long-standing bound by constructing shortcut sets of size $\widetilde{O}(n)$ with hopbound $\widetilde{O}(n^{1/3})$. Later, [BW23] showed that the same upper bound also holds for directed $(1+\epsilon)$-hopsets (up to log factors in weights and aspect ratio). More on these existential bounds can be found in Section 5.3, where we trade them off with our approximation algorithms. Another line of work focused on obtaining subsequently better existential lower bounds for directed graphs [Hes03, HP21, KP23, WXX24]. The current best known lower bound for a linear size shortcut set is $\widetilde{\Omega}(n^{1/4})$ [BH23, WXX24].

Another recent result [BH23] showed that for *exact hopsets* both in directed and undirected *weighted* graphs, the folklore sampling (see Section 5.3) is existentially optimal, implying that the separation between approximate hopsets for directed and undirected graphs, does not hold for exact hopsets.

On the approximation algorithms side, most relevant to hopsets are the weighted pairwise spanner algorithms [GKL23] (see Table 2), 2-hop covers [CHKZ03], and multicriteria spanners [GKL24b] (Table 2). In particular, weighted pairwise spanners (like hopsets and unlike k-spanners) do not have n as a trivial lower bound on OPT, and hence some of the difficulties encountered when designing algorithms are similar. Additionally, when we do not have distance bounds—that is, when stretch is ∞ for all demand pairs—weighted pairwise spanners captures the hopset problem.

Approximation algorithms for 2-hop covers [CHKZ03], hub labelings [CHKZ03, BGGN16], and 2-spanners [KP94, DK11] in particular are closely related to our hopbound 2 regime (see Section 4). In all of these problems, the requirement of covering using 2-paths leads to algorithms that are essentially solving some variant of Set Cover. Our situation is similar, so we can use similar techniques, but our Set Cover variant is slightly different from these other problems, most notably because we do not "pay" for edges that already exist in the graph.

[1]The upperbounds for directed hopsets and shortcut sets give a smooth trade-off between size and hopbound. For simplicity, we discuss the important linear-size regime.

Multicriteria spanners were recently introduced by [GKL24b]; they also rely on a modification of the junction tree framework to give an approximation algorithm. While their junction tree framework works for our setting, we present a different framework tailored to hopsets that yields better approximations for our setting than their framework gives. Other related problems are optimizations for other variants of spanners, including traditional directed k-spanner (unit edge costs, all-pairs demands) [DK11,BBM$^+$11] (Table 2), buy-at-bulk spanners [GKL24a], and transitive closure spanners [BGJ$^+$08, CJMN25]. In addition to transitive closure spanners, [CJMN25] shows hardness of bicriteria approximation for shortcut sets (where they allow the hopbound to be violated).

2 Preliminaries

Going forward, we will generally operate on what we call the *weighted transitive closure* of G, defined as follows. Let $d_G(s,t)$ denote the distance in G from s to t.

Definition 1 (Weighted Transitive Closure)**.** The weighted transitive closure of a graph G, denoted by $G_M = (V, E_M)$, is the weighted graph obtained by first taking the transitive closure of G, then assigning weight $d_G(u,v)$ to each edge (u,v) in the transitive closure. We use $\widetilde{E} = E_M \setminus E$ to denote the set of edges in the weighted transitive closure that are not provided in the input G.

Formally, we consider the following problem:

Definition 2 (Generalized β-Hopset)**.** Given a directed graph $G = (V, E)$, edge weights (or "lengths") $\ell : E \to \{1, 2, 3, ..., \texttt{poly}(n)\}$, a set of vertex pairs $\mathcal{D} \subseteq V \times V$, and a distance bound function $Dist : \mathcal{D} \to \mathbb{N}_{\geq 0}$, find the smallest set of edges $H \subseteq \widetilde{E}$ such that for each vertex pair $(s,t) \in \mathcal{D}$, it is the case that $d_G(s,t) \leq d^{(\beta)}_{H \cup G}(s,t) \leq Dist(s,t)$. In other words, there must be an $s-t$ path P in $H \cup G$ such that $|P| \leq \beta$ and $\sum_{e \in P} \ell(e) \leq Dist(s,t)$.

If G is directed, we say that the problem of interest is *directed* Generalized β-Hopset; otherwise it is *undirected* Generalized β-Hopset. Additionally, if for all $(s,t) \in \mathcal{D}$ we have that $Dist(s,t) = k \cdot d_G(s,t)$ for some k, then this is the *stretch-k* Generalized β-Hopset problem. Note that Generalized β-Hopset is a generalized version of the traditional hopset problem: all of our results hold for *any* set of vertex demand pairs, and many of our results hold for arbitrary distance bound functions.

We will use OPT to refer to the cost of the optimal solution to the Generalized β-Hopset problem instance. That is, OPT will refer to the number of edges in the optimal hopset. A path is an *i-hop path* if there are exactly i edges on the path. We also say that a demand $(s,t) \in \mathcal{D}$ is *settled* or *satisfied* by a graph G (or an edge set E) if there is a path from s to t in G (in E) with at most β hops and distance at most $Dist(s,t)$. Otherwise, demand (s,t) is considered *unsettled* or *unsatisfied.*

Note that the weighted transitive closure of a graph can be found in polynomial-time. When working in the weighted transitive closure, we will generally assign costs to the edges in G_M: edges in E will have cost 0, while edges in $\widetilde{E}$ will have cost 1. It is easy to see that Generalized β-Hopset on G is equivalent to finding a min-cost subgraph of G_M that settles all demand pairs. We will use $c(F)$ to denote the *cost* of an edge set F; namely, $c(F) = |F \cap \widetilde{E}|$ is the number of edges in F not provided in the input. Finally, we will assume each edge $(u,v) \in E$ in the input is a shortest path from u to v in G.

3 LP Relaxation

In this section, we state and solve a cut-covering linear program (LP) for Generalized β-Hopset. A few of our approximation algorithms will randomly round the solution to this LP as a subroutine.

Let $\mathcal{P}_{s,t}$ be the set of paths from s to t that have at most β hops and path length at most $Dist(s,t)$. We will refer to these paths as "allowed" or "valid" paths. The natural flow LP for our problem requires building enough capacity to send one unit of (non-interfering) flow along valid paths for each demand; this is the basic

LP used in essentially all network design problems, and was introduced for spanners by [DK11]. An equivalent LP, which is what we will use for Generalized β-Hopset, is obtained through the duality between flows and fractional cuts (of valid paths). This version of the LP for spanners was studied by [DNZ20], and strengthens the anti-spanner LP of [BBM$^+$11].

In more detail, for a graph $G = (V, E)$, we say that an "$s - t$ cut against valid paths" is a set of edges F such that in the graph $G \setminus F$, there are no valid paths from s to t. In the cut covering LP we will use, the constraints will ensure that any feasible solution must "cover" every cut against valid paths; that is, any feasible solution must buy an edge in each of these cuts. This leads to our LP relaxation, which we call LP(Hopset). The LP has a variable x_e for every edge $e \in E_M$. Note that because edges in E—edges from the input graph—do not contribute to the cost of the solution, we can assume without loss of generality that $x_e = 1$ for all $e \in E$. Let $\mathcal{Z}_{s,t} = \{\mathbf{z} \in [0,1]^{|E|} : \forall P \in \mathcal{P}_{s,t}\ ,\ \sum_{e\in P} z_e \geq 1\}$ be the set of vectors representing all fractional cuts against valid $s - t$ paths. For each demand, our LP requires any feasible integral solution to buy at least one edge from every cut against valid paths.

$$\boxed{\begin{array}{lll} \min & \displaystyle\sum_{e\in\widetilde{E}} x_e & \\ \text{s.t.} & \displaystyle\sum_{e\in E_M} z_e x_e \geq 1 & \forall (s,t) \in \mathcal{D},\ \forall \mathbf{z} \in \mathcal{Z}_{s,t} \\ & x_e \geq 0 & \forall e \in E_M \end{array}} \qquad \text{(LP(Hopset))}$$

The two main differences between the k-spanner cut-covering LP and ours are 1) for spanners, valid paths are those that satisfy the stretch constraint, whereas for hopsets, we want paths that satisfy both distance and hop constraints, and 2) rather than a subgraph of G, we are interested in a subgraph of the *weighted transitive closure* of $G = (V, E)$ (see Definition 1).

As written, LP(Hopset) has an infinite number of constraints. Consider the polytope $\mathcal{Z}_{s,t}$ for some demand (s, t). Due to convexity, we only need to keep the constraints that correspond to the vectors $\mathbf{z}$ that form the vertices of of $\mathcal{Z}_{s,t}$. Since there are only an exponential number of these constraints, we can assume the LP has exponential size. We now quickly prove that the LP is a valid LP relaxation of Generalized β-Hopset.

Lemma 3.1. *Let H be a feasible solution to* Generalized β-Hopset. *For every edge $e \in E_M$, let $x_e = 1$ if $e \in H$ or $e \in E$. Otherwise let $x_e = 0$. Then, $\boldsymbol{x}$ is a feasible integral solution to LP(Hopset).*

Proof. Clearly, $x_e \geq 0$ for all $e \in E_M$, so it is left to show that the cut covering constraints are satisfied. For some demand $(s, t) \in \mathcal{D}$, consider some edge cut against valid $s - t$ paths, C. There is a vector $\mathbf{z} \in \mathcal{Z}_{s,t}$ that serves as the indicator vector for cut C—specifically, there is a vector $\mathbf{z} \in \mathcal{Z}_{s,t}$ such that for every edge $e \in C$, we have $z_e = 1$, and $z_e = 0$ otherwise.

Since H is a feasible solution to Generalized β-Hopset, there must be some edge $e \in C \cap H$ (otherwise, there is no valid path from s to t in $H \cup E$, making H infeasible). This also means $x_e = 1$ and $z_e = 1$. Hence we have that for $\mathbf{z}$, the constraint $\sum_{e\in E_M} z_e x_e \geq 1$ is satisfied. □

Lemma 3.2. *Let $\mathbf{x}$ be a feasible integral solution to LP(Hopset), and let $H = \{e \mid x_e = 1, e \notin E\}$. Then H is a feasible solution to* Generalized β-Hopset.

Proof. Suppose for contradiction there is some demand, $(s, t) \in \mathcal{D}$, that is not satisfied by H. Then there is some cut C against valid $s - t$ paths such that $H \cap C = \emptyset$ (and $x_e = 0$ for all $e \in C$). Since C is a cut against valid $s-t$ paths, there is an indicator vector $\mathbf{z} \in \mathcal{Z}_{s,t}$ such that for every edge $e \in C$, we have $z_e = 1$ ($z_e = 0$ otherwise). Therefore we have that $\sum_{e\in E_M} z_e x_e = 0 < 1$, violating the constraint in LP(Hopset), and thus the assumption that $\mathbf{x}$ is feasible. Hence H must be feasible. □

3.1 Solving the LP

Some of our algorithms for GENERALIZED β-HOPSET involve some flavor of random LP rounding, so in this section we will (approximately) solve LP(Hopset), our LP of interest. The LP has an exponential number of constraints, so we must solve it using the ellipsoid algorithm with a separation oracle. To do so, we start by defining another LP, LP(Oracle 1), that captures the problem of finding a violated constraint of LP(Hopset). To solve LP(Oracle 1), we find yet another (approximate) separation oracle. This second oracle boils down to solving a hopbounded version of the Restricted Shortest Path problem. We show that this problem admits an FPTAS, and that this ultimately translates to a $(1+\epsilon)$-approximation for LP(Hopset). More formally, we will prove the following:

Theorem 3.3. *There is a $(1+\epsilon)$-approximation to the optimal solution of LP(Hopset) for any arbitrarily small constant $\epsilon > 0$.*

3.1.1 Oracle 1

As noted, LP(Hopset) has exponential size, so we find a separation oracle in order to run ellipsoid. To do so, we must determine if there is a fractional cut $\mathbf{z}$ that is not covered by the LP solution $\mathbf{x}$—that is, given $\mathbf{x}$ (satisfying the non-negativity constraints), we want to find $\mathbf{z} \in \mathcal{Z}_{s,t}$, for some demand $(s,t) \in \mathcal{D}$, such that $\sum_{e\in E_M} z_e x_e < 1$. We can find such a violated cut-covering constraint by solving the following LP for each demand $(s,t) \in \mathcal{D}$.

$$\boxed{\begin{array}{lll} \min & \displaystyle\sum_{e\in E_M} z_e x_e & \\ \text{s.t.} & \displaystyle\sum_{e\in P} z_e \geq 1 & \forall P \in \mathcal{P}_{s,t} \\ & z_e \geq 0 & \forall e \in E_M \end{array}} \tag{LP(Oracle 1)}$$

3.1.2 Oracle 2: Hopbounded Restricted Shortest Path Problem

LP(Oracle 1) is also exponential in the number of constraints, so we use the ellipsoid algorithm again, with yet another separation oracle. Given an LP solution $\mathbf{z}$, we must determine whether there is some valid $s-t$ path that is not fractionally cut (or, covered) by $\mathbf{z}$. Specifically, we need to find a violated constraint of the form $\sum_{e\in P} z_e < 1$ for some path $P \in \mathcal{P}_{s,t}$. Observe that we now have three separate metrics: the $\mathbf{z}$ metric (given by a solution to LP(Oracle 1)), our original distance metric from the input, and a hop metric (where each edge has hop "length" 1). We only care about valid $s-t$ paths; namely, paths with length at most $D(s,t)$ in our original distance metric and with length at most β in our hop metric. Thus, our goal is to find the shortest path in the $\mathbf{z}$ metric that respects these upper bounds in the distance and hop metrics.

When there are only two metrics—that is, when the goal is to find the shortest path in one metric subject to an upper bound in the other—this is the Restricted Shortest Path problem, defined as follows.

Definition 3 (Restricted Shortest Path Problem)**.** Let $G = (V, E)$ be a graph such that each edge $e \in E$ is associated with a cost z_e and a delay ℓ_e. Let T be a positive integer, and $s, t \in V$ be the source and target nodes, respectively. The Restricted Shortest Path problem is to find a path P from s to t such that the delay along this path is at most T, and the cost of P is minimal.

The Restricted Shortest Path problem is well studied [Has92,War87,Phi93,XZTT08,HK18]. The problem admits an FPTAS, meaning there exists a polynomial-time $(1+\epsilon)$-approximation for the problem [LR01]. Since our problem of interest has a third metric (the hop metric), we refer to it as the *Hopbounded* Restricted Shortest Path problem. We will modify the Restricted Shortest Path FPTAS algorithm of [LR01] to give an FPTAS for the Hopbounded Restricted Shortest Path problem, thus giving a $(1+\epsilon)$-approximate separation oracle for LP(Oracle 1). We formally define the Hopbounded Restricted Shortest Path problem with respect to LP(Oracle 1):

Definition 4 (Hopbounded Restricted Shortest Path problem)**.** Let $G_M = (V, E_M)$ be a graph such that each edge $e \in E$ is associated with a cost z_e and a delay, or length, ℓ_e. Let $T = D(s,t)$, and $s, t \in V$ be the source and target nodes, respectively. The Hopbounded Restricted Shortest Path problem is to find a path P from s to t with at most β hops, such that the length along this path is at most T, and the cost of P is minimal.

3.1.3 Hopbounded Restricted Shortest Paths Algorithm

At a high level, the Restricted Shortest Path algorithm works as follows. The algorithm runs multiple binary searches to find good upper and lower bounds on the cost of the optimal solution, then uses these bounds to scale and discretize (or "bucket") the costs of the edges. They then give a pseudo-polynomial time dynamic programming algorithm on the problem with bucketed edge costs, which they show is a $(1+\epsilon)$-approximation for the original problem. For runtime, the pseudo-polynomial time DP algorithm is polynomial in U/L (and in $1/\epsilon$), where U and L are the upper and lower bounds, respectively, on the optimal solution. The bounds they find are such that $U/L = O(n)$, and so the algorithm is an FPTAS.

To get this algorithm to work for the Hopbounded Restricted Shortest Path problem, we simply modify the DP algorithm to take our hop metric into account, adding a factor β to the runtime (see Algorithm 1). Like the Restricted Shortest Path DP algorithm, our DP algorithm takes in as input valid upper and lower bounds, U and L, respectively. We also scale the edge costs in the same way, and refer to these scaled costs as "pseudo-costs." Let $D(v,i,j)$ indicate the minimum length (in the original distance metric) of a path from vertices s to v with pseudo-cost at most i and at most j hops. Our algorithm differs from the Restricted Shortest Path algorithm in that we add an additional dimension for hops to the dynamic programming algorithm.

Using arguments from [LR01], we will show that Algorithm 1 is a $(1+\epsilon)$-approximation of the Hopbounded Restricted Shortest Path problem. The following is directly from [LR01] and also holds for our slightly modified algorithm. Let $z(P)$ denote the cost (in the $\mathbf{z}$ metric) of a path P in G_M, and let z^* denote the cost of the optimal path for the original Hopbounded Restricted Shortest Path instance.

Lemma 3.4 (Lemma 3 of [LR01])**.** *If $U \geq z^*$, then Algorithm 1 returns a feasible path, P', that satisfies $z(P') \leq z^* + L\epsilon$*

[LR01] uses this Lemma, and others, to show that there is a $(1+\epsilon)$-approximation for the Restricted Shortest Path problem (Theorems 3 and 4 of [LR01]). Their algorithm achieves a runtime of $O(mn/\epsilon + mn \log\log \frac{U}{L})$. While their full argument would also give a similar runtime for our setting (with an additional factor β), we only include the arguments necessary to achieve a polynomial runtime. As a result, we get a worse runtime than what [LR01] would imply.

Lemma 3.5. *Given valid lower and upper bounds $0 \leq L \leq z^* \leq U$, Algorithm 1 is a $(1+\epsilon)$-approximation for the Hopbounded Restricted Shortest Path problem that runs in time $O(\beta mnU/L\epsilon)$.*

Proof. The approximation ratio is directly implied by Lemma 3.4.

Each edge in the algorithm is examined a constant number of times, so the overall time complexity is

$$\begin{aligned} O(\beta m\tilde{U}) &= O\left(\beta m \frac{n}{\epsilon}\frac{U}{L} + nm\right) \\ &= O\left(\beta m \frac{n}{\epsilon}\frac{U}{L}\right) && (U \geq L \text{ and } \epsilon \leq 1) \end{aligned}$$

□

Now we find upper and lower bounds U and L such that $U/L \leq n$, just as in [LR01], to give an overall runtime of $O(\beta mn^2/\epsilon)$ for our problem. Let $\ell_1 < \ell_2 < \ldots, \ell_p$ be the distinct lengths of all edges in E_M (note that $p \leq m$). Let E_i be the set of edges in E_M with length at most ℓ_i, and let $G_i = (V, E_i)$. We also set $E_0 = \emptyset$. There must be some index $j \in [1,p]$ such that there exists a T-length-bounded, β-hopbounded

```
Input: Graph $G = (V, E)$, demand pair $s, t \in V$,
cost, distance, and hop metrics $\{z_e, \ell_e, h_e\}_{e \in E_M}$,
parameters $T, \beta, L, U$, and $\epsilon$

Let $S \leftarrow \frac{L\epsilon}{n+1}$, $\tilde{U} \leftarrow \lfloor U/S \rfloor + n + 1$

// scale edge costs
foreach edge $e \in E$ do
    Let $\tilde{z}_e \leftarrow \lfloor z_e/S \rfloor + 1$

// set base cases
foreach index $i = 0, 1, \dots, T$ do
    $D(s, i, 0) \leftarrow 0$
foreach index $j = 0, 1, \dots, \beta$ do
    $D(s, 0, j) \leftarrow 0$
foreach vertex $v \in V$ such that $v \neq s$ do
    foreach index $i = 0, 1, \dots, T$ do
        $D(v, i, 0) \leftarrow \infty$
    foreach index $j = 0, 1, \dots, \beta$ do
        $D(v, 0, j) \leftarrow \infty$

// build up DP table
foreach index $i = 1, 2, \dots, \tilde{U}$ do
    foreach index $j = 1, 2, \dots, \beta$ do
        foreach vertex $v \in V$ do
            $D(v, i, j) \leftarrow \min\{ D(v, i-1, j),\ D(v, i, j-1) \}$
            foreach edge $e \in \{(u, v) \mid \tilde{z}_{(u,v)} \leq i\}$ do
                $D(v, i, j) \leftarrow \min\{ D(v, i, j),\ \ell_e + D(v, i - \tilde{z}_e, j-1) \}$
            if $D(t, i, j) \leq T$ then
                Return the corresponding path
Return FAIL
```

Algorithm 1: Hopbounded Restricted Shortest Path Algorithm

path in G_j, but not in G_{j-1}. As observed in Claim 1 of [LR01], this means that $\ell_j \leq z^* \leq n\ell_j$, giving bounds $L = \ell_j$ and $U = n\ell_j$, with $U/L = n$. To find ℓ_j, we can binary search in the range $\ell_1, \dots, \ell_p$, running a shortest hopbounded path algorithm at each step of the binary search to check if there exists a T-length-bounded, β-hopbounded $s - t$ path on the subgraph G_i corresponding to that step. The binary search requires $O(\log m)$ steps, and the shortest hopbounded path algorithm takes $O(m \log n)$ time, so the runtime for finding these bounds is dominated by the runtime of the the DP algorithm.

In summary, the Hopbounded Restricted Shortest Path algorithm is as follows: Run binary search on $\ell_1, \dots, \ell_p$ to find ℓ_j and get bounds U and L. Then run the dynamic programming algorithm, Algorithm 1, using bounds U and L.

Lemma 3.6. *The Hopbounded Restricted Shortest Path problem admits an FPTAS.*

3.1.4 Proof of Theorem 3.3

With an FPTAS for the Hopounded Restricted Shortest Path problem (by Lemma 3.6), we have an approximate separation oracle for LP(Oracle 1). Using the ellipsoid algorithm with this oracle, we find a solution $\mathbf{z}$ for LP(Oracle 1). While $\mathbf{z}$ may not be feasible, it only violates each constraint by a factor of at most $(1+\epsilon)$. That is, $\mathbf{z}$ satisfies $(1+\epsilon)\sum_{e \in P} z_e \geq 1$ for all $P \in \mathcal{P}_{s,t}$. Thus if we scale $\mathbf{z}$ by $(1+\epsilon)$, we get a feasible solution. Let $\mathbf{z'}$ be this scaled solution, where $z'_e = (1+\epsilon)z_e$ for all $e \in E_M$. We then also have that $\mathbf{z'}$ is a $(1+\epsilon)$-approximation for LP(Oracle 1). This is implied by the fact that for any feasible solution $\mathbf{z''}$ of LP(Oracle 1), the value of the objective on $\mathbf{z}$ is at most the value of the objective on $\mathbf{z''}$ (the entire feasible region satisfies the $(1+\epsilon)$ "approximate" constraints, and therefore the feasible region is in the search space of the ellipsoid algorithm). That is, $\sum_{e \in E_M} z_e x_e \leq \sum_{e \in E_M} z''_e x_e$ for all feasible solutions $\mathbf{z''}$. Thus we have a $(1+\epsilon)$-approximation for LP(Oracle 1).

The purpose of solving LP(Oracle 1) is to give a separation oracle for our starting LP, LP(Hopset). We therefore must show that the $(1+\epsilon)$-approximation we have for LP(Oracle 1) translates to a $(1+\epsilon)$-approximation for the hopset LP. The approach is similar to how we approximate LP(Oracle 1). Running ellipsoid with the LP(Oracle 1) approximation as a separation oracle, we get a solution $\mathbf{x}$ that approximately (within a factor $(1+\epsilon)$) satisfies the constraints. We can scale $\mathbf{x}$ by a factor $(1+\epsilon)$ to get a feasible solution $\mathbf{x'}$. Additionally, we have $\sum_{e \in E_M} x_e \leq \sum_{e \in E_M} x''_e$ for all feasible solutions $\mathbf{x''}$. We therefore have a $(1+\epsilon)$-approximation for the hopset LP relaxation.

4 Approximation for Hopbound 2

When the hopbound is 2, we can get improved bounds by using the hopset version of a spanner approximation algorithm. Interestingly, while we will require the hopbound to be 2, we can handle arbitrary distance bound functions (including exact hopsets, $(1+\epsilon)$-stretch hopsets, and larger stretch hopsets), in contrast with the spanner version which requires all edges to have unit length and requires the *stretch* to be at most 2.

We use a version of the LP rounding algorithm for stretch 2 from [DK11]. In particular, we first solve our LP relaxation (LP(Hopset)) to get an optimal solution $\mathbf{x}$, and then round as follows. First, every vertex $v \in V$ chooses a threshold $T_v \in [0,1]$ uniformly at random. We then return as a hopset the set of edges $E' = \{(u,v) : \min(T_u, T_v) \leq (c \ln n) \cdot x_{(u,v)}\}$, where c is an appropriately chosen constant.

Lemma 4.1. $\mathbb{E}[c(E')] \leq O(\ln n) \cdot \mathsf{OPT}$

Proof. Fix some $e = (u,v) \in \widetilde{E}$. Clearly $\Pr[e \in E'] \leq (2c \ln n) x_e$, and hence $\mathbb{E}[c(E')] \leq \sum_{e \in \widetilde{E}} (2c \ln n) x_e \leq (2c \ln n) \cdot \mathsf{OPT}$. □

Lemma 4.2. *E' is a valid hopset with high probability.*

Proof. Fix some $(u,v) \in V \times V$. If $x_{(u,v)} \geq 1/(c \ln n)$ then our algorithm will directly include the (u,v) edge. So without loss of generality, we may assume that $x_{(u,v)} \leq 1/2$. As discussed, (LP(Hopset)) is equivalent to the flow LP, and hence $\mathbf{x}$ supports flows $f : \mathcal{P}_{u,v} \to [0,1]$ such that $\sum_{P \in \mathcal{P}_{u,v}} f_P \geq 1$. Since $\beta = 2$ and $x_{(u,v)} \leq 1/2$, this means that $\sum_{P \in \mathcal{P}_{u,v} \setminus \{(u,v)\}} f_P \geq 1/2$, i.e., at least 1/2 flow is sent on paths of length 2 in $\mathcal{P}_{u,v}$. Let $W \subseteq V \setminus \{u,v\}$ be the set of nodes that are the middle node of such a path, and for each $w \in W$ let P_w denote the path $u - w - v \in \mathcal{P}_{u,v}$.

Fix $w \in W$. Note that if $T_w \leq (c \ln n) f_{P_w}$ then our algorithm will include the path P_w, since the $\mathbf{x}$ variables support the flow f. Hence the probability that we fail to include every path $P \in \mathcal{P}_{u,v}$ is at most

$$\prod_{w \in W} (1 - (c \ln n) f_{P_w}) \leq \exp\left(- \sum_{w \in W} (c \ln n) f_{P_w} \right) \leq \exp(-(c \ln n)(1/2)) = n^{-c/2}.$$

We can now take a union bound over all $(u,v) \in V \times V$ to get that the probability that E' is not a valid hopset is at most $n^{2-(c/2)}$. Hence by changing c, we can make the failure probability $1/n^{c'}$ for arbitrarily large constant c'. □

Theorem 4.3. *There is an $O(\ln n)$-approximation algorithm for* Generalized β-Hopset *when $\beta = 2$.*

Proof. This is directly implied by Lemmas 4.1 and 4.2. □

5 Approximation Algorithms for General Hopbounds

Continuing the connection to spanners, there is a reduction from Generalized β-Hopset to the directed pairwise weighted spanner problem (where "weighted" refers to edge costs). The most general version of this problem, studied by [GKL23], allows for any demand set, any positive rational edge costs, integer edge weights in $\texttt{poly}(n)$, and arbitrary distance bound functions. The reduction starts with the transitive closure G_M, and builds a layered graph with $\beta+1$ copies of each node in G_M, and β copies of each edge (see Section 5.1.2 for a more detailed description of the reduction). Since [GKL23] achieved an $\widetilde{O}(n^{4/5+\epsilon})$-approximation for directed pairwise weighted spanners, this reduction immediately gives an $\widetilde{O}((\beta n)^{4/5+\epsilon})$-approximation for Generalized β-Hopset.

In this section, we improve upon this result and get an $\widetilde{O}(n^{4/5+\epsilon})$-approximation for Generalized β-Hopset, removing the dependence on β. We will give approximation algorithms in terms of n, β, OPT, and "local neighborhood size." All of our algorithms are based on spanner algorithms, and we must modify them (and provide different analyses) to accommodate the hop constraint. We will then trade these algorithms off with known existential hopset results to get approximations (in terms of n and β) in many regimes, including the general setting (directed graphs, arbitrary stretch), and in more restricted settings, such as small and large hopbound, and specific stretch regimes. The approximation we get in any given regime will be a function of how good the existential bounds are for that regime, so better existential results will result in better approximations.

Our first algorithm, the junction tree algorithm of Section 5.1, will perform best when β and OPT are relatively small. Our second and third algorithms, the star sampling and randomized LP rounding algorithms of Section 5.2, will together give better approximations as OPT gets larger.

5.1 Junction Tree Algorithm

In this section, we prove the following theorem for directed Generalized β-Hopset.

Theorem 5.1. *There is a polynomial-time $\widetilde{O}(\beta n^{\epsilon} \cdot \mathsf{OPT})$-approximation for directed* Generalized β-Hopset.

To prove the theorem, we give an algorithm similar to a subroutine of the directed pairwise weighted spanner algorithm of [GKL23], where "weighted" refers to edge costs. Just as for hopsets, the directed pairwise weighted spanner problem does not have $n-1$ as a lower bound on the cost of the optimal solution. This allows their techniques to be useful in our setting.

In the pairwise weighted spanner subroutine of [GKL23], they define a variant of the junction tree (the "distance-preserving junction tree"). Junction trees are rooted trees that satisfy demands, and *good* junction trees satisfy many demands at low cost; that is, they have low "density." Junction trees have been used in several spanner approximation algorithms (e.g. [GKL23, CDKL20, GKL24b]). In [GKL23], they give an algorithm that iteratively buys their version of low density junction trees until all demands are satisfied. Our algorithm will follow the same structure. The main technical work in this section is in showing that low-density *hopbounded* junction trees exist in our setting, and that we can use the subroutine of [GKL23] to find these hopbounded junction trees, even though their subroutine does not have any hop guarantees.

We note that the junction tree framework developed by [GKL24b] for multicriteria spanners also works for our setting. Their framework implies an $\widetilde{O}(|\mathcal{D}|^{\epsilon} \cdot \beta)$-approximation for finding the hopbounded minimum-density junction tree, and thus a $\widetilde{O}(\beta^2 \cdot |\mathcal{D}|^{\epsilon} \cdot \mathsf{OPT})$-approximation for directed Generalized β-Hopset. By tailoring our framework to hopsets, we achieve an $O(n^{\epsilon})$-approximation for finding the hopbounded min-density junction tree (note the lack of dependence on β), implying our main result of this section (Theorem 5.1), a $\widetilde{O}(\beta \cdot n^{\epsilon} \cdot \mathsf{OPT})$-approximation overall. By using our hopbounded junction tree framework, we lose a factor β in our overall approximation compared to what is implied by [GKL24b].

We first define a hopbounded variant of the junction tree, which we call an (i,j)-distance-preserving hopbounded junction tree. We parameterize by i, j, where $i + j \leq \beta$, to ensure that both "sides" of the rooted tree—the in-arborescence and the out-arborescence that make up the tree—are hopbounded by i and j, respectively, so that all paths in the tree have at most β hops.

Definition 5 ((i,j)-Distance-Preserving Hopbounded Junction Tree)**.** An (i,j)*-distance-preserving hop-bounded junction tree*, where $i + j \leq \beta$, is a subgraph of G_M that is a union of an in-arborescence and an out-arborescence, both rooted at some vertex $r \in V$, with the following properties: 1) every leaf of the in-arborescence has a path of at most i hops to r, 2) for every leaf w in the out-arborescence, there is a path of at most j hops from r to w, and 3) for some node s in the in-arborescence and some node t in the out-arborescence, there is an $s - t$ path through r with distance at most $Dist(s,t)$. The *density* of an (i,j)-distance-preserving hopbounded junction tree T is the ratio of the cost of T to the number demands settled by T.

Going forward, we will refer to the (i,j)-distance-preserving hopbounded junction tree as simply an "(i,j)-junction tree." Our algorithm will find and remove a low-density hopbounded junction tree from G_M, add its edges to the current solution, and repeat, until all demand pairs are settled. We will give an $O(n^\epsilon)$-approximation for finding these low-density junction trees. The algorithm will return a subgraph with total cost of $\widetilde{O}(\beta^2 n^\epsilon \cdot \mathsf{OPT}^2)$.

5.1.1 Existence of Low-Density Junction Trees

Let D be the set of *unsettled* vertex pairs at some iteration of the algorithm. We first argue that a hopbounded junction tree with density $O(\beta^2 \cdot \mathsf{OPT}^2 / \ |D|)$ always exists (at any iteration), where OPT is the cost of the optimal solution to the problem instance.

Lemma 5.2. *For any set of unsettled demands D, there exists an (i,j)-junction tree with density $O(\beta \cdot \mathsf{OPT}^2/|D|))$.*

Proof. Let H be an optimal solution subgraph to the graph G_M, and let S be the cheapest subgraph of G_M that settles all demands in D. We have that $c(S) \leq c(H) = \mathsf{OPT}$. We now look at the set of paths in S that settle the demands in D. Each of these $|D|$ paths must use some edge in $S \cap \widetilde{E}$; otherwise, the demand is already settled and cannot be in D. Due to averaging, there must be some edge $e \in S \cap \widetilde{E}$ that belongs to at least $|D|/|S \cap \widetilde{E}| \geq |D|/\mathsf{OPT}$ of these paths.

Let $D_e \subseteq D$ be the set of demands settled by paths through $e = (u,v)$ in S. Let P_e be these demand-settling paths (that is, P_e is the set of paths that settle the demands in D_e through e). We now place the demands in D_e into $O(\beta)$ classes as follows. Let x, y be nonnegative integers such that $x + y = \beta$. We say that a demand $(s,t) \in D_e$ is in class (x,y) if its corresponding path in P_e has at most x hops from s to u and at most y hops from u to t. Note that every demand in D_e belongs to at least one class. Recall that e belongs to at least $|D| \ / \ \mathsf{OPT}$ demand-settling paths in S. Again due to averaging, there must be some class containing $\Omega\left(|D| \ / \ (\beta \cdot \mathsf{OPT})\right)$ demands. Let (i,j) denote such a class, let $D_e^{(i,j)} \subseteq D_e$ denote the set of demands in this class, and let $P_e^{(i,j)}$ be their corresponding demand-settling paths.

Now consider the tree created by adding all paths in $P_e^{(i,j)}$, rooted at vertex u. This tree is an (i,j)-junction tree, as it is the union of an in-arborescence where each leaf has a path of at most i hops to u, and an out-arborescence where u has a path with at most j hops to each leaf, where $i + j \leq \beta$. This tree has cost at most OPT and settles at least $|D_e^{(i,j)}| = \Omega\left(|D| \ / \ (\beta \cdot \mathsf{OPT})\right)$ demands, and thus has density $O(\beta \cdot \mathsf{OPT}^2 / \ |D|)$. □

5.1.2 Layered Graph Reduction

We want to show that we can *find* a junction tree with low enough density at each iteration of the algorithm. To do so, we will use the junction-tree finding subroutine provided in [GKL23]. Their subroutine, however, finds non-hop-constrained junction trees. We therefore transform our input graph in order to use their subroutine to find (i,j)-junction trees. We build the following β-layered graph out of G_M.

Layered Graph Construction. To construct the layered graph $G_L = (V_L, E_L)$ with costs $c_L : E_L \to \{0,1\}$, weights $\ell_L : E_L \to \{1, 2, \ldots, \texttt{poly}(n)\}$, demand set $\mathcal{D}_L$, and distance bounds $Dist_L : \mathcal{D}_L \to \mathbb{N}_{\geq 0}$, we first create $\beta+1$ copies of each vertex $u \in V$ so that u corresponds to vertices $u_0, u_1, \ldots, u_\beta$ in V_L. For each edge $(u,v) \in E_M$ we add edges $\{(u_i, v_{i+1}) : i \in [0, \beta-1]\}$ to E_L. For each edge $e = (u_i, v_{i+1})$ of this type, we set $\ell_L(e) = \ell(u,v)$. We also set $c_L(e) = 1$ if $(u,v) \in \widetilde{E}$; otherwise we set $c_L(e) = 0$. For each vertex in V_L, we also add edges $\{(u_i, u_{i+1}) : i \in [0, \beta-1]\}$ to E_L, and set their costs and weights to 0. Finally, we add a demand (s_0, t_β) to $\mathcal{D}_L$ for demand each $(s,t) \in \mathcal{D}$.

By design, (i,j)-junction trees in G_M correspond to (i,j)-junction trees in G_L (and vice versa) with the same density. The proof of this is straightforward, and is given in the following pair of lemmas.

Lemma 5.3. *For any (i,j)-junction tree in G_M, there exists an (i,j)-junction tree in G_L with the same density.*

Proof. Fix an (i,j)-junction tree T in G_M. We build an (i,j)-junction tree T_L in G_L with the same density. Tree T is the union of an in-arborescence, which we denote as A^{in}, and an out-arborescence, denoted by A^{out}, both of which are rooted at some node r. We add node $r_i \in V_L$ to T_L as its root. For each vertex $u \in A^{in}$, let h_u be the number of hops (edges) on the $u-r$ path in A^{in}; likewise, for each vertex $w \in A^{out}$, let h_w be the number of hops on the $r-w$ path in A^{out}. For each edge $(u,v) \in A^{in}$, we add edge (u_{i-h_u}, v_{i-h_u+1}) (and corresponding nodes) to T_L. Similarly, for each edge $(u,v) \in A^{out}$, we add edge (u_{i+h_u}, v_{i+h_u+1}) (and corresponding nodes) to T_L.

Finally, we add "dummy paths" to T_L to ensure that the corresponding demands in $\mathcal{D}_L$ are settled. These dummy paths will handle demand-settling paths in T that have fewer that β hops. Let A_L^{in} and A_L^{out} denote the in- and out-arborescences of T_L, respectively. For each vertex $v_x \in A_L^{in}$, we add edges $\{(v_k, v_{k+1}) : 0 \le k < x\}$ (and corresponding nodes) to T_L (if they don't already exist in T_L). Similarly for each vertex $v_x \in A_L^{out}$, we add edges $\{(v_k, v_{k+1}) : x \le k < \beta\}$ (and corresponding nodes) to T_L (if they don't already exist in T_L). With these dummy paths, we ensure that if demand (s,t) is settled by a path in T with fewer than β hops, then the corresponding demand (s_0, t_β) is also settled in T_L.

Tree T_L has the same cost as T: for every edge $(u,v) \in T$ we add an edge (u_k, v_{k+1}), for some integer k, to T_L. Recall that (u,v) and (u_k, v_{k+1}) have the same cost, for any $0 \le k < \beta$. All other edges in T_L (i.e., all edges on dummy paths) are of the form (u_k, u_{k+1}), for some k, and have cost 0.

The number of demands satisfied in both trees is also the same, which we show by mapping each demand $(s,t) \in \mathcal{D}$ settled by T to a unique demand $(s_0, t_\beta) \in \mathcal{D}_L$ settled by T_L (and vice versa). We now show that if (s,t) is settled by T, then (s_0, t_β) is settled by T_L. Let $P = (s, a, b, c \ldots, t)$ be the path in T that settles (s,t). Then, path $P_L = (s_0, \ldots, s_k, a_{k+1}, b_{k+2}, c_{k+3}, \ldots, t_m, \ldots, t_\beta)$ is in T_L, for some k, m. Note that subpaths $(s_0, \ldots, s_k)$ and $(t_m, \ldots, t_\beta)$ are dummy subpaths. Paths P and P_L have the same length—any edge $(u,v) \in P$ has the same length as its corresponding edge (u_k, v_{k+1}) (for some k) in P_L, and all other edges (i.e., edges from the dummy subpaths $(s_0, \ldots, s_k)$ and $(t_m, \ldots, t_\beta)$, if they exist) have length 0. We therefore have that $d_{T_L}(s_0, t_\beta) = d_T(s,t) \le \mathit{Dist}(s,t) = \mathit{Dist}_L(s_0, t_\beta)$. Also, path P_L has exactly β hops. Thus, demand (s_0, t_β) is settled by T_L. It is also not difficult to see that each demand $(s_0, t_\beta) \in \mathcal{D}_L$ settled by T_L can be mapped to a unique demand $(s,t) \in \mathcal{D}$ settled by T, using similar arguments. Trees T and T_L therefore have the same cost and settle the same number of demands, and so have the same density. $\square$

Lemma 5.4. *For any (i,j)-junction tree in G_L, there exists an (i,j)-junction tree in G_M with the same density.*

Proof. Given an (i,j)-junction tree T_L of G_L, we can build an (i,j)-junction tree T in G_M with the same density. Note that edges only exist between adjacent layers in G_L (and in T_L)—namely, all edges in T_L are of the from (u_k, v_{k+1}) for some k. For each $k \in [0, \beta]$ and for each edge (u_k, v_{k+1}) in T_L such that $u \neq v$, we add edge $(u,v) \in G_M$ (and corresponding nodes) to T.

Trees T_L and T have the same cost: For each edge $(u_k, v_{k+1}) \in T_L$ (for some k) with cost 1, we add edge (u,v) to T, which also has cost 1. For all other edges in T_L (all of which have no cost), we either add the corresponding edge to T, which also has no cost, or we add no edge.

Both trees also settle the same number of demands. Each demand $(s_0, t_\beta) \in \mathcal{D}_L$ settled by T_L can be mapped to a unique demand $(s,t) \in \mathcal{D}$ settled by T. Let $P_L = (s_0, \ldots, s_k, a_{k+1}, b_{k+2}, c_{k+3}, \ldots, t_m, \ldots, t_\beta)$ be the path that settles (s_0, t_β) in T_L. Then, the path $P = (s, a, b, c, \ldots, t)$ is in T. Paths P and P_L have the same length—any edge of the form $(u_k, v_{k+1}) \in P_L$ (for some k), where $u \neq v$, has the same length as its corresponding edge $(u, v) \in P$. All other edges in P_L (i.e., edges from the dummy subpaths if they exist) have length 0. We therefore have that $d_T(s,t) = d_{T_L}(s_0, t_\beta) \leq Dist_L(s_0, t_\beta) = Dist(s,t)$. Path P also has at most β hops, so (s,t) is settled by T. It is also not difficult to see that each demand $(s,t) \in \mathcal{D}$ settled by T can be mapped to a unique demand $(s_0, t_\beta) \in \mathcal{D}_L$ settled by T_L, using similar arguments. We've shown that T_L and T have the same cost and settle the same number of demands, and so they have the same density. □

Let $\Delta(T)$ denote the density of junction tree T. The above lemmas imply the following:

Corollary 5.5. *Let T^* be the min-density (i,j)-junction tree (over all possible i,j) in G_M, and let T_L^* be the min-density (i,j)-junction tree (over all possible i,j) in G_L. Then, $\Delta(T^*) = \Delta(T_L^*)$.*

Proof. By Lemma 5.3, we have that $\Delta(T_L^*) \leq \Delta(T^*)$. By Lemma 5.4, $\Delta(T^*) \leq \Delta(T_L^*)$. Therefore, $\Delta(T^*) = \Delta(T_L^*)$. □

We now use this Corollary to reduce from finding min-density (i,j)-junction trees in G_M to finding them in G_L. We say that Min Density (i,j)-Junction Tree is the optimization problem of finding the minimum density (i,j)-junction tree in an input graph, over all possible values of i,j.

Lemma 5.6. *If there is an α-approximation algorithm for* Min Density (i,j)-Junction Tree *on G_L, then there is also an α-approximation algorithm for* Min Density (i,j)-Junction Tree *on G_M.*

Proof. Suppose we have an α-approximation for Min Density (i,j)-Junction Tree on graph G_L. Then, the following is an algorithm for Min Density (i,j)-Junction Tree on G_M. First, run the α-approximation algorithm on G_L, and let T_L be the tree returned by the algorithm. Using the procedure described in Lemma 5.4, we can build (in polynomial time) a valid (i,j)-junction tree T of G_M with the same density as T_L. The density of T is as follows:

$$\Delta(T) = \Delta(T_L) \leq \alpha \cdot \Delta(T_L^*) = \alpha \cdot \Delta(T^*).$$

The final equality is due to to Corollary 5.5. □

5.1.3 Junction Tree-Finding Subroutine

We now show that we can find low-density junction trees at each iteration of the algorithm. Although (i,j)-junction trees are hopbounded by definition, we can now use the following length-bounded junction tree-finding subroutine of [GKL23] to find hopbounded junction trees, thanks to the reduction to the β-layered graph G_L.

Lemma 5.7 (Lemma 16 of [GKL23])**.** *For any constant $\epsilon > 0$, there is a polynomial-time approximation algorithm for finding the minimum density distance-preserving junction tree. That is, there is a polynomial time algorithm which, given a weighted directed n-vertex graph $G = (V,E)$ where each edge $e \in E$ has cost $c(e) \in \mathbb{R}_{\geq 0}$ and integral length $\ell(e) \in \{0, 1, \ldots, \texttt{poly}(n)\}$, terminal pairs $\mathcal{D} \subseteq V \times V$, and distance bounds $Dist : \mathcal{D} \to \mathbb{N}$ for each terminal pair $(s,t) \in \mathcal{D}$, approximates the following problem to within an $O(n^\epsilon)$ factor:*

- *Find a non-empty set of edges $F \subseteq E$ minimizing the ratio:*

$$\min_{r \in V} \frac{\sum_{e \in F} c(e)}{|\{(s,t) \in \mathcal{D} : d_{F,r}(s,t) \leq Dist(s,t)\}|}$$

where $d_{F,r}(s,t)$ is the length of the shortest path using edges in F which connects s to t while going through r (if such a path exists). We call this problem Min Density Length-Bounded Junction Tree.

This gives an $O(n^\epsilon)$-approximation algorithm for finding the min-density (i,j)-junction tree on G_M.

Lemma 5.8. *There is an $O(n^\epsilon)$-approximation for* Min Density (i,j)-Junction Tree *on G_M.*

Proof. By Lemma 5.6, we can prove the lemma by giving an $O(n^\epsilon)$-approximation for Min Density (i,j)-Junction Tree on G_L. The algorithm is as follows: Simply run the algorithm of Lemma 5.7 on G_L.

We now show that the tree T_L returned by this algorithm is an (i,j)-junction tree of G_L, where $i+j=\beta$, and that the density of T_L is at most a factor $O(n^\epsilon)$ of the density of the optimal tree. Tree T_L has some root r_i. Fix a demand $(s_0, t_\beta) \in \mathcal{D}_L$ that has length at most $Dist_L(s,t)$ in T_L. Due to the structure of G_L, the path from s_0 to t_β in T_L must have i hops from s_0 to r_i and $\beta - i$ hops from r_i to t_β. Thus T_L is an (i,j)-junction tree. To see that this is an $O(n^\epsilon)$-approximation, first observe that the optimal density (i,j)-junction tree is a feasible solution to Min Density Length-Bounded Junction Tree on G_L. As for the approximation ratio, the algorithm gives a $O(|V_L|^\epsilon) = O(\beta^\epsilon |V|^\epsilon) = O(n^{\epsilon'})$ approximation, where $n = |V|$ and $\epsilon' > 0$ is an arbitrarily small constant. □

Lemma 5.9. *There is a polynomial-time algorithm that finds an (i,j)-junction tree with density $O(\beta n^\epsilon \cdot \mathsf{OPT}^2/|D|))$, where D is the set of unsettled demands in G.*

Proof. By Lemma 5.2, there exists an (i,j)-junction tree with density $O(\beta \cdot \mathsf{OPT}^2/|D|))$. We can run the $O(n^\epsilon)$-approximation algorithm (Lemma 5.8) on G_M, which outputs an (i,j)-junction tree with density $O(\beta n^\epsilon \cdot \mathsf{OPT}^2/|D|))$. □

5.1.4 Proof of Theorem 5.1

By iteratively buying these low-density (i,j)-junction trees, we get an $O(\beta n^\epsilon \cdot \mathsf{OPT})$-approximation for Generalized β-Hopset.

Proof. The algorithm for Generalized β-Hopset builds and returns subgraph H, which is initialized as empty. Let D be initialized as the set of unsettled demands in the input. The algorithm first finds an (i,j)-junction tree T with density $O(\beta n^\epsilon \cdot \mathsf{OPT}^2/|D|))$, as described in Lemma 5.9. It then removes T from G_M, adds T to H, and removes all settled demands from D. This process repeats until all demands are settled. Now we show that the algorithm gives an $O(\beta n^\epsilon \cdot \mathsf{OPT}^2)$-approximation. Suppose the algorithm runs for ℓ total iterations. For iteration k of the algorithm, let T_k be the (i,j)-junction tree found at that iteration, let $c(T_k)$ be its cost, and let s_k be the number of demands settled by T_k. Let D_k be the set of unsettled demands at the start of iteration k. The cost of H is the following:

$$\begin{aligned} c(H) = \sum_{k=1}^{\ell} c(T_k) = \sum_{k=1}^{\ell} O\left(\frac{\beta n^\epsilon \cdot \mathsf{OPT}^2}{|D_k|} \cdot s_k\right) &= O\left(\beta n^\epsilon \cdot \mathsf{OPT}^2 \sum_{k=1}^{\ell} \frac{s_k}{|D_k|}\right) \\ &= O\left(\beta n^\epsilon \cdot \mathsf{OPT}^2 \cdot \log n\right) \qquad (|D|^{\text{th}} \text{ Harmonic number}) \\ &= \widetilde{O}\left(\beta n^\epsilon \cdot \mathsf{OPT}\right) \cdot \mathsf{OPT}. \end{aligned}$$

Note that $\sum_{k=1}^{\ell} \frac{s_k}{|D_k|} = O(H_{|D|}) = O(\log n)$, where $H_{|D|}$ is the $|D|$th Harmonic number. □

5.2 Star Sampling with Randomized LP Rounding Algorithm

In this section we prove the following theorem.

Theorem 5.10. *There is a randomized algorithm for directed* Generalized β-Hopset *with expected approximation ratio $O(n \ln n \,/ \sqrt{\mathsf{OPT}})$.*

The pair of algorithms we give closely follow the $\widetilde{O}(n^{2/3})$- and $\widetilde{O}(\sqrt{n})$-approximations for the unweighted k-spanner problem, given by [DK11] and [BBM+11], respectively. The k-spanner algorithm is a trade-off between two algorithms: an arborescence sampling algorithm for settling a class of edges (or demands) that they call "thick," and a randomized LP rounding algorithm for settling "thin" edges. For hopsets we will settle these thick demands by sampling directed stars instead of arborescences, and for thin demands we will use a similar LP rounding approach. Although our hopset algorithms are similar to the k-spanner algorithms, we get a different approximation ($\widetilde{O}(n/\sqrt{\mathsf{OPT}})$ for hopsets versus $\widetilde{O}(\sqrt{n})$ for spanners). This is because [DK11, BBM+11] take advantage of the fact that for spanners, $\mathsf{OPT} \geq n-1$. This is not the case for hopsets so we get a different approximation out of the algorithms, in terms of OPT. This approach is also similar to that of [CDKL20] for pairwise distance preservers, where again, $\Omega(n)$ is not a lower bound for OPT.

We note that our $O(n \ln n / \sqrt{\mathsf{OPT}})$-approximation is achieved by trading off the two aforementioned algorithms (star-sampling and randomized-rounding). To later achieve the optimal trade-off with other algorithms, one should a priori treat each of these two algorithms as separate, with their own individual approximation ratios. It is however equivalent to trade these two algorithms off first and treat them as one combined algorithm, which we do going forward. This is because these are our only algorithms that will depend on the "local neighborhood size" parameter.

To define thick and thin demands, we must first define subgraphs $G^{s,t}$ for all demands (s,t), as in [DK11, BBM+11]:

Definition 6. For a demand $(s,t) \in \mathcal{D}$, let $G^{s,t} = (V^{s,t}, E^{s,t})$ be the subgraph of G_M induced by the vertices on paths in $\mathcal{P}_{s,t}$. We call $|V^{s,t}|$ the *local neighborhood size.*

Definition 7 (Thick and Thin Demands)**.** Let b be a parameter in $[1,n]$. If $|V^{s,t}| \geq n/b$ then the corresponding demand (s,t) is thick, otherwise it is thin. We shall always assume that $b = \sqrt{\mathsf{OPT}}$.

Let $\mathcal{D}_{thick}$ and $\mathcal{D}_{thin}$ be the set of all thick and thin demands, respectively. We will run two algorithms to build two edge sets, E' and E'', such that all thick demands are settled by E' and all thin demands are settled by E''. The set E' will have cost $O(bn \ln n)$ in expectation, while E'' will have cost $O((n/b)\ln n \cdot \mathsf{OPT})$ in expectation. The optimal trade-off of these algorithms has $b = \sqrt{\mathsf{OPT}}$, so each edge set will have cost $O(n \ln n \cdot \sqrt{\mathsf{OPT}})$ in expectation.

5.2.1 Star-Sampling Algorithm for Thick Demands

We describe the random sampling subroutine for constructing the edge set E', which will settle all thick demands (Algorithm 2).

```
Input: Graph $G_M = (V, E_M)$
Let $E' \leftarrow \emptyset$, $S \leftarrow \emptyset$                    // Set $S$ is only used for the analysis
foreach index $i = 1, 2, \ldots, b \ln n$ do
    $v \leftarrow$ a uniformly random element from $V$
    $T_v^{in} \leftarrow$ inward star of $G_M$ rooted at $v$
    $T_v^{out} \leftarrow$ outward star of $G_M$ rooted at $v$
    $E' \leftarrow E' \cup T_v^{in} \cup T_v^{out}$, $S \leftarrow S \cup \{v\}$
foreach unsettled demand $(s,t) \in \mathcal{D}_{thick}$ do
    $E' \leftarrow E' \cup (s,t)$
Return $E'$
```

Algorithm 2: Star-Sampling Algorithm

This algorithm is nearly identical to that of [DK11]. The only difference is that, since we operate on the weighted transitive closure of G, we build directed in- and out-stars as opposed to the shortest path in- and out-arborescences used for the spanner setting.

We now show that E' has the desired cost in expectation. While [DK11] proves this for spanners, it is easy to see that a near identical argument also holds for hopsets in G_M. We restate the proof for completeness.

Lemma 5.11 ([DK11])**.** *Algorithm 2, in polynomial time, computes an edge set E' that settles all thick demands and has expected cost $O(bn \ln n)$. If $b = \sqrt{\mathsf{OPT}}$, then the expected size is $O(n \ln n \cdot \sqrt{\mathsf{OPT}})$.*

Proof. After the execution of the first for loop in Algorithm 2, $|E'| \leq 2(n-1)b \ln n$.

If some vertex v from a set $V^{s,t}$ appears in the set S of vertices selected by Algorithm 2, then T_v^{in} and T_v^{out} contain shortest, 1-hop paths from s to v and from v to t, respectively. Thus, both paths are contained in E'. Since $v \in V^{s,t}$, the sum of lengths of these two paths is at most $Dist(s,t)$. Therefore, if $S \cap V^{s,t} \neq \emptyset$, then the demand (s,t) is settled. For a thick demand (s,t), the set $S \cap V^{s,t}$ is empty with probability at most $(1-1/b)^{b \ln n} \leq e^{-\ln n} = 1/n$. Thus, the expected number of unsettled thick demands added to E' in the final for loop of Algorithm 2 is at most $|\mathcal{D}|/n \leq n$.

The final for loop ensures that E', returned by the algorithm, settles all thick demands. Computing in- and out-stars and determining whether a demand is thick can be done in polynomial time. □

5.2.2 Randomized LP Rounding Algorithm for Thin Demands

We now give the algorithm for finding a set E'' to settle thin demands. [BBM+11] introduces the notion "anti-spanners," which is crucial for the algorithm and analysis for settling thin demands. In particular, they formulate an anti-spanner covering LP that captures the problem of settling all thin demands. They then solve the LP (with high probability) by constructing a separation oracle that utilizes randomized rounding. We will also use randomized LP rounding, though instead of rounding the solution to an "anti-hopset" covering LP, we will round based on LP(Hopset). Our LP is stronger than the "anti-hopset" covering LP, since our LP is for *fractional* cuts against valid paths, while the anti-hopset covering LP is only for integer cuts.

Going forward, we will assume without loss of generality that we know the value of the optimal solution—OPT is in $\{0, 1, \ldots, n^2\}$, so we can just try each of these values for OPT and return the smallest hopset found over all tries. We can therefore replace the objective function of LP(Hopset) with the following:

$$\sum_{e \in \widetilde{E}} x_e \leq \mathsf{OPT} \tag{4}$$

We use this modified version of LP(Hopset) for the randomized rounding algorithm. Given a fractional solution $\mathbf{x}^*$ to LP(Hopset), our algorithm will return an edge set E'' that, with high probability, will cost at most $2\mathsf{OPT} \cdot 2(n/b) \ln n$ and satisfy all thin demands (see Algorithm 3). We say that the algorithm *fails* if $c(E'') > 2\mathsf{OPT} \cdot 2(n/b) \ln n$ or if E'' does not satisfy all thin demands. The algorithm will fail with low probability.

```
Input: Graph $G_M = (V, E_M)$, LP(Hopset) fractional solution $\mathbf{x}^*$
Let $E'' \leftarrow \emptyset$

// sample edges into $E''$
foreach edge $e \in E_M$ do
    Let $p_e \leftarrow \min(1, 2(n/b) \ln n \cdot x_e^*)$
    Add $e$ to $E''$ with probability $p_e$

if $E''$ settles all thin demands then
    Return $E''$
else
    Return $E_M \setminus E$
```

Algorithm 3: Randomized LP Rounding Algorithm

To show that with high probability, Algorithm 3 does not fail, we start by defining "anti-hopsets," the analogous of anti-spanners in the hopset setting. For a given demand (s,t), an anti-hopset is a set of edges such that removing them from G_M results in no valid paths from s to t in what remains.

Definition 8 (Anti-Hopsets)**.** An edge set $C \subseteq E$ is an anti-hopset for demand $(s,t) \in \mathcal{D}$ if there is no β-hopbounded path of length at most $Dist(s,t)$ in $G_M \setminus C$. If no proper subset of an anti-hopset C is an anti-hopset, we say that C is a minimal.

Thus, a set of edges is a valid hopset if and only if it is a hitting set for the collection of (minimal) anti-hopsets—that is, to be a valid hopset, an edge set must include at least one edge from every anti-hopset. We now show that the probability is exponentially small that the algorithm fails. The argument is very similar to that given by [BBM+11] for spanners; we state it here for completeness.

Lemma 5.12 (Theorem 2.2 of [BBM+11])**.** *The probability that Algorithm 3 fails is exponentially small in n.*

Proof. There are two different events that can cause the algorithm to fail:

1. The cost of the sampled set E'' is too high—that is, $c(E'') > 2\mathsf{OPT}\cdot 2(n/b)\ln n$. The expected cost of E'' is at most $2(n/b)\ln n \cdot \sum_{e\in E_M\setminus E} x_e \leq \mathsf{OPT}\cdot 2(n/b)\ln n$. By the Chernoff bound (recall that $b = \sqrt{\mathsf{OPT}}$), we have that $\Pr[c(E'') > 2\mathsf{OPT}\cdot 2(n/b)\ln n] \leq e^{-c\cdot\mathsf{OPT}\cdot(n/b)\ln n} = e^{-cn\ln n\cdot\sqrt{\mathsf{OPT}}} = e^{-\Omega(n\ln(n))}$. Thus, the probability that the algorithm fails because $c(E'') > 2\mathsf{OPT}\cdot 2(n/b)\ln n$ is exponentially small.

2. E'' does not settle all thin demands. We prove that the probability that E'' does not settle all thin demands (that is, that E'' does not intersect all minimal anti-hopsets for thin demands) is exponentially small in the following Lemma.

Lemma 5.13 (Lemma 2.3 of [BBM+11])**.** *The probability that there exists a demand (s,t) and a minimal anti-hopset C for it such that $C \subset E^{s,t}\setminus E''$ is at most $|\mathcal{D}_{thin}|\cdot(1/bn)^{n/b}$. In particular, if $b = \sqrt{\mathsf{OPT}}$, then the probability is at most $|\mathcal{D}_{thin}|\cdot(1\,/\,n\sqrt{\mathsf{OPT}})^{n/\sqrt{\mathsf{OPT}}}$.*

Proof. First, we bound the total number of minimal anti-hopsets for thin demands.

Proposition 5.14 (Proposition 2.1 of [BBM+11])**.** *If (s,t) is a thin demand, then there are at most $(n/b)^{n/b}$ minimal anti-hopsets for (s,t).*

Proof. Fix a thin demand (s,t) and consider an arbitrary minimal anti-hopset C for (s,t). For the rest of this argument, for any two paths of the same length between the same pair of vertices, we consider one path to be shorter than the other if that path has fewer hops. More specifically, for any vertex pair $x,y\in V$ and any two $x-y$ paths P,P' that have the same length, we say that P is shorter than P' if P has fewer hops (edges).

Let A_C be the outward shortest path tree (arborescence) rooted at s in the graph $(V^{s,t}, E^{s,t}\setminus C)$. Our tie-breaking of same-length paths with different hops ensures that A_C includes shortest paths with the lowest number of hops (recall that every edge $(u,v)\in E_M$ is a shortest path from u to v). Denote by $f^{(\beta)}_{A_C}(u)$ the β-hopbounded distance from s to u in the tree A_C. If there is no β-hopbounded directed path from s to u in A_C, let $f^{(\beta)}_{A_C}(u) = \infty$.

We show that $C = \left\{(u,v)\in E^{s,t} : f^{(\beta)}_{A_C}(u) + \ell(u,v) < f^{(\beta)}_{A_C}(v)\right\}$, and thus, A_C uniquely determines C for a given thin demand (s,t). If $(u,v)\in C$, then, since C is a *minimal* anti-hopset, there exists a β-hopbounded path from s to t of length at most $Dist(s,t)$ in the graph $(G_M\setminus C)\cup\{(u,v)\}$. This path must lie in $(G^{s,t}\setminus C)\cup\{(u,v)\}$ and must contain the edge (u,v). Thus, the β-hopbounded distance from s to t in the graph $(G^{s,t}\setminus C)\cup\{(u,v)\}$ is at most $Dist(s,t)$ and is strictly less than $f^{(\beta)}_{A_C}(t)$. Hence, A_C is not the shortest path tree in the graph $(G^{s,t}\setminus C)\cup\{(u,v)\}$. Therefore, $f^{(\beta)}_{A_C}(u) + \ell(u,v) < f^{(\beta)}_{A_C}(v)$. If $(u,v)\in E^{s,t}$ satisfies the condition $f^{(\beta)}_{A_C}(u) + \ell(u,v) < f^{(\beta)}_{A_C}(v)$, then $(u,v)\notin E^{s,t}\setminus C$ (otherwise, A_C would not be the shortest path tree), hence $(u,v)\in C$.

We now count the number of outward trees rooted at s in $G^{s,t}\setminus C$. For every vertex $u\in V^{s,t}$, we may choose the parent vertex in at most $|V^{s,t}|$ possible ways (if a vertex is isolated we assume that it is its own parent), thus the total number of trees is at most $|V^{s,t}|^{|V^{s,t}|} \leq (n/b)^{n/b}$ □

Proposition 5.15 (Proposition 2.2 of [BBM+11])**.** *For a demand (s,t) and a minimal anti-hopset C for (s,t), the probability that $E'' \cap C = \emptyset$ is at most $e^{-2(n/b)\ln n}$.*

Proof. Suppose there is an anti-hopset edge $e \in C$ such that $x_e^* \geq (2(n/b)\ln n)^{-1}$. In this case, e is in E'' with probability 1, and we are done. Otherwise, the probability that e is in E'' is exactly $2(n/b)\ln n \cdot x_e$. In this case, the probability that E'' does not include e is

$$\prod_{e \in C} (1 - 2(n/b)\ln n) < \exp\left(-\sum_{x \in C} 2(n/b)\ln n \cdot x_e^* \right) \leq e^{-2(n/b)\ln n}.$$

The first inequality holds from the fact that $1 - x < \exp(-x)$ for $x > 0$. For the last inequality, observe that every anti-hopset is an integer cut against valid paths. Thus, each anti-hopset C corresponds to an LP(Hopset) constraint of the form $\sum_{e \in E_M} z_e x_e \geq 1$, where $\mathbf{z} \in \mathcal{Z}_{s,t}$ is the indicator vector for cut C. □

To finish the proof of Lemma 5.13, we use Propositions 5.14 and 5.15 to take a union bound over all minimal anti-hopsets for all thin demands. Let $\mathcal{S}_{s,t}$ be the set of all minimal anti-hopsets for a thin demand (s,t), and let $\mathcal{S}$ be the collection of all minimal anti-hopsets for all thin demands. The probability that E'' does not intersect all minimal anti-hopsets in $\mathcal{S}$ is the following:

$$\begin{aligned}
\Pr[E'' \text{ is not a hitting set for } \mathcal{S}] &\leq \sum_{(s,t) \in \mathcal{D}_{thin}} \sum_{C \in \mathcal{S}_{s,t}} e^{-2(n/b)\ln n} \\
&\leq |\mathcal{D}_{thin}| \cdot \left(\frac{n}{b}\right)^{n/b} \cdot e^{-2(n/b)\ln n} \\
&= |\mathcal{D}_{thin}| \cdot \left(\frac{1}{bn}\right)^{n/b} \\
&= |\mathcal{D}_{thin}| \cdot \left(\frac{1}{n\sqrt{\mathsf{OPT}}}\right)^{\frac{n}{\sqrt{\mathsf{OPT}}}}. \qquad (b = \sqrt{\mathsf{OPT}})
\end{aligned}$$

If $\mathsf{OPT} = \widetilde{\Theta}(n^2)$, then we can achieve an $\widetilde{O}(1)$-approximation by just returning $E_M \setminus E''$. Thus, we can assume without loss of generality that $\mathsf{OPT} = \widetilde{o}(n^2)$. Given this, we have that the probability that E'' does not cover all minimal anti-hopsets for a thin demand (s,t) is exponentially small. □

□

5.2.3 Proof of Theorem 5.10

Proof. All thick demands can be satisfied by running Algorithm 2 to build E', which has expected cost $O(n \ln n \cdot \sqrt{\mathsf{OPT}})$ (by Lemma 5.11) and runs in polynomial time. The thin demands can be satisfied by running Algorithm 3, which runs in polynomial time. Algorithm 3 fails with exponentially small probability (in which case we return all possible hopset edges, $\widetilde{E}$), and thus the expected cost of E'' is at most $\mathsf{OPT} \cdot 2(n/b)\ln n + o(1) = O(n \ln n \cdot \sqrt{\mathsf{OPT}})$ (Lemma 5.12). Thus the overall approximation ratio is $O(n \ln n / \sqrt{\mathsf{OPT}})$. □

5.3 Trade-Offs with Existential Bounds

There are a number of constructive existential results for hopsets that we trade off with our junction tree-based algorithm from Section 5.1 (an $\widetilde{O}(\beta n^{\epsilon} \cdot \mathsf{OPT})$-approximation) and our star-sampling/randomized-rounding algorithm from Section 5.2 (an $\widetilde{O}(n/\sqrt{\mathsf{OPT}})$-approximation) to give approximations in several regimes. Our junction tree algorithm gives much better approximations than all other existential results when OPT and β are relatively small, so it will be used in the trade off for all regimes. The star-sampling/randomized-rounding algorithm gives improved approximations over the junction tree algorithm as OPT gets larger.

We first state a folklore existential bound that we will use throughout. For exact hopsets in both directed and undirected graphs, the following bound is the best-known (and is known to be tight [BH23]). Exact hopsets satisfy any distance bound function, so we can trade off the hopset produced by the folklore existential bound with other algorithms in any regime. For more restricted regimes—specifically for limited stretch, hopbound, and undirected graphs—we will trade off with stronger existential bounds to get improved approximations.

Lemma 5.16. *Given any weighted directed graph G and a parameter $\beta > 0$, there is an exact hopset of G with hopbound β and size $\widetilde{O}(n^2/\beta^2)$. This hopset can be constructed in polynomial time via random sampling.*

This implies the following straightforward characterization based on OPT, which will allow us to trade the folklore construction off with our other approximation algorithms that also depend on OPT.

Corollary 5.17. *There is a randomized polynomial-time $\widetilde{O}(n^2/(\beta^2 \cdot \mathsf{OPT}))$-approximation for directed* Generalized β-Hopset.

As β gets larger, this folklore construction gives improved approximations over the junction tree and star-sampling/randomized-rounding algorithms, especially when OPT is also relatively large.

5.3.1 Directed Hopsets with Arbitrary Distance Bounds

We trade off our junction-tree and star-sampling/randomized-rounding algorithms with the Corollary 5.17 folklore approximation to achieve an overall $\widetilde{O}(n^{4/5+\epsilon})$-approximation for directed Generalized β-Hopset.

Theorem 5.18. *There is a randomized polynomial-time $\widetilde{O}(n^{4/5+\epsilon})$-approximation for directed* Generalized β-Hopset.

Proof. We trade off the junction tree based algorithm from Section 5.1, the star-sampling/randomized-rounding algorithms from Section 5.2, and the algorithm of Corollary 5.17 to give an algorithm for directed Generalized β-Hopset; that is, we run all three algorithms and return the solution with the minimum cost. The overall approximation ratio of this algorithm, α, is the maximum over all possible β, OPT of the minimum cost:

$$\alpha = \widetilde{O}\left(\max_{\beta,\mathsf{OPT}}\left(\min\left\{\beta n^{\epsilon}\cdot \mathsf{OPT},\ \frac{n}{\sqrt{\mathsf{OPT}}},\ \frac{n^2}{\beta^2\cdot \mathsf{OPT}}\right\}\right)\right).$$

The value of α is $\widetilde{O}(n^{4/5+\epsilon})$, achieved at $\beta = \widetilde{O}(n^{2/5})$, $\mathsf{OPT} = \widetilde{O}(n^{2/5-\epsilon})$ (this is also the point at which all three curves intersect). Additionally, each of the three algorithms runs in polynomial-time, so the overall algorithm is polynomial-time. ☐

When $\beta = \widetilde{O}(n^{2/5})$, the folklore construction gives worse approximations than the trade off between our other algorithms. By just trading off the junction tree and star-sampling/randomized-rounding algorithms, we achieve a better approximation in this regime.

Theorem 5.19. *When $\beta = \widetilde{O}(n^{2/5})$, there is a randomized polynomial-time $\widetilde{O}(\beta^{1/3}\cdot n^{2/3+\epsilon})$-approximation for directed* Generalized β-Hopset.

Corollary 5.20. *When $\beta = \widetilde{O}(1)$, Theorem 5.19 gives an $\widetilde{O}(n^{2/3+\epsilon})$-approximation for* Generalized β-Hopset.

We can also get improved approximations in the large β setting by trading off the junction tree algorithm with the Corollary 5.17 folklore approximation. Note that because the folklore construction has a better inverse dependence on β than the star-sampling/randomized-rounding algorithm (in fact, the latter has *no* dependence on β), the folklore construction performs better than star-sampling/randomized rounding when β is sufficiently large.

Theorem 5.21. *When $\beta = \widetilde{\Omega}(n^{2/5})$, there is a randomized polynomial-time $\widetilde{O}(n^{1+\epsilon}/\sqrt{\beta})$-approximation for directed* Generalized β-Hopset.

5.3.2 Directed Hopsets with Small Stretch

For $(1+\epsilon)$-approximate directed hopsets, the best known existential bound is the following:

Lemma 5.22 (Theorem 1.1 of [BW23]). *For any directed graph G with integer weights in $[1, W]$, given $\epsilon \in (0,1)$ and $\beta \geq 20 \log n$, there is a $(1+\epsilon)$-hopset with hopbound β and size:*

- $\widetilde{O}\left(\frac{n^2 \log^2(nW)}{\beta^3 \epsilon^2}\right)$ *for $\beta \leq n^{1/3}$,*
- $\widetilde{O}\left(\frac{n^{\frac{3}{2}} \log^2(nW)}{\beta^{3/2} \epsilon^2}\right)$ *for $\beta > n^{1/3}$.*

This hopset can be constructed in polynomial time.

Corollary 5.23. *When $\beta \geq 20 \log n$, there is a polynomial-time approximation for directed $(1+\epsilon)$ GENERALIZED β-HOPSET with approximation ratio:*

- $\widetilde{O}\left(\frac{n^2 \log^2(nW)}{\beta^3 \epsilon^2 \cdot \mathsf{OPT}}\right)$ *for $\beta \leq n^{1/3}$,*
- $\widetilde{O}\left(\frac{n^{\frac{3}{2}} \log^2(nW)}{\beta^{3/2} \epsilon^2 \cdot \mathsf{OPT}}\right)$ *for $\beta > n^{1/3}$*

where $\epsilon \in (0,1)$.

Using the Corollary 5.23 algorithm, we get an improved approximation for directed GENERALIZED β-HOPSET when we restrict to $(1+\epsilon)$ stretch.

Theorem 5.24. *When $\beta \geq 20 \log n$ and $\epsilon \in (0,1)$, there is a randomized polynomial-time $\widetilde{O}(n^{3/4+\epsilon'} \cdot \epsilon^{-\frac{1}{4}})$-approximation for directed stretch-$(1+\epsilon)$ GENERALIZED β-HOPSET, where $\epsilon' > 0$ is an arbitrarily small constant.*

Proof. The approximation is achieved by trading off the junction tree algorithm of Section 5.1, the star-sampling/randomized-rounding algorithms of Section 5.2, the Corollary 5.17 folklore algorithm, and the Corollary 5.23 algorithm. Note that $W = \mathit{poly}(n)$, so the $\log(nW)$ factor is hidden by the $\widetilde{O}(\cdot)$ notation. $\square$

5.3.3 Undirected Hopsets with Small Stretch

For *undirected* hopsets with $(1+\epsilon)$ stretch, there is the following constructive existential result.

Lemma 5.25 ([EN19b]). *For any weighted undirected graph G, any integer $\eta \geq 1$, and any $0 < \rho < 1$, there is a randomized algorithm that runs in polynomial time in expectation and computes a hopset with size $O(n^{1+1/\eta})$, which is a hopset for any $\epsilon \in (0,1)$ with hopbound*

$$\beta = \left(\frac{\log(\eta) + 1/\rho}{\epsilon}\right)^{\log(\eta)+\frac{1}{\rho}+1}.$$

Let $W_0(x)$ be the principle branch of the Lambert W function. When $x \geq 3$, the function is upper bounded by $\ln x - (1/2) \ln \ln x$. The Lemma 5.25 existential result implies the following:

Corollary 5.26. *Let $\eta = \lfloor \beta^{1/W_0(\ln \beta)} \rfloor > \beta^{1/(\ln \ln \beta - \frac{1}{2} \ln \ln \ln \beta)}$ (inequality holds when $\beta \geq 3$). There is a randomized polynomial-time $O(n^{1+1/\eta}/\mathsf{OPT})$-approximation for undirected stretch-$(1+\epsilon)$ GENERALIZED β-HOPSET, where $\epsilon \in (0,1)$.*

For some insight into the behavior of η, note first that for all $\beta \geq 3$, $\eta \geq 6$. Additionally, the η function grows faster than $\ln \beta$, but much slower than β. The Corollary 5.26 construction performs better than the star-sampling/randomized-rounding algorithms as OPT grows, resulting in improved approximations compared to the directed graph, arbitrary stretch regime when β is relatively small.

Theorem 5.27. *Let $\eta = \lfloor \beta^{1/W_0(\ln \beta)} \rfloor > \beta^{1/(\ln\ln\beta - \frac{1}{2}\ln\ln\ln\beta)}$ (inequality holds when $\beta \geq 3$). When $\beta = \widetilde{O}(n^{\frac{1}{2}-\frac{1}{2\eta}})$, there is a randomized polynomial-time $\widetilde{O}(\sqrt{\beta} \cdot n^{\frac{1}{2}+\frac{1}{2\eta}+\epsilon'})$-approximation for undirected $(1+\epsilon)$-stretch* Generalized β-Hopset*, where $\epsilon \in (0,1)$, and $\epsilon' > 0$ is an arbitrarily small constant.*

Proof. The approximation is achieved by trading off the junction tree algorithm of Section 5.1 with the Corollary 5.26 algorithm. □

5.3.4 Undirected Hopsets with Odd Stretch

The following result is directly implied by Thorup-Zwick approximate distance oracles.

Lemma 5.28 ([TZ05])**.** *Let $k \geq 1$ be an integer. For any weighted undirected graph G, a stretch-$(2k-1)$ hopset with hopbound 2 and size $O(kn^{1+1/k})$ can be found in polynomial-time.*

Corollary 5.29. *Let $k \geq 1$ be an integer. There is a polynomial-time $O(kn^{1+1/k}/\mathsf{OPT})$-approximation for undirected stretch-$(2k-1)$* Generalized β-Hopset*.*

We can use Corollary 5.29 to get an improved approximation (over the directed graph, arbitrary stretch setting) for undirected stretch-$(2k-1)$ Generalized β-Hopset when β is relatively small.

Theorem 5.30. *Let $k \geq 1$ be an integer. When $\beta = \widetilde{O}(k^{-1/2} \cdot n^{\frac{1}{2}-\frac{1}{2k}})$, there is a polynomial-time $\widetilde{O}(\sqrt{k\beta} \cdot n^{\frac{1}{2}+\frac{1}{2k}+\epsilon})$-approximation for undirected stretch-$(2k-1)$* Generalized β-Hopset*, where $\epsilon > 0$ is an arbitrarily small constant.*

Proof. The approximation is achieved by trading off the junction tree algorithm of Section 5.1 and the Corollary 5.29 algorithm. □

6 Label Cover Hardness for Directed Hopsets and Shortcut Sets

In this section we prove Theorem 1.1. In particular, we prove hardness of approximation for directed shortcut sets on directed graphs, which immediately implies hardness for directed hopsets for any stretch bound as long as the hopbound is at least 3. We use a reduction from Min-Rep (the natural minimization version of Label Cover) in a very similar way to the hardness proofs for graph spanners from [Kor01, EP07, DKR16, CD16]. The main technical difficulty / difference is that these previous reductions for spanners heavily use the fact that only edges from the original graph can be included in the spanner. This is not true of shortcut sets (or hopsets), and makes it simpler to reason about the space of "all feasible spanners" than it is to reason about the space of "all possible shortcut sets".

The Min-Rep problem, first introduced by [Kor01] and since used to prove hardness of approximation for many network design problems, can be thought of as a minimization version of Label Coverwith the additional property that the alphabets are represented explicitly as vertices in a graph. A Min-Rep instance consists of an undirected bipartite graph $G = (A, B, E)$, together with partitions of $A_1, A_2, \ldots, A_m$ of A and $B_1, B_2, \ldots, B_m$ of B with the property that $|A_i| = |A_j| = |B_i| = |B_j$ for every $i, j \in [m]$[2]. Each part of one of the partitions is called a *group*. If there is at least one edge between A_i and B_j, then we say that (i, j) is a *superedge*. Our goal is to choose a set $S \subseteq A \cup B$ of vertices of V if minimum size so that, for every superedge (i, j), there is some vertex $x \in A_i \cap S$ and $y \in B_j \cap S$ with $\{x, y\} \in E$. In other words, we must choose vertices to *cover* each superedge. Any such cover is called a REP-cover, and our goal is to find the minimum size REP-cover. Note that we must choose at least one vertex from every group (assuming that each group is involved in at least one superedge), and hence $OPT \geq 2m$.

Since Min-Rep is essentially Label Cover, which is a two-query PCP, the PCP theorem implies hardness of approximation for Min-Rep. More formally, the following hardness is known [Kor01].

[2]In some versions of the problem we do not assume that each partition has the same number of parts or that each $|A_i| = |B_i|$, but those assumptions can both be made without loss of generality so we do so for convenience

Theorem 6.1 ([Kor01]). *Assuming that $NP \not\subseteq DTIME(2^{polylog(n)})$, then for any constant $\epsilon > 0$ there is no polynomial-time algorithm that approximates Min-Rep better than $2^{\log^{1-\epsilon} n}$. In particular, no polynomial time algorithm can distinguish between when $OPT = 2m$ and when $OPT \geq 2m \cdot 2^{\log^{1-\epsilon} n}$*

6.1 Reduction from Min-Rep to shortcut sets

We can now give our formal reduction from Min-Rep to shortcut sets with hopbound $\beta \geq 3$. Consider some instance of Min-Rep $G = (A, B, E)$ with partition $A_1, A_2, \ldots, A_m$ and $B_1, B_2, \ldots, B_m$. First we create two nodes for every group: let

$$A' = \{a_1^1, a_1^2, a_2^1, a_2^2, \ldots, a_m^1, a_m^2\} \qquad \text{and} \qquad B' = \{b_1^1, b_1^2, b_2^1, b_2^2, \ldots, b_m^1, b_m^2\}.$$

Then for every edge $e = \{a, b\}$, we create a set of $\beta - 3$ vertices $V_e = \{v_e^1, v_e^2, \ldots, v_e^{\beta-3}\}$ (note that these sets are empty if $\beta = 3$). Our final vertex set is

$$V' = V \cup A' \cup B' \cup (\cup_{e \in E} V_e).$$

We now create the (directed) edges of the graph. For each $e = \{a, b\} \in E$ we create a path:

$$P_e = \{(a, v_e^1)\} \cup \{(v_e^i, v_e^{i+1}) : i \in [\beta - 4]\} \cup \{(b, v_e^{\beta-3})\}.$$

If $\beta = 3$, we simply set $P_e = \{(a, b)\}$. We can now specify our final edge set:

$$\begin{aligned} E' = &(\cup_{e \in E} P_e) \\ &\cup \{(a_i^1, a_i^2) : i \in [m]\} \cup \{(b_i^2, b_i^1) : i \in [m]\} \\ &\cup \{(a_i^2, x) : i \in [m], x \in A_i\} \cup \{(x, b_i^2) : i \in [m], x \in B_i\}. \end{aligned}$$

Our final shortcut set instance is $G' = (V', E')$ with hopbound β.

6.2 Analysis

The analysis of this reduction has two parts, completeness and soundness (so-called due to their connections back to probabilistically checkable proofs). For completeness, we show that if the original Min-Rep instance has $OPT = \gamma$ then there is a shortcut set of size at most γ. For soundness, we show that if there is a shortcut set of size γ, then there is a REP-cover of size at most $O(\gamma)$. Together with Theorem 6.1, these will imply our desired hardness bound for shortcut sets. To get Theorem 1.1, one just needs to observe that in G', if there is a path from some node u to some node v then in fact *every* path from u to v has exactly the same length. Hence any shortcut set is a hopset with the same hopbound and stretch 1, and any hopset with stretch α is also shortcut set.

6.2.1 Completeness

We begin with completeness, which is the easier direction. We prove the following lemma.

Lemma 6.2. *If there is a REP-cover of the Min-Rep instance with size γ, then there is a shortcut set for G' with hopbound β and size γ.*

Proof. Suppose that there is a REP-cover $S \subseteq V$ with $|S| = \gamma$. Consider the edge set

$$S' = \{(a_i^1, x) : i \in [m], x \in S \cap A_i\} \cup \{(x, b_i^1) : i \in [m], x \in S \cap B_i\}.$$

In other words, we add an edge between each "outer" node representing a group and the nodes in the REP-cover that are in that group (directed oppositely for the left and right sides of the bipartite graph). Clearly $|S'| = |S| = \gamma$, so it remains to show that S' is a shortcut set with hopbound β.

An easy observation is that in G', the only nodes x, y where y is reachable from x but there are more than β hops on every $x \rightsquigarrow y$ path are when $x = a_i^1$ and $y = b_j^1$ for some $i, j \in [m]$ where (i, j) is a superedge. So consider such an a_i^1, b_j^1. Then since (i, j) is a superedge and S is a REP-cover, we know that there is some $a, b \in S$ with $a \in A_i$ and $b \in B_j$ and $\{a, b\} \in E$. Then by definition S' includes the edges (a_i^1, a) and (b, b_j^1). Thus there is a path from a_i^1 to b_j^1 which first uses the edge to a added by S, then uses the edges of $P_{\{a,b\}}$ to get to b, and then uses the edge to b_j^1 added by S. This path has exactly β hops by construction. Hence S' is a shortcut set with hopbound β. □

6.2.2 Soundness

We now argue about soundness. To do this, we first need the following definition.

Definition 9. A shortcut edge is called *canonical* if it is of the form (a_i^1, a) for some $i \in [m]$ and $a \in A_i$ or of the form (b, b_i^1) for some $i \in [m]$ and $b \in B_i$. A shortcut set S' is called *canonical* if every edge in S' is canonical.

We now show that without loss of generality, we may assume that any shortcut set is canonical.

Lemma 6.3. *Suppose that S is a shortcut set of G' with hopbound β and size γ. Then there is a canonical shortcut set S' of G' with hopbound β and size at most 2γ.*

Proof. Consider some $(x, y) \in S$ which is not canonical. Since it is not canonical but also must exist in the transitive closure of G', there must exist a super edge (i, j) so that x and y are as follows.

- x is either a_i^1, a_i^2, is in A_i, or is in P_e for some $e = \{a, b\}$ with $a \in A_i$ and $b \in B_j$.
- y is either b_j^1, b_j^2, is in B_j, or is in P_e for some $e = \{a, b\}$ with $a \in A_i$ and $b \in B_j$.

In particular, this implies that (x, y) is only useful in decreasing the number of hops to at most β for the pair (a_i^1, b_j^1). In other words, if we *removed* (x, y) from S, then the only pair of nodes which be reachable but not within β hops would be (a_i^1, b_j^1). So choose some arbitrary edge $\{a, b\} \in E$ with $a \in A_i$ and $b \in B_j$, and let (a_i^1, a) and (b, b_j^1) be the *replacement edges* for (x, y). Note that these replacement edges are both canonical, and suffice to decrease the number of hops from a_i^1 to b_j^1 to β.

So we let S' consist of all of the canonical edges of S together with the two replacement edges for every noncanonical edge in S. Clearly $|S'| \leq 2|S| = 2\gamma$, and S' is a shortcut set with hopbound β. □

With this lemma in hand, it is now relatively straighforward to prove soundness.

Lemma 6.4. *If there is a shortcut set for G' with hopbound β and size γ, then there is a REP-cover of the Min-Rep instance with size at most 2γ.*

Proof. Let $\hat{S}$ be a shortcut set for G' with hopbound β and size γ. By Lemma 6.3, there is a shortcut set S' for G' with hopbound β and size at most 2γ. Let S be the set of nodes such that they are an endpoint of some edge in S' and are in $A \cup B$. In other words, for every canonical edge $(a_i^1, a) \in S'$ we add a to S, and similarly for every canonical edge $(b, b_i^1) \in S'$ we add b to S. Clearly $|S| \leq |S'| \leq 2\gamma$, so we just need to prove that S is a REP-cover.

Consider some superedge (i, j). Then G', we know that a_i^1 can reach b_j^1 but every such path has length $\beta + 2$. Since S' is a canonical shortcut set, this means that it must add an edge (a_i^1, a) and an edge (b, b_j^1) for some $a \in A_i$ and $b \in B_i$ with $(a, b) \in E$. Hence $a, b \in S$ by definition. Thus S is a REP-cover as claimed. □

References

[AB24] Amir Abboud and Greg Bodwin. Reachability preservers: New extremal bounds and approximation algorithms. *SIAM Journal on Computing*, 53(2):221–246, 2024.

[ABP18] Amir Abboud, Greg Bodwin, and Seth Pettie. A hierarchy of lower bounds for sublinear additive spanners. *SIAM Journal on Computing*, 47(6):2203–2236, 2018.

[ADD+93] Ingo Althöfer, Gautam Das, David P. Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discret. Comput. Geom.*, 9:81–100, 1993.

[BBM+11] Piotr Berman, Arnab Bhattacharyya, Konstantin Makarychev, Sofya Raskhodnikova, and Grigory Yaroslavtsev. Improved approximation for the directed spanner problem. In *Proceedings of the 38th International Colloquim Conference on Automata, Languages and Programming - Volume Part I*, ICALP'11, page 1–12, Berlin, Heidelberg, 2011. Springer-Verlag.

[BGGN16] Maxim Babenko, Andrew V. Goldberg, Anupam Gupta, and Viswanath Nagarajan. Algorithms for hub label optimization. *ACM Trans. Algorithms*, 13(1), November 2016.

[BGJ+08] Arnab Bhattacharyya, Elena Grigorescu, Kyomin Jung, Sofya Raskhodnikova, and David P. Woodruff. Transitive-closure spanners, 2008.

[BH23] Greg Bodwin and Gary Hoppenworth. Folklore sampling is optimal for exact hopsets: Confirming the $\sqrt{n}$ barrier. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 701–720. IEEE, 2023.

[BHT23] Greg Bodwin, Gary Hoppenworth, and Ohad Trabelsi. Bridge girth: A unifying notion in network design. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 600–648. IEEE, 2023.

[BLP20] Uri Ben-Levy and Merav Parter. New (α, β) spanners and hopsets. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1695–1714. SIAM, 2020.

[BPGS21] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental sssp and approximate min-cost flow in almost-linear time. In *62 Annual IEEE Symposium on Foundatios of Computer Science (FOCS 2022)*, 2021.

[BR11] Aaron Bernstein and Liam Roditty. Improved dynamic algorithms for maintaining approximate shortest paths under deletions. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1355–1365. SIAM, 2011.

[BW23] Aaron Bernstein and Nicole Wein. Closing the gap between directed hopsets and shortcut sets. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 163–182. SIAM, 2023.

[CD16] Eden Chlamtác and Michael Dinitz. Lowest-degree k-spanner: Approximation and hardness. *Theory Comput.*, 12(1):1–29, 2016.

[CDK12] Eden Chlamtac, Michael Dinitz, and Robert Krauthgamer. Everywhere-sparse spanners via dense subgraphs. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 758–767. IEEE Computer Society, 2012.

[CDKL20] Eden Chlamtác, Michael Dinitz, Guy Kortsarz, and Bundit Laekhanukit. Approximating spanners and directed steiner forest: Upper and lower bounds. *ACM Trans. Algorithms*, 16(3):33:1–33:31, 2020.

[CDR19] Eden Chlamtác, Michael Dinitz, and Thomas Robinson. Approximating the norms of graph spanners. In Dimitris Achlioptas and László A. Végh, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2019, September 20-22, 2019, Massachusetts Institute of Technology, Cambridge, MA, USA*, volume 145 of *LIPIcs*, pages 11:1–11:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[CEGS11] Chandra Chekuri, Guy Even, Anupam Gupta, and Danny Segev. Set connectivity problems in undirected graphs and the directed steiner network problem. *ACM Trans. Algorithms*, 7(2), March 2011.

[CF23] Nairen Cao and Jeremy T Fineman. Parallel exact shortest paths in almost linear work and square root depth. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4354–4372. SIAM, 2023.

[CFG+24] Emilio Cruciani, Sebastian Forster, Gramoz Goranci, Yasamin Nazari, and Antonis Skarlatos. Dynamic algorithms for k-center on graphs. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3441–3462. SIAM, 2024.

[CFR20a] Nairen Cao, Jeremy T Fineman, and Katina Russell. Efficient construction of directed hopsets and parallel approximate shortest paths. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 336–349, 2020.

[CFR20b] Nairen Cao, Jeremy T Fineman, and Katina Russell. Improved work span tradeoff for single source reachability and approximate shortest paths. In *ACM Symposium on Parallelism in Algorithms and Architectures*, 2020.

[CHDKL21] Keren Censor-Hillel, Michal Dory, Janne H Korhonen, and Dean Leitersdorf. Fast approximate shortest paths in the congested clique. *Distributed Computing*, 34:463–487, 2021.

[Che18] Shiri Chechik. Near-optimal approximate decremental all pairs shortest paths. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 170–181. IEEE, 2018.

[CHKZ03] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, 32(5):1338–1355, 2003.

[CJMN25] Parinya Chalermsook, Yonggang Jiang, Sagnik Mukhopadhyay, and Danupon Nanongkai. Shortcuts and transitive-closure spanners approximation, 2025.

[CKL+22] Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623. IEEE, 2022.

[Coh00] Edith Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *Journal of the ACM (JACM)*, 2000.

[DK11] Michael Dinitz and Robert Krauthgamer. Directed spanners via flow-based linear programs. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, STOC '11, page 323–332, New York, NY, USA, 2011. Association for Computing Machinery.

[DKR16] 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):25:1–25:16, 2016.

[DM24] Michal Dory and Shaked Matar. Massively parallel algorithms for approximate shortest paths. In *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '24, page 415–426, New York, NY, USA, 2024. Association for Computing Machinery.

[DN19] Michael Dinitz and Yasamin Nazari. Massively parallel approximate distance sketches. *OPODIS*, 2019.

[DNZ20] Michael Dinitz, Yasamin Nazari, and Zeyu Zhang. Lasserre integrality gaps for graph spanners and related problems. In *Approximation and Online Algorithms: 18th International Workshop, WAOA 2020, Virtual Event, September 9–10, 2020, Revised Selected Papers*, page 97–112, Berlin, Heidelberg, 2020. Springer-Verlag.

[DZ10] Erik D Demaine and Morteza Zadimoghaddam. Minimizing the diameter of a network using shortcut edges. In *Scandinavian Workshop on Algorithm Theory*, pages 420–431. Springer, 2010.

[EN18] Michael Elkin and Ofer Neiman. Near-optimal distributed routing with low memory. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*. ACM, 2018.

[EN19a] Michael Elkin and Ofer Neiman. Hopsets with constant hopbound, and applications to approximate shortest paths. *SIAM Journal on Computing*, 2019.

[EN19b] Michael Elkin and Ofer Neiman. Linear-size hopsets with small hopbound, and constant-hopbound hopsets in rnc. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, pages 333–341, 2019.

[EN20] Michael Elkin and Ofer Neiman. Near-additive spanners and near-exact hopsets, a unified view. *arXiv preprint arXiv:2001.07477*, 2020.

[EP07] Michael Elkin and David Peleg. The hardness of approximating spanner problems. *Theory Comput. Syst.*, 41(4):691–729, 2007.

[Fin18] Jeremy T Fineman. Nearly work-efficient parallel algorithm for digraph reachability. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 457–470, 2018.

[FKN12] Moran Feldman, Guy Kortsarz, and Zeev Nutov. Improved approximation algorithms for directed steiner forest. *Journal of Computer and System Sciences*, 78(1):279–292, 2012.

[GKL23] Elena Grigorescu, Nithish Kumar, and Young-San Lin. Approximation Algorithms for Directed Weighted Spanners. In Nicole Megow and Adam Smith, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2023)*, volume 275 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:23, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[GKL24a] Elena Grigorescu, Nithish Kumar, and Young-San Lin. Directed buy-at-bulk spanners, 2024.

[GKL24b] Elena Grigorescu, Nithish Kumar, and Young-San Lin. Multicriteria spanners – a new tool for network design, 2024.

[GLQ21] Elena Grigorescu, Young-San Lin, and Kent Quanrud. Online directed spanners and steiner forests. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, August 16-18, 2021, University of Washington, Seattle, Washington, USA (Virtual Conference)*, volume 207 of *LIPIcs*, pages 5:1–5:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[GWN20] Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Deterministic algorithms for decremental approximate shortest paths: Faster and simpler. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2522–2541. SIAM, 2020.

[Has92] Refael Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations Research*, 17(1):36–42, 1992.

[Hes03] William Hesse. Directed graphs requiring large numbers of shortcuts. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 665–669, 2003.

[HK18] Markó Horváth and Tamás Kis. Multi-criteria approximation schemes for the resource constrained shortest path problem. *Optimization Letters*, 12(3):475–483, 2018.

[HKN14] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Decremental single-source shortest paths on undirected graphs in near-linear total update time. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 146–155. IEEE, 2014.

[HP19] Shang-En Huang and Seth Pettie. Thorup–zwick emulators are universally optimal hopsets. *Information Processing Letters*, 142:9–13, 2019.

[HP21] Shang-En Huang and Seth Pettie. Lower bounds on sparse spanners, emulators, and diameter-reducing shortcuts. *SIAM Journal on Discrete Mathematics*, 35(3):2129–2144, 2021.

[HXX25] Gary Hoppenworth, Yinzhan Xu, and Zixuan Xu. New separations and reductions for directed hopsets and preservers. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4405–4443. SIAM, 2025.

[JLS19] Arun Jambulapati, Yang P Liu, and Aaron Sidford. Parallel reachability in almost linear work and square root depth. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1664–1686. IEEE, 2019.

[Kor01] Guy Kortsarz. On the hardness of approximating spanners. *Algorithmica*, 30(3):432–450, 2001.

[KP94] Guy Kortsarz and David Peleg. Generating sparse 2-spanners. *J. Algorithms*, 17(2):222–236, 1994.

[KP98] Guy Kortsarz and David Peleg. Generating low-degree 2-spanners. *SIAM J. Comput.*, 27(5):1438–1456, 1998.

[KP22a] Shimon Kogan and Merav Parter. Having hope in hops: New spanners, preservers and lower bounds for hopsets. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 766–777. IEEE, 2022.

[KP22b] Shimon Kogan and Merav Parter. New diameter-reducing shortcuts and directed hopsets: Breaking the barrier. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1326–1341. SIAM, 2022.

[KP23] Shimon Kogan and Merav Parter. Towards bypassing lower bounds for graph shortcuts. In *31st Annual European Symposium on Algorithms (ESA 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

[KS97] Philip N Klein and Sairam Subramanian. A randomized parallel algorithm for single-source shortest paths. *Journal of Algorithms*, 1997.

[ŁN22] Jakub Łącki and Yasamin Nazari. Near-optimal decremental hopsets with applications. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[LR01] Dean H. Lorenz and Danny Raz. A simple efficient approximation scheme for the restricted shortest path problem. *Oper. Res. Lett.*, 28(5):213–219, June 2001.

[Mad10] Aleksander Madry. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 121–130, 2010.

[MPVX15] Gary L Miller, Richard Peng, Adrian Vladu, and Shen Chen Xu. Improved parallel algorithms for spanners and hopsets. In *Proceedings of the Symposium on Parallelism in Algorithms and Architectures*. ACM, 2015.

[Phi93] Cynthia A. Phillips. The network inhibition problem. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, page 776–785, New York, NY, USA, 1993. Association for Computing Machinery.

[TZ05] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, January 2005.

[UY90] Jeffery Ullman and Mihalis Yannakakis. High-probability parallel transitive closure algorithms. In *Proceedings of the second annual ACM symposium on Parallel algorithms and architectures*, pages 200–209, 1990.

[War87] Arthur Warburton. Approximation of pareto optima in multiple-objective, shortest-path problems. *Oper. Res.*, 35:70–79, 1987.

[WXX24] Virginia Vassilevska Williams, Yinzhan Xu, and Zixuan Xu. Simpler and higher lower bounds for shortcut sets. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2643–2656. SIAM, 2024.

[XZTT08] Guoliang Xue, Weiyi Zhang, Jian Tang, and Krishnaiyan Thulasiraman. Polynomial time approximation algorithms for multi-constrained qos routing. *IEEE/ACM Transactions on Networking*, 16(3):656–669, 2008.