$O(\log^2 k / \log\log k)$-Approximation Algorithm for Directed Steiner Tree: A Tight Quasi-Polynomial-Time Algorithm.

Fabrizio Grandoni [*] Bundit Laekhanukit [†] Shi Li [‡]

November 8, 2018

Abstract

In the Directed Steiner Tree (DST) problem we are given an n-vertex directed edge-weighted graph, a root r, and a collection of k terminal nodes. Our goal is to find a minimum-cost arborescence that contains a directed path from r to every terminal. We present an $O(\log^2 k / \log\log k)$-approximation algorithm for DST that runs in quasi-polynomial-time, i.e., in time $n^{\text{poly}\log(k)}$. By assuming the Projection Game Conjecture and NP $\not\subseteq \bigcap_{0<\epsilon<1}$ZPTIME$(2^{n^\epsilon})$, and adjusting the parameters in the hardness result of Halperin and Krauthgamer [STOC'03], we show the matching lower bound of $\Omega(\log^2 k / \log\log k)$ for the class of quasi-polynomial-time algorithms, meaning that our approximation ratio is asymptotically the best possible. This is the first improvement on the DST problem since the classical quasi-polynomial-time $O(\log^3 k)$ approximation algorithm by Charikar et al. [SODA'98 & J. Algorithms'99]. (The paper erroneously claims an $O(\log^2 k)$ approximation due to a mistake in prior work.)

Our approach is based on two main ingredients. First, we derive an approximation preserving reduction to the *Label-Consistent Subtree (LCST)* problem. Here we are given a rooted tree with node labels, and a feasible solution is a subtree satisfying proper constraints on the labels. The LCST instance has quasi-polynomial size and logarithmic height. We remark that, in contrast, Zelikovsky's heigh-reduction theorem [Algorithmica'97] used in all prior work on DST achieves a reduction to a tree instance of the related Group Steiner Tree (GST) problem of similar height, however losing a logarithmic factor in the approximation ratio.

Our second ingredient is an LP-rounding algorithm to approximately solve LCST instances, which is inspired by the framework developed by [Rothvoß, Preprint'11; Friggstad et al., IPCO'14]. We consider a Sherali-Adams lifting of a proper LP relaxation of LCST. Our rounding algorithm proceeds level by level from the root to the leaves, rounding and conditioning each time on a proper subset of *label* variables. The limited height of the tree and small number of labels on root-to-leaf paths guarantees that a small enough (namely, polylogarithmic) number of Sherali-Adams lifting levels is sufficient to condition up to the leaves.

We believe that our basic strategy of combining label-based reductions with a round-and-condition type of LP-rounding over hierarchies might find applications to other related problems.

[*] IDSIA, USI-SUPSI, E-mail: fabrizio@idsia.ch.

[†] Institute for Theoretical Computer Science, Shanghai University of Finance and Economics. E-mail: bundit@sufe.edu.cn.

[‡] Department of Computer Science and Engineering, University at Buffalo. E-mail: shil@buffalo.edu.

1 Introduction

In the *Directed Steiner Tree (DST)* problem, we are given an n-vertex digraph $G = (V, E)$ with cost c_e on each edge $e \in E$, a root vertex $r \in V$ and a set of k terminals $K \subseteq V \setminus \{r\}$. The goal is to find a minimum-cost out-arborescence $H \subseteq G$ rooted at r that contains an $r \to t$ directed path for every terminal $t \in K$. W.l.o.g. we assume that edge costs satisfy triangle inequality.

The DST problem is a fundamental problem in the area of network design that is known for its bizarre behaviors. While constant-approximation algorithms have been known for its undirected counterpart (see, e.g., [3, 29, 31]), the best known polynomial-time approximation algorithm for this problem could achieve only an $O((1/\epsilon)^3 k^\epsilon)$ approximation ratio in time $O(n^{1/\epsilon})$ for any $0 < \epsilon \leq 1/\log_2 k$, due to the classical work of Charikar et al. [5]. Even allowing this algorithm to run in quasi-polynomial-time, the best approximation ratio remains $O(\log^3 k)$ [5][1]. Since then, there have been efforts to get improvements either in the running-time or in the approximation guarantee of this problem, e.g, using the the primal-dual method [33], Sum-of-Squares (a.k.a. Lasserre) hierarchy [30], Sherali-Adams and Lovász-Schrijver hierarchies [12]. Despite all these efforts, there has been no significant improvement over the course of the last two decades for both polynomial and quasi-polynomial time algorithms. In fact, it is known from the work of Halperin and Krauthgamer [17] that unless NP $\subseteq$ ZPTIME$(n^{\text{polylog}(n)})$, it is not possible to achieve an approximation ratio $O(\log^{2-\epsilon} k)$, for any constant $\epsilon > 0$, and such lower bound applies to both polynomial and quasi-polynomial time algorithms. This means that there is a huge gap between the upper bound of k^ϵ and the lower bound of $\log^{2-\epsilon} k$ for polynomial-time algorithms. All efforts were failed to obtain even an $n^{o(1)}$-approximation algorithm that runs in polynomial-time.

For the class of quasi-polynomial-time algorithms, the approximation ratio of $O(\log^3 k)$ is arguably disappointing. This is because its closely related special case, namely, the *Group Steiner Tree* (GST) problem, is known to admit a quasi-polynomial-time $O(\log^2 k)$-approximation algorithm on general graphs due to the work of Chekuri and Pal [6]. A natural question would be whether such an approximation ratio could be achieved in quasi-polynomial-time for DST as well. Nevertheless, achieving this improvement with the known techniques seems to be impossible. Indeed, all previous algorithms for DST [5, 30, 12] rely on the well-known Zelikovsky's height-reduction theorem [32, 19]. These algorithms (implicitly) reduce DST to GST on trees, which loses an $\Theta(\log k)$ approximation factor in the process. Furthermore, the $\Omega(\log^{2-\epsilon} k)$-hardness of Halperin and Krauthgamer [17] carries over to GST on trees. We remark that algorithms for many related problems (see, e.g., [10, 15]) rely on the same height-reduction theorem.

1.1 Our Results and Techniques

The purpose of this work is to close the gap between the lower and upper bounds on the approximability of DST in quasi-polynomial time. Our main result is as follows.

Theorem 1.1. *There is a randomized $O(\log^2 k / \log\log k)$-approximation algorithm for DST with running time $n^{O(\log^5 k)}$.*

By analyzing the proofs in [17], we also show that this bound is asymptotically tight under stronger assumptions; please see more discussion in Appendix C.

[1] The original paper claims an $O(\log^2 k)$-approximation algorithm; however, their result was based on the initial statement of the Zelikovsky's height-reduction theorem in [32], which was later found to contain a subtle flaw and was restated by Helvig, Robin and Zelikovsky [19].

Theorem 1.2. *There is no quasi-polynomial-time algorithm for DST that achieves an approximation ratio* $o(\log^2 k/\log\log k)$ *unless* NP $\subseteq \bigcap_{0<\epsilon<1}$ZPTIME$(2^{n^\epsilon})$ *or the* Projection Game Conjecture *is false.*

Our upper bound is based on two main ingredients. The first one is a quasi-polynomial-time approximation-preserving reduction to a novel *Label-Consistent Subtree* (LCST) problem. Roughly speaking, in LCST we are given a rooted tree plus node labels of two types, global and local. A feasible solution consists of a subtree that satisfies proper constraints on the labels. Intuitively, local labels are used to guarantee that a feasible solution induces an arborescence rooted at r in the original problem, while global labels are used to enforce that all the terminals are included in such arborescence. In our reduction the tree has size $n^{\text{poly}\log(k)}$ and height $h = O(\log k/\log\log k)$, with k global labels. For a comparison, Zelikovsky's height-reduction theorem [32], used in all prior work on DST, reduces (implicitly) the latter problem to a GST instance over a tree of height $O(\log k)$. However, this reduction alone loses a factor $\Theta(\log k)$ in the approximation (while our reduction is approximation-preserving).

Our second ingredient is a quasi-polynomial-time $O(\log^2 k/\log\log k)$-approximate LP-rounding algorithm for LCST instances arising from the previous reduction. Here we exploit the LP-hierarchy framework developed by Rothvoß [30] (and later simplified by Friggstad et al. [12]). We define a proper LP relaxation for the problem, and solve an R-level Sherali-Adams lifting of this LP for a parameter $R = \text{poly}\log k$. We then round the resulting fractional solution level by level from the root to the leaves. At each level we maintain a small set of labels that must be provided by the subtree. By randomly rounding label-based variables and conditioning, we push the set of labels all the way down to the leaves, guaranteeing that the output tree is always label-consistent. Thanks to the limited height of the tree and to the small number of labels along root-to-leaf paths, a polylogarithmic number of lifting levels is sufficient to perform the mentioned conditioning up to the leaves. As in [30], the probability that each global label appears in the tree we directly construct is only $1/(h+1)$. We need to repeat the process $O(h\log k) = O(\log^2 k/\log\log k)$ times in order to make sure all labels are included with high probability, leading to the claimed approximation ratio. Our result gives one more application of using LP/SDP hierarchies to obtain improved approximation algorithms, in addition to a few other ones (see, e.g., [2, 8, 9, 25, 14]).

We believe that our basic strategy of combining a label-based reduction with a *round-and-condition* rounding strategy as mentioned above might find applications to other problems, and it might therefore be of independent interest.

1.2 Comparison to Previous Work

Our algorithm is inspired by two results. First is the recursive greedy algortihm of Chekuri and Pal for GST [6], and second is the hierrachical based LP-rounding techniques by Rothvoß [30].

As mentioned, the algorithm of Chekuri and Pal is the first one that yields an approximation ratio of $O(\log^2 k)$ for GST, which is a special case of DST, in quasi-polynomial-time. This is almost tight for the class of quasi-polynomial-time algorithms. Their algorithm exploits the fact that any optimal solution can be shortcut into a path of length k, while paying only a factor of 2 (such path exists in the metric-closure of the input graph). This simple observation allows them to derive a recursive greedy algorithm. In more detail, they try to identify a vertex that separates the optimal path into two equal-size subpaths by iterating over all the vertices; then they recursively (and approximately) solve two subproblems and pick the best approximate sub-solution greedily.

Their analysis, however, requires the fact that both recursive calls end at the same depth (because each subpath has length different by at most one).

We imitate the recursive greedy algorithm by recursively splitting the optimal solution via balanced tree separators. The same approach as in [6], unfortunately, does not quite work out for us since subproblem sizes may differ by a multiplicative factor. This process, somehow, gives us a decision tree that contains a branch-decomposition of every solution, which is sufficient to devise an approximation algorithm. Note, however, that not every subtree of this decision tree can be transformed into a connected graph, and thus, it is not guaranteed that we can find a feasible DST solution from this decision tree. We introduce node-labels and label-consistent constraints specifically to solve this issue.

The label-consistency requirement could not be handled simply by applying DST algorithms as a blackbox. This comes to the second component that is inspired by the framework developed by Rothvoß [30]. While the framework was originally developed for the Sum-of-Squares hierarchy, it was shown by Friggstad et al. [12] that it also applies to Sherali-Adams, which is a weaker hierarchy. We apply the framework of Rothvoß to our Sherali-Adams lifted-LP but taking the label-consistency requirement into account.

1.3 Related Work

We already mentioned some of the main results about DST and GST. For GST there is a polynomial-time algorithm by Garg et al. [13] that achieves an approximation factor of $O(\log^2 k \log n)$, where k is the number of groups. Their algorithm first maps the input instance into a tree instance by invoking the *Probabilistic Metric-Tree Embeddings* [1, 11], thus losing a factor $O(\log n)$ in the approximation ratio. They then apply an elegant LP-based randomized rounding algorithm to the instance on a tree. A well-known open problem is whether it is possible to avoid the $\log n$ factor in the approximation ratio. This was later achieved by Chekuri and Pal [6], however their algorithm runs in quasi-polynomial-time.

Some works were devoted to the *survivable network* variants of DST and GST, namely ℓ-DST and ℓ-GST, respectively. Here one requires to have ℓ edge-disjoint directed (resp., undirected) paths from the root to each terminal (resp., group). Cheriyan et al. [7] showed that ℓ-DST admits no $2^{\log^{1-\varepsilon} n}$-approximation algorithm, for any $\varepsilon > 0$, unless NP $\subseteq$ DTIME$(2^{\text{polylog}(n)})$. Laekhanukit [23] showed that the problem admits no $\ell^{1/2-\varepsilon}$-approximation for any constant $\varepsilon > 0$, unless NP $=$ ZPP. Nevertheless, the negative results do not rule out the possibility of achieving reasonable approximation factors for small values of ℓ. In particular, Grandoni and Laekhanukit [15] (exploiting some ideas in [24]) recently devised a poly-logarithmic approximation algorithm for 2-DST that runs in quasi-polynomial time.

Concerning ℓ-GST, Gupta et al. [16] presented a $\tilde{O}(\log^3 n \log k)$-approximation algorithm for 2-GST. The same problem admits an $O(\alpha \log^2 n)$-approximation algorithm, where α is the largest cardinality of a group [21]. Chalermsook et al. [4] presented an LP-rounding bicriteria approximation algorithm for ℓ-GST that returns a subgraph with cost $O(\log^2 n \log k)$ times the optimum while guaranteeing a connectivity of at least $\Omega(\ell / \log n)$. They also showed that ℓ-GST is hard to approximate to within a factor of ℓ^σ, for some fixed constant $\sigma > 0$, and if ℓ is large enough, then the problem is at least as hard as the *Label-Cover* problem, meaning that ℓ-GST admits no $2^{\log^{1-\varepsilon} n}$-approximation algorithm, for any constant $\varepsilon > 0$, unless NP $\subseteq$ DTIME$(2^{\text{polylog}(n)})$.

2 Preliminaries

Given a graph G', we denote by $V(G')$ and $E(G')$ the vertex and edge set of G', respectively. Throughout this paper, we treat a rooted tree as an out-arborescence; that is, edges are directed towards the leaves. Given a rooted tree T, we use $\text{root}(T)$ to denote its root. For any rooted tree T and $v \in V(T)$, we shall use $T[v]$ to denote the sub-tree of T containing v and all descendants of v. For a directed edge $e = (u, v)$, we use $\text{head}(e) = u$ and $\text{tail}(e) = v$ to denote the head and tail of e. Generally, we will use the term *vertex* to mean a vertex of a DST instance, and we will use the term *node* to mean a vertex in an instance of the Label-Consistent Subtree problem, defined below:

Label-Consistent Subtree (LCST). The new problem we introduce is the Label-Consistent Subtree (LCST) problem. The input consists of a rooted tree T^0 of size $N = |V(T^0)|$ and height h, a node cost vector $c \in \mathbb{R}_{\geq 0}^{V(T^0)}$, and a set L of labels, among which there are k *global labels* $K \subseteq L$. The other labels $L \setminus K$ are called *local labels*. Each node $v \in V(T^0)$ has two label sets: a set $\text{dem}(v) \subseteq L \setminus K$ of *demand labels*, and a set $\text{ser}(v) \subseteq L$ of *service labels*.

We say that a subtree T of T^0 with $\text{root}(T) = \text{root}(T^0)$ is *label-consistent* if for every vertex $u \in V(T)$ and $\ell \in \text{dem}(u)$, there is a descendant v of u in T such that $\ell \in \text{ser}(v)$. The goal of the LCST problem is to find a label-consistent subtree T of T^0 of minimum cost that contains all global labels, i.e, for every $\ell \in K$, there is a $v \in V(T)$ with $\ell \in \text{ser}(v)$.

In Section 4, we give an $(shN)^{O(sh^2)}$-time $O(h \log k)$-approximation algorithm for the LCST problem, where $s = \max_{v \in V(T^0)} |\text{dem}(v)|$. Thus, we require s to be small in order to derive a quasi-polynomial-time algorithm; fortunately, this is the case for the instance reduced from DST.

One may generalize LCSs to general graphs, say *Label-Consistent Steiner Subgraph* (LCSS).

Balanced Tree Partition. A main tool in our reduction is the following standard balanced-tree-partition lemma (with proof given in Appendix A for completeness).

Lemma 2.1 (Balanced-Tree-Partition)**.** *For any $n \geq 3$, for any n-vertex tree T rooted at a vertex r, there exists a vertex $v \in V(T)$ such that T can be decomposed into two trees T_1 and T_2 rooted at r and v, respectively, in such a way that $E(T_1) \uplus E(T_2) = E(T)$, $V(T_1) \cup V(T_2) = V(T)$ and $V(T_1) \cap V(T_2) = \{v\}$ and $|V(T_1)|, |V(T_2)| < 2n/3 + 1$. In other words, T_1 and T_2 are sub-trees that form a balanced partition of (the edges of) T.*

Sherali-Adams Hierarchy. In this section, we give some basic facts about Sherali-Adams hierarchy that we will need. Assume we have a linear program polytope $\mathcal{P}$ defined by $Ax \leq b$. We assume that $0 \leq x_i \leq 1, \forall i \in [n]$ are part of the linear constraints. The set of integral feasible solutions is defined as $\mathcal{X} = \{x \in \{0,1\}^n : Ax \leq b\}$. It is convenient to think of each $i \in [n]$ as an event, and in a solution $x \in \{0,1\}^n$, x_i indicates whether the event i happens or not.

The idea of Sherali-Adams hierarchy is to strengthen the original LP $Ax \leq b$ by adding more variables and constraints. Of course, each $x \in \mathcal{X}$ should still be a feasible solution to the strengthened LP (when extended to a vector in the higher-dimensional space). For some $R \geq 1$, the R-th round of Sherali-Adams lift of the linear program has variables x_S, for every $S \in \binom{[n]}{\leq R} := \{S \subseteq [n] : |S| \leq R\}$. For every solution $x \in \mathcal{X}$, x_S is supposed to indicate whether

all the events in S happen or not in the solution x; that is, $x_S = \prod_{i \in S} x_i$. Thus each $x \in \mathcal{X}$ can be naturally extended to a 0/1-vector in the higher-dimensional space defined by all the variables.

To derive the set of constraints, let us focus on the j-th constraint $\sum_{i=1}^{n} a_{j,i} x_i \leq b_j$ in the original linear program. Consider two subsets $S, T \subseteq [n]$ such that $|S| + |T| \leq R - 1$. Then the following constraint is valid for $\mathcal{X}$; i.e, all $x \in \mathcal{X}$, the constraint is satisfied:

$$\textstyle\prod_{i \in S} x_i \prod_{i \in T} (1 - x_i) \left(\sum_{i=1}^{n} a_{j,i} x_i - b_j \right) \leq 0.$$

To *linearize* the above constraint, we expand the left side of the above inequality and replace each monomial with the corresponding $x_{S'}$ variable. Then, we obtain the following :

$$\textstyle\sum_{T' \subseteq T} (-1)^{|T'|} \left(\sum_{i=1}^{n} a_{j,i} x_{S \cup T' \cup \{i\}} - b_j x_{S \cup T'} \right) \leq 0. \tag{1}$$

The R-th round of Sherali-Adams lift contains the above constraint for all j, S, T such that $|S| + |T| \leq R - 1$, and the trivial constraint that $x_\emptyset = 1$. For a polytope $\mathcal{P}$ and an integer $R \geq 1$, we use $\mathrm{SA}(\mathcal{P}, R)$ to denote the poltyope obtained by the R-th round Sherali-Adams lift of $\mathcal{P}$. For every $i \in [n]$, we identify the variable x_i in the original LP and $x_{\{i\}}$ in a lifted LP.

Let $x \in \mathrm{SA}(\mathcal{P}, R)$ for some linear program $\mathcal{P}$ on n variables and $R \geq 2$. Let $i \in [n]$ be an event such that $x_i > 0$; then we can define a solution $x' \in \mathrm{SA}(\mathcal{P}, R-1)$ obtained from x by "conditioning" on the event i. For every $S \in \binom{[n]}{R-1}$, x'_S is defined as $x'_S := \frac{x_{S \cup \{i\}}}{x_i}$. We shall show that x' will be in $\mathrm{SA}(\mathcal{P}, R-1)$ (Property (2.2e)).

It is useful to consider the ideal case where x corresponds to a convex combination of integral solutions in $\mathcal{X}$. Then we can view x as a distribution over $\mathcal{X}$. Conditioning on the event i over the solution x corresponds to conditioning on i over the distribution x. With this view, it is not hard to image the statements in the following claim (which we prove in the appendix) should hold:

Claim 2.2. *For some $x \in \mathrm{SA}(\mathcal{P}, R)$ with $R \geq 2$, the following statements hold:*

(2.2a) $x_S \geq x_{S'}$ *for every* $S \subseteq S' \in \binom{[n]}{\leq R}$.

(2.2b) *If* $x_i = 1$ *for some* $i \in [n]$, *then* $x_{\{i,i'\}} = x_{i'}$ *for every* $i' \in [n]$.

(2.2c) *If every* $\hat{x} \in \mathcal{P}$ *has* $\hat{x}_i \leq \hat{x}_{i'}$, *then* $x_{\{i,i'\}} = x_i$.

Letting x' be obtained from x by conditioning on some event $i \in [n]$, the following holds:

(2.2d) $x'_i = 1$.

(2.2e) $x' \in \mathrm{SA}(\mathcal{P}, R-1)$.

(2.2f) *If* $x_{i'} \in \{0, 1\}$ *for some* $i' \in [n]$, *then* $x'_{i'} = x_{i'}$.

Keep in mind that the three properties (2.2a), (2.2d) and (2.2f) will be used over and over again, often without referring to them. (2.2d) says that conditioning on i will fix x_i to 1. (2.2f) says that once a variable is fixed to 0 or 1, then it can not be changed by conditioning operations.

3 Reducing Directed Steiner Tree to Label-Consistent Subtree

In this section, we present a reduction from DST to LCST. In Section 3.1, we define a *decomposition tree*, which corresponds to a recursive partitioning of a Steiner tree T of G. We show that the DST

problem is equivalent to finding a small cost decomposition tree. Due to the balanced-partition lemma (Lemma 2.1), we can guarantee that decomposition trees have depth $O(\log k)$, a crucial property needed to obtain a quasi-polynomial-time algorithm. Then in Section 3.2 we show that the task of finding a small cost decomposition tree can be reduced to an LCST instance on a tree of depth $O(\log k)$. Roughly speaking, for a decomposition tree to be valid, we require that the separator vertex appears in both parts of a partition: as a root in one part and possibly a non-root in the other. This can be captured by the label-consistency requirement.

We shall use T to denote a Steiner tree in the original graph G, and u, v to denote vertices in G. We use τ to denote a decomposition tree, and α, β to denote *nodes* of a decomposition tree. $\mathbf{T}^0$ will be used for the input tree of the LCST instance. We use $\mathbf{T}$ for a sub-tree of $\mathbf{T}^0$ and p, q, o for *nodes* in $\mathbf{T}^0$. The convention extends to variants of these notations as well.

3.1 Decomposition Trees

We now define decomposition trees. Recall that in the DST problem, we are given a graph $G = (V, E)$, a root $r \in V$, and a set $K \subseteq V \setminus \{r\}$ of k terminals.

Definition 3.1. A decomposition tree τ is a rooted tree where each node α is associated with a vertex $\mu_\alpha \in V(G)$ and each leaf-node α is associated with an edge $e_\alpha \in E(G)$. Moreover, the following conditions are satisfied:

- (3.1a) $\mu_{\mathrm{root}(\tau)} = r$.
- (3.1b) For every leaf β of τ, we have $\mu_\beta = \mathrm{head}(e_\beta)$.
- (3.1c) For every non-leaf α of τ and every child α_2 of α with $\mu_{\alpha_2} \neq \mu_\alpha$ the following holds. There is a child α_1 of α with $\mu_{\alpha_1} = \mu_\alpha$ such that $\mu_{\alpha_2} = \mathrm{tail}(e_\beta)$ for some leaf $\beta \in V(\tau[\alpha_1])$. In particular, this implies that α has at least one child α_1 with $\mu_{\alpha_1} = \mu_\alpha$.

The cost of a decomposition tree τ is defined as $\mathrm{cost}(\tau) := \sum_{\alpha \text{ a leaf of } \tau} c(e_\alpha)$.

We say a vertex v is *involved* in a sub-tree $\tau[\alpha]$ of a decomposition tree τ if either $v = \mu_\alpha$ or there is a leaf β of $\tau[\alpha]$ such that $v = \mathrm{tail}(e_\beta)$. So the second sentence in Property (3.1c) can be changed to the following: There is a child α_1 of α with $\mu_{\alpha_1} = \mu_\alpha$ such that μ_{α_2} is involved in $\tau[\alpha_1]$.

We show that the DST problem can be reduced to the problem of finding a small-cost decomposition tree of depth $O(\log k)$. This is done in two directions.

From Directed Steiner Tree to Decomposition Tree. We first show that the optimum directed Steiner tree T^* of G connecting r to all terminals in K gives a good decomposition tree τ^* of cost at most that of T^*, which we denote by opt. Since we assumed costs of edges in G satisfy triangle inequalities, we can assume every vertex $v \in V(T^*) \setminus (\{r\} \cup K)$ has at least two children in T^*. This implies $|V(T^*)| \leq 2k$. The decomposition tree τ^* can be constructed by applying Lemma 2.1 on T^* recursively until we obtain trees with singular edges. Formally, we set $\tau^* \leftarrow \textsf{cstr-opt-dcmp-tree}(T^*)$, where cstr-opt-dcmp-tree is defined in Algorithm 1. Notice that the algorithm is only for analysis purpose and is not a part of our algorithm for DST.

Claim 3.2. *τ^* is a full binary decomposition tree of height $O(\log k)$ and cost* opt *that involves all vertices in K. Moreover, for every $v \in K$, there is exactly one leaf β of τ^* with* $\mathrm{tail}(e_\beta) = v$.

Algorithm 1 cstr-opt-dcmp-tree(T)

if T consists of a single edge (u, v) **then return** a node β with $\mu_\beta = u$ and $e_\beta = (u, v)$
else
    create a node α with $\mu_\alpha = \text{root}(T)$
    apply Lemma 2.1 to find two rooted trees T_1 and T_2 with $\text{root}(T_1) = \text{root}(T)$
    $\tau_1 \leftarrow$ cstr-opt-dcmp-tree(T_1), $\tau_2 \leftarrow$ cstr-opt-dcmp-tree(T_2)
    return the tree rooted at α with two sub-trees τ_1 and τ_2

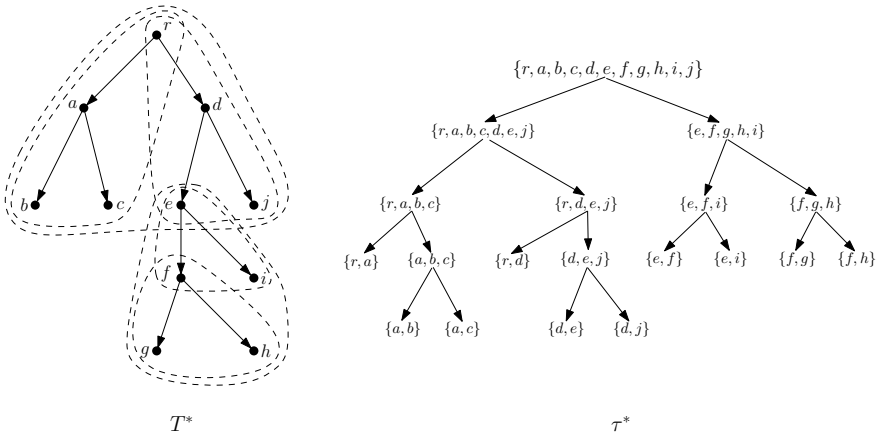

Figure 1: An example for construction of τ^*. For each node τ^*, the set denotes the vertices in the sub-tree of T^* correspondent to the node; the μ value of the node is the first element in the set. For a leaf node, its e value is the edge from the first element to the second element in the set.

From Decomposition Tree to Directed Steiner Tree. Now we show the other direction of the reduction. The lemma we shall prove is the following:

Lemma 3.3. *Given a decomposition tree τ that involves all terminals in K, we can efficiently construct a directed Steiner tree T in G connecting r to all terminals in K with cost at most* $\text{cost}(\tau)$.

Thus, our goal is to find a decomposition tree of small cost involving all terminals in K. To do so, we construct an instance of the LCST problem.

3.2 Construction of LCST Instance

Let $\bar{h}$ be the $O(\log k)$ term in Claim 3.2 that upper bounds the height of τ^*. In the reduction, we shall "collapse" every $g := \lceil \log_2 \log_2 k \rceil$ levels of a decomposition tree into one level; this is used to obtain the improvement of $\Theta(\log \log k)$ in the approximation ratio. It motivates the definition of a *twig*, which corresponds to a full binary tree of depth at most g that can appear as a part of a decomposition tree:

Definition 3.4. A *twig* is a rooted full binary tree η of depth at most g, where

- each $\alpha \in V(\eta)$ is associated with a $\mu_\alpha \in V(G)$, such that for every internal node α in η, at least one child α' of α has $\mu_{\alpha'} = \mu_\alpha$, and

- each leaf β of η *may or may not* be associated with a value $e_\beta \in E(G)$; if e_β is defined then $\text{head}(e_\beta) = \mu_\beta$.

With the twigs defined, our LCST instance $\mathbf{T}^0$ is constructed by calling $\mathbf{T}^0 \leftarrow \text{cstr-label-tree}(r, 0)$, where cstr-label-tree is defined in Algorithm 2. See Figure 2 for illustration of one recursion of cstr-label-tree.

Algorithm 2 cstr-label-tree(u, j)

1: create a new node p with $c_p = 0, u_p = u$ and $\text{dem}(p) = \{\ell\}$ for a newly created local label ℓ
2: **if** $j < \lceil \bar{h}/g \rceil$ **then**
3: **for** each possible non-singular twig η with $\mu_{\text{root}(\eta)} = u$ **do**
4: create a node q with $c_q = \sum_{\text{leaf } \beta \text{ of } \eta : e_\beta \text{ defined}} c(e_\beta), \eta_q = \eta, \text{ser}(q) = \{\ell\}$, and $\text{dem}(q) = \emptyset$
5: let q be a child of p
6: **for** every leaf β of η **do**
7: **if** e_β is defined **then**
8: **if** $\text{tail}(e_\beta) \in K$ **then** add the global label $\text{tail}(e_\beta)$ to $\text{ser}(q)$
9: **else**
10: $\mathbf{T}^q_\beta \leftarrow \text{cstr-label-tree}(\mu_\beta, j+1)$, let $\text{root}(\mathbf{T}^q_\beta)$ be a child of q
11: create a new label ℓ', add ℓ' to $\text{dem}(q)$ and $\text{ser}(\text{root}(\mathbf{T}^q_\beta))$.
12: **for** every internal node α of η **do**
13: let α_1 be a child of α with $\mu_{\alpha_1} = \mu_\alpha$ and α_2 be the other child
14: **if** $\mu_{\alpha_2} \neq \mu_\alpha$ and $\nexists$ leaf β of $\eta[\alpha_1]$ with e_β defined and $\text{tail}(e_\beta) = \mu_{\alpha_2}$ **then**
15: create a new label ℓ' and add it to $\text{dem}(q)$
16: **for** every leaf β of $\eta[\alpha_1]$ with e_β undefined, and q' in $\mathbf{T}^q_\beta$ **do**
17: **if** $\eta_{q'}$ has a leaf β' with $e_{\beta'}$ defined and $\text{tail}(e_{\beta'}) = \mu_{\alpha_2}$ **then** add ℓ' to $\text{ser}(q')$
18: **return** the tree rooted at p

Remark 3.5. The u and η values of nodes in $\mathbf{T}^0$ are irrelevant for the LCST instance. They will, however, help us in mapping the decomposition tree to its corresponding solution to LCST.

Notice that there are two types of nodes in $\mathbf{T}^0$: (1) p-nodes are those created in Step 1 and (2) q-nodes are those created in Step 4. We always use p (q, resp.) and its variants to denote p-nodes (q-nodes resp.).

We give some intuition behind the construction of $\mathbf{T}^0$. We can partition the edges of a decomposition tree τ into an $O(\bar{h}/g)$-depth tree $\mathbf{H}$ of twigs. For each η in the tree, we apply the following operation. First, we replace η with a node q with $\eta_q = \eta$. Second, we insert a virtual parent p of q with $u_p = \mu_{\text{root}(\eta)}$ between this q and its actual parent. Then it is fairly straightforward to see that we can find a copy of this resulting tree in $\mathbf{T}^0$. Thus, we reduced the problem of finding $\mathbf{H}$ (and thus τ) to the problem of finding a subtree $\mathbf{T}$ of $\mathbf{T}^0$. The label-consistency requirements shall guarantee that $\mathbf{T}$ will correspond to a valid τ. In particular, the demand label ℓ for a node p created in Step 1 guarantees that if p is selected then we shall select at least one child of p. The demand labels created in Step 11 for a node q guarantee that if q is selected, then all its children must be selected, while the demand labels created in Step 15 guarantee Property (3.1c) of τ. The set of global labels is exactly K. In Step 8, we add a global label $v \in K$ to q if η_q contains a leaf β with $\text{tail}(e_\beta) = v$.

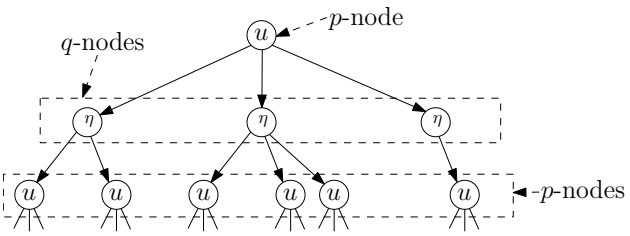

Figure 2: Nodes created in one recursion of cstr-label-tree. Each p-node has a u_p value, and each q-node is associated with a twig η_q with $\mu_{\text{root}(\eta_q)}$ being the u value of its parent p-node. Each child p' of q corresponds to a leaf β of η_q with e_β undefined.

A simple observation we can make is the following:

Claim 3.6. *$\mathbf{T}^0$ is a rooted tree with $n^{O(\log^2 k/\log\log k)}$ vertices and height $O(\bar{h}/g) = O(\log k/\log\log k)$, where $n = |V(G)|$.*

Also, it is easy to see that a node p will have exactly one demand label, while a node q can have up to $O(2^g)$ demand labels. So, we have $s := \max_{p \in V(\mathbf{T}^0)} |\text{dem}(v)| = O(2^g) = O(\log k)$.

We then show that the problem of finding a decomposition tree can be reduced to that of finding a label-consistent subtree of $\mathbf{T}^0$. Again, this is done in two directions.

From Decomposition Tree to Label-Consistent Subtree To show that there is a good label-consistent subtree $\mathbf{T}^*$ of $\mathbf{T}^0$, we need to construct a tree of twigs from τ^*. This is done as follows. For every $i = 0, 1, 2, \cdots$, and every internal node α in τ^* of depth ig, we create a twig rooted at α containing all descendants of α at depth $ig, ig+1, ig+2, \cdots, (i+1)g$. Let $\mathcal{V}$ be the set of twigs created. A rooted tree $\mathbf{H}$ over $\mathcal{V}$ can be naturally defined: a twig η is a parent of η' if and only if $\text{root}(\eta')$ is a leaf in η. So, $\mathbf{H}$ has depth at most $\lceil \bar{h}/g \rceil$.

Algorithm 3 cstr-opt-LCST(p, η)

1: add p and the child q of p with $\eta_q = \eta$ to $\mathbf{T}^*$ ▷ such a q exists since $\mu_{\text{root}(\eta)} = u_p$
2: **for** every leaf β of η such that e_β is not defined **do**
3: let η' be the twig in $\mathcal{V}$ with $\text{root}(\eta') = \beta$
4: cstr-opt-LCST$(\text{root}(\mathbf{T}^q_\beta), \eta')$

$\mathbf{T}^*$ can be found naturally by calling cstr-opt-LCST$(\text{root}(\mathbf{T}^0), \text{root}(\mathbf{H}))$ (with $\mathbf{T}^*$ being empty initially), where cstr-opt-LCST is defined in Algorithm 3, and the trees $\mathbf{T}^q_\beta$ are as defined in Algorithm 2. The recursive procedure takes two parameters: a node p in $\mathbf{T}^0$ and a twig $\eta \in \mathcal{V}$. It is guaranteed that $u_p = \mu_{\text{root}(\eta)}$: The root recursion satisfy this condition since $u_{\text{root}(\mathbf{T}^0)} = \mu_{\text{root}(\text{root}(\mathbf{H}))} = r$; in Step 4, we also have $u_{\text{root}(\mathbf{T}^q_\beta)} = \mu_\beta = \mu_{\text{root}(\eta')}$. The tree can be constructed as $\mathbf{H}$ has depth at most $\lceil \bar{h}/g \rceil$. Again, this algorithm is only for analysis purpose and is not a part of our algorithm for DST. We prove in the appendix the following lemma.

Lemma 3.7. *$\mathbf{T}^*$ is a label-consistent sub-tree of $\mathbf{T}^0$ with cost exactly $\text{cost}(\tau^*) = \text{opt}$. Moreover, all global labels in K are supplied by $\mathbf{T}^*$.*

From Label-Consistent Subtree to Decomposition Tree. The following lemma gives the other direction, and its proof will be deferred to the appendix.

Lemma 3.8. *Given any feasible solution* $\mathbf{T}$ *to the LCST instance* $\mathbf{T}^0$*, in time* $\text{poly}(|V(\mathbf{T})|)$ *we can construct a decomposition tree* τ *with* $\text{cost}(\tau) = \text{cost}(\mathbf{T})$*. Moreover, if a global label* $v \in K$ *is supplied by* $\mathbf{T}$*, then* τ *involves* v*.*

Wrapping up. We prove the following theorem in the next section. Recall that N and h are respectively the size and height of the input tree T^0 to the LCST instance, and k is the number of global labels.

Theorem 3.9. *There is an* $(shN)^{O(sh^2)}$*-time* $O(h \log k)$*-approximation algorithm for the Label-Consistent Subtree problem where* $s := \max_{v \in V(T^0)} |\text{dem}(v)|$*.*

With this theorem at hand, we can now finish our $O(\log^2 k / \log\log k)$-approximation for DST that runs in quasi-polynomial time. Given a DST instance, we shall construct the LCST instance $\mathbf{T}^0$ of size $N = n^{O(\log^2 k/\log\log k)}$ and height $h = O(\log k/\log\log k)$ as in Algorithm 2. Notice that for the LCST instance, we have $s := \max_{p \in V(\mathbf{T}^0)} |\text{dem}(p)| = O(2^g) = O(\log k)$. By Claim 3.2 and Lemma 3.7, there is a solution $\mathbf{T}^*$ to the LCST instance $\mathbf{T}^0$ of cost at most opt. Applying Theorem 3.9, we can obtain a feasible solution $\mathbf{T}$ of cost at most $O(h \log k) \cdot \text{opt} = O(\log^2 k/\log\log k) \cdot \text{opt}$ in time $(shN)^{O(sh^2)} = n^{O(\log^5 k)}$ (as $s = O(\log k)$). Applying Lemma 3.8 and Lemma 3.3, we can obtain a Directed Steiner tree T in G of cost at most $O(\log^2 k/\log\log k) \cdot \text{opt}$ connecting r to all terminals in K. This gives a $O(\log^2 k/\log\log k)$-approximation for DST in running time $n^{O(\log^5 k)}$, finishing the proof of Theorem 1.1.

4 Approximation Algorithm for Label-Consistent Subtree

The goal of this section is to prove Theorem 3.9, which is repeated below. Since we are not dealing with the original DST problem any more, we use T^0, T for trees and u, v for nodes in this section.

Theorem 3.9. *There is an* $(shN)^{O(sh^2)}$*-time* $O(h \log k)$*-approximation algorithm for the Label-Consistent Subtree problem where* $s := \max_{v \in V(T^0)} |\text{dem}(v)|$*.*

4.1 Redefining the LCST Problem

We shall first simplify the input instance w.l.o.g in the following ways that will make our presentation much cleaner. Indeed, some properties are already satisfied by the LCST instance reduced from the DST problem; however we want to make Theorem 3.9 as general as possible and thus we do not make these assumptions in the theorem statement.

1. We can assume for every two distinct nodes u and v, $\text{dem}(u)$ and $\text{dem}(v)$ are disjoint. If some local label ℓ appears in $\text{dem}(u)$ for $t \geq 2$ different nodes u, we can make t copies of ℓ and let each copy be contained in $\text{dem}(u)$ for exactly one u. We can replace the appearance of ℓ in some $\text{ser}(v)$ with the t copies.

2. We can assume the demand labels are only at the internal nodes. Suppose a leaf v has $\ell \in \mathrm{dem}(v)$. If $\ell \in \mathrm{ser}(v)$, then ℓ can be removed from $\mathrm{dem}(v)$; otherwise v can never be selected thus can be removed from T^0.
3. We can assume that the service labels are only at the leaves and each leaf contains exactly one service label. A leaf without a service label can be removed. For a non-leaf v with $\mathrm{ser}(v) \neq \emptyset$, we can attach $|\mathrm{ser}(v)|$ leaves of cost 0 to v and distribute the service labels to the newly added leaves. Similarly, if a leaf v has $|\mathrm{ser}(v)| > 1$, we can attach $|\mathrm{ser}(v)|$ new leaves to v.

Notice that the above operations do not change the set K of global labels and $s = \max_{v \in V(T^0)} |\mathrm{dem}(v)|$.

With the above operations and simplifications, we can redefine the LCST instance. Let V^{leaf} and V^{int} respectively be the sets of leaves and internal nodes of T^0. For every node $v \in V^{\mathrm{int}}$, let Λ_v be the set of children of v. For every $v \in V(T^0)$, let $\Lambda_v^{\mathrm{leaf}} = V(T^0[v]) \cap V^{\mathrm{leaf}}$ be the set of descendants of v that are leaves.

For every $v \in V^{\mathrm{leaf}}$, let a_v be the unique label in $\mathrm{ser}(v)$. From now on we shall not use the notation $\mathrm{ser}(\cdot)$ anymore. Thus, a rooted subtree T of T^0 with $\mathrm{root}(T) = \mathrm{root}(T^0)$ is label-consistent if, for every $u \in V(T) \cap V^{\mathrm{int}}$ and $\ell \in \mathrm{dem}(u)$, there is a node $v \in V(T) \cap \Lambda_u^{\mathrm{leaf}}$ with $a_v = \ell$.

The goal of the problem is to find the minimum cost label-consistent subtree T of T^0 that provides all the global labels, i.e, that satisfies for all $\ell \in K$ there exists a $v \in V(T) \cap V^{\mathrm{leaf}}$ with $a_v = \ell$. Recall that we are given a node-cost vector $c \in \mathbb{R}_{\geq 0}^{V(T^0)}$. The cost of a sub-tree T of T^0, denoted as $\mathrm{cost}(T)$, is defined as $\mathrm{cost}(T) := \sum_{v \in V(T)} c_v$.

We consider the change in the size and height of T^0 after we applied the above operations. Abusing notations slightly, we shall use N' and h' to store the size and height of the old T^0 (i.e, the T^0 before we apply the operations), and N and h be the size and height of the new T^0 (i.e, the T^0 after we apply the operations). Notice that we only added leaves to T^0. Thus, we have $h \leq h' + 1$. The number of internal nodes in the new T^0 is at most N'. A leaf v is relevant only when it is providing a label that are in $\mathrm{dem}(u)$ for some ancestor u of v. If a node has many leaf children with the same service label, we only need to keep the one with the smallest cost. Since each u has $|\mathrm{dem}(u)| \leq s$ and the height of the old T^0 is h', we can assume that the number of leaves in the new T^0 is at most $s(h'+1)N'$. So $N \leq s(h'+1)N' + N' = O(sh'N')$.

Let T^* be the optimum tree for the given instance. Let opt be the cost of the T^*, i.e, $\mathrm{opt} = \mathrm{cost}(T^*)$.[2] As every local label appears only once in V^{int}, we can assume that for every $\ell \in L$, there is at most one node $v \in V(T^*) \cap V^{\mathrm{leaf}}$ with $a_v = \ell$: if there are multiple such nodes v, we can keep one without violating the label-consistency condition and that all global labels are provided. Thus additionally we can assume T^* satisfies the following conditions:

(4.1a) For every $\ell \in K$, there is exactly one node $v \in V(T^*) \cap V^{\mathrm{leaf}}$ such that $a_v = \ell$.

(4.1b) For every $\ell \in L \setminus K$, there is at most one node $v \in V(T^*) \cap V^{\mathrm{leaf}}$ such that $a_v = \ell$.

The main theorem we shall prove is the following

Theorem 4.2. *There is an $(sN)^{O(sh^2)}$-time algorithm that outputs a random label-consistent tree $\tilde{T}$ such that, $\mathbb{E}\left[c(\tilde{T})\right] \leq \mathrm{opt}$, and for every $\ell \in K$, we have $\Pr\left[\exists v \in V^{\mathrm{leaf}} \cap V(\tilde{T}) : a_v = \ell\right] \geq \frac{1}{h+1}$.*

[2] We remark that it is easy to check whether a valid solution exists or not: an $u \in V^{\mathrm{int}}$ is useless if for some $\ell \in \mathrm{dem}(u)$ there is no $v \in \Lambda_u^{\mathrm{leaf}}$ with $a_v = \ell$. We repeatedly remove useless nodes and their descendents until no such nodes exist. There is a valid solution iff the remaining T^0 provides all labels in K. So we can assume the instance has a valid solution.

With theorem 4.2, we can finish the proof of Theorem 3.9.

Proof of Theorem 3.9. We run $O(h \log k)$ times the algorithm stated in Theorem 4.2 and let T' be the union of all the trees $\tilde{T}$ produced. It is easy to see that T' is always label-consistent. The expected cost of T' is

$$\mathbb{E}\left[\text{cost}(T')\right] \leq O(h \log k)\text{opt}.$$

If the $O(h \log k)$ term is sufficiently large, by the union bound, we can obtain

$$\Pr\left[\forall \ell \in K, \exists v \in V^{\text{leaf}} \cap V(T'), a_v = \ell\right] \geq 1/2. \tag{2}$$

We repeatedly run the above procedure until $\forall \ell \in K, \exists v \in V^{\text{leaf}} \cap V(T'), a_v = \ell$ happens and output the tree T' satisfying the property. Let T^{final} be this tree. Then we have $\mathbb{E}\left[\text{cost}(T^{\text{final}})\right] \leq O(h \log k)\text{opt}$ due to (2). In expectation we only need run the procedure twice.

Thus, we obtain an $O(h \log k)$-approximation algorithm for LCST. The running time of the algorithm is $(sN)^{O(sh^2)} = (sh'N')^{O(sh'^2)}$. Recall that h' and N' are the height and size of T^0 before we applied the operations; thus the theorem follows. □

Thus, our goal is to prove Theorem 4.2. Our algorithm is very similar to that of [30] for GST on trees. We solve the lifted LP relaxation for the LCST problem and then round the fractional solution via a recursive procedure. In the procedure, we focus on some sub-tree $T^0[u]$, and we are given a set L' of labels that must appear in $\tilde{T}[u]$, where $\tilde{T}$ is our output tree. We are also given a lifted LP solution x; we can restrict x on the tree $T^0[u]$. The set L' of labels appear in $T^0[u]$ fully according to x. Then, for every $\ell \in L'$, we randomly choose child v of u that is responsible for this ℓ and then apply some conditioning operations on x. We recursively call the procedure for the children of u. This way, we can guarantee that the tree $\tilde{T}$ we output is always label-consistent. Finally, we show that each global label $v \in K$ appears in $\tilde{T}$ with large probability, using the technique that is very similar to that of [30].

4.2 Basic LP Relaxation

The remaining part of the section is dedicated to the proof of Theorem 4.2. We formulate an LP relaxation that aims at finding the T^*, where the variables of the LP are indexed by $\mathbb{D} = V(T^0) \cup (V(T^0) \times L)$. We view every element in $\mathbb{D}$ also as an event. Supposedly, an event $u \in V(T^0)$ happens if and only if $u \in V(T^*)$, and an event $(u, \ell) \in V(T^0) \times L$ happens if and only if $u \in V(T^*)$ and $\Lambda_u^{\text{leaf}} \cap V(T^*)$ has a node with label ℓ (such a node is unique if it exists by Properties (4.1a) and (4.1b)). For every $e \in \mathbb{D}$, $x_e \in \{0, 1\}$ is supposed to indicate whether event e happens or not. Then the following linear constraints are valid:

$$x_v \leq x_u, \quad \forall u \in V^{\text{int}}, v \in \Lambda_u \tag{3}$$

$$x_{(u,\ell)} \leq x_u, \quad \forall u \in V(T^0), \ell \in L \tag{4}$$

$$x_{(u,\ell)} = x_u, \quad \forall u \in V^{\text{int}}, \ell \in \text{dem}(u) \tag{5}$$

$$x_{(v,a_v)} = x_v, \quad \forall v \in V^{\text{leaf}} \tag{6}$$

$$x_{(u,\ell)} = \sum_{v \in \Lambda_u} x_{(v,\ell)}, \quad \forall u \in V^{\text{int}}, \ell \in L \tag{7}$$

$$x_{(v,\ell)} = 0, \quad \forall v \in V^{\text{leaf}}, \ell \neq a_v \tag{8}$$

$$x_{(\text{root}(T^0),\ell)} = 1, \quad \forall \ell \in K \tag{9}$$

(3) holds since T^* is rooted sub-tree of T^0 with $\text{root}(T^*) = \text{root}(T^0)$, (4) holds by definition of events, (5) follows from that T^* is label-consistent, and (6) holds trivially. (7) follows from Properties (4.1a) and (4.1b). (8) holds trivially and (9) follows from Property (4.1a).

Let $\mathcal{P}$ be the polytope containing all vectors $x \in [0,1]^{\mathbb{D}}$ satisfying constraints (3) to (9). The following simple observation can be made:

Claim 4.3. *For every $x \in \mathcal{P}$, $u' \in V(T^0)$, and $\ell' \in L$, we have* $\sum_{v \in \Lambda_{u'}^{\text{leaf}}} x_{v,\ell'} = x_{u',\ell'}$.

Proof. The claim holds trivially if $u' \in V^{\text{leaf}}$. When $u' \notin V^{\text{leaf}}$, summing up (7) over all internal nodes u in $T^0[u']$ and $\ell = \ell'$ gives the equality. □

4.3 Rounding a Lifted Fractional Solution

Let $R = O(sh^2)$ be large enough. Since $\mathcal{P}$ contains an integral solution of cost at most opt, we can find a solution $x^* \in \text{SA}(\mathcal{P}, R)$ with $\sum_{v \in V(T^0)} c_v x_v^* \leq \text{opt}$ in running time $|\mathbb{D}|^{O(sh^2)} = (sN)^{O(sh^2)}$.

Remark 4.4. Indeed, our algorithm only needs to use variables that correspond to paths of $\mathbf{T}^0$ starting at the root. Using this one can remove a $\log k / \log\log k$ factor from the exponent of the running time. However, we choose to use the Sherali-Adams hierarchy as it is much easier to describe.

In the main rounding algorithm (Algorithm 4), we let $\tilde{V} = \emptyset$ initially and call $\text{solve}(\text{root}(T^0), \text{dem}(\text{root}(T^0)), x^*)$, as described in Algorithm 5. We output the subtree $\tilde{T}$ of T^0 induced by $\tilde{V}$.

Algorithm 4 Main Rounding

Given: $x^* \in \text{SA}(\mathcal{P}, R)$
Output: a label-consistent tree $\tilde{T}$

1: $\tilde{V} \leftarrow \emptyset$
2: $\text{solve}(\text{root}(T^0), \text{dem}(\text{root}(T^0)), x^*)$
3: **return** the tree $\tilde{T}$ induced by $\tilde{V}$

Algorithm 5 $\text{solve}(u, L', x)$

1: $\tilde{V} \leftarrow \tilde{V} \cup \{u\}$
2: **if** $u \in V^{\text{leaf}}$ **then return**
3: let $S_v \leftarrow \emptyset$ for every $v \in \Lambda_u$
4: **for** every $\ell \in L'$ **do**
5: randomly choose a child v of u, so that v is chosen with probability $x_{(v,\ell)}$ (see Property (4.5b))
6: $S_v \leftarrow S_v \cup \{\ell\}$
7: $x \leftarrow x$ conditioned on the event (v, ℓ)
8: **for** every $v \in \Lambda_u$, with probability x_v, **do**
9: $\text{solve}(v, S_v \cup \text{dem}(v), x$ conditioned on event $v)$

In the recursive algorithm $\text{solve}(u, L', x)$, u is the current node we are dealing with. L' is the set of labels that must be supplied in $\tilde{T}[u]$; in particular, we shall guarantee that $\text{dem}(u) \subseteq L'$. x is the LP hierarchy solution that is passed to u, which satisfies $x_u = 1$ and $x_{(u,\ell)} = 1$ for every $\ell \in L'$ (Property (4.5a) in Claim 4.5 that appears later). We add u to $\tilde{V}$ in Step 1; thus the final $\tilde{T}$ contains the set of nodes for which we called solve.

If $u \in V^{\text{leaf}}$, we then do nothing; so focus on the case $u \notin V^{\text{leaf}}$. To guarantee that a label $\ell \in L'$ is supplied in $\tilde{T}[u]$, we need to specify one child v of u such that $\tilde{T}[v]$ supplies ℓ; we say that v is responsible for this label ℓ. This is done via a random procedure by using the solution x as a guide: the probability that v is chosen is exactly $x_{(v,\ell)}$ (Step 5). We shall show that $\sum_{v \in \Lambda_u} x_{v,\ell} = 1$

(Property (4.5b)) and thus the process is well-defined. After choosing the v for this $\ell \in L'$, we update x by conditioning on the event (v, ℓ) (Step 7). So far the number of nested conditioning operations we apply on x is $|L'|$; we will see soon that $|L'|$ is small and thus we can apply these operations.

For every $v \in \Lambda_u$, let S_v be the set of labels in L' that v is responsible for. In Loop 8, we independently and recursively call solve on the children of u. Notice that x_v is the extent to which v is included in $V(T^*)$. So we only call solve on v with probability x_v; the LP solution passed to the sub-recursion is x conditioned on the event v. In particular if $S_v \neq \emptyset$ then $x_v = 1$. We remark that the conditioning operations for all children v of u are done "in parallel" and thus we "lose only 1 level" of our Sherali-Adams lifting.

We now analyze the algorithm. To prove Theorem 4.2, we need to show that $\tilde{T}$ is a label-consistent subtree with small expected cost; moreover, every label $\ell \in K$ is provided by $\tilde{T}$ with large enough probability. Let us first assume that the number R of rounds is large enough so that all the conditioning operations can be applied. We start from some simple observations for the algorithm.

Claim 4.5. *For every recursion of* solve *that the algorithm invokes,*

(4.5a) at the beginning the recursion, we have $x_u = 1$ and $x_{(u,\ell)} = 1$ for all $\ell \in L'$, and

(4.5b) the random sampling process in Step 5 is well-defined: we have $\sum_{v \in \Lambda_u} x_{(v,\ell)} = 1$ before the step.

Proof. (4.5a) holds for the root recursion as (9) implies $x^*_{\mathrm{root}(T^0)} = 1$ and (5) implies $x^*_{\mathrm{root}(T^0),\ell} = x^*_{\mathrm{root}(T^0)} = 1$ for every $\ell \in \mathrm{dem}(\mathrm{root}(T^0))$.

Now assume (4.5a) holds for some recursion for $u \notin V^{\mathrm{leaf}}$. So, at the beginning of an iteration of Loop 4, we have $x_{u,\ell} = 1$ for every $\ell \in L'$. Thus, by (7), we have $\sum_{v \in \Lambda_u} x_{v,\ell} = 1$, implying (4.5b) for this recursion.

Since we conditioned on the event (v, ℓ) in Step 7 after adding ℓ to S_v, we have $x_{(v,\ell)} = 1$ for every $v \in \Lambda_u$ and $\ell \in S_v$ after finishing Loop 4. (Notice that Property (2.2f) says that once a variable has value 0 or 1, conditioning operations do not change its value.) Focus on Step 9 for some $v \in \Lambda_u$, and let x' be the x passed to the sub-recursion, i.e, x' is obtained from x by conditioning on the event v. Then we have that $x'_v = 1$ and $x'_{(v,\ell)} = 1$ for every $\ell \in S_v$. Also, $x'_{(v,\ell)} = x'_v = 1$ for every $\ell \in \mathrm{dem}(v)$ by (5). Since $L' = S_v \cup \mathrm{dem}(v)$ in the sub-recursion of solve for v, (4.5a) holds for the sub-recursion for v. □

Claim 4.6. *The tree $\tilde{T}$ returned by Algorithm 4 is label-consistent.*

Proof. When we call solve for an u, it is guaranteed that $\mathrm{dem}(u) \subseteq L'$ (by Step 2 in Algorithm 4 and Step 9 in Algorithm 5). By the way we construct S_v's in Loop 4 of Algorithm 5, each label $\ell \in \mathrm{dem}(u)$ will be passed down all the way to some leaf node $v \in \Lambda_u^{\mathrm{leaf}}$. By Property (4.5a) for the recursion of solve for v, we must have $x_{v,\ell} = 1$ at the beginning of this recursion. Then by (8), $\ell = a_v$ must hold. □

Claim 4.7. *If $R = O(sh^2)$ is large enough, then all the conditioning operations can be performed.*

Proof. Notice that for the recursion of solve for u, the size of $|L'|$ passed to the recursion is at most $s(\mathrm{depth}(u) + 1)$, where $\mathrm{depth}(u)$ is the depth of u in the tree T^0, i.e, the distance from $\mathrm{root}(T^0)$

to u. This holds since in a recursion of solve for u, S_v's are subsets of L', and the L' passed to the sub-recursion for v is $S_v \cup \mathrm{dem}(v)$ and $|\mathrm{dem}(v)| \le s$.

Inside each recursion of solve, the number of nested conditioning operations is $|L'| + 1 \le s(\mathrm{depth}(u) + 1) \le s(h+2)$. Since the recursion can take up to $h+1$ levels, the number R of rounds we need is at most $s(h+2)(h+1)+1 = O(sh^2)$. □

Notations and Maintenance of Marginal Probabilities. We say that an event $e \in \mathbb{D}$ is inside $T^0[u]$ for some $u \in V(T^0)$ if either $e = v \in V(T^0[u])$ or $e = (v, \ell)$ for some $v \in V(T^0[u])$. For every integer $i \in [0, sh]$, let $x^{(u,i)}$ be the value of x after the i-th iteration of Loop 4 in the recursion $\mathsf{solve}(u, \cdot, \cdot)$. If this recursion does not exist, then let $x^{(u,i)}$ be the all-0 vector over $\mathbb{D}$; if this recursion exists but Loop 4 terminates in less than i iterations in the recursion, then let $x^{(u,i)}$ be the value of x at the end of loop. Notice that Loop 4 terminates in at most sh iterations from the proof of Claim 4.7.

The randomness of the algorithm comes from Steps 5 and 8 in solve. Each time we run Step 5 or 8, we assume we first generate a random number and then use it to make the decision. We say a random number is generated before $x^{(u,i)}$, if the random number is generated in $\mathsf{solve}(u', \cdot, \cdot)$ for some ancestor u' of u, or in $\mathsf{solve}(u, \cdot, \cdot)$ before or at the i-th iteration of Loop 4. Notice that each $x^{(u,i)}$ is completely determined by the random numbers generated before it. The following two claims state that the marginal probabilities of events are maintained in our random process.

Claim 4.8. *Let $u \in V(T^0), i \in [sh]$, $x^{\mathrm{old}} = x^{(u,i-1)}$ and $x^{\mathrm{new}} = x^{(u,i)}$. Let $\mathcal{E}$ be any event determined by the random numbers generated before $x^{\mathrm{old}} = x^{(u,i-1)}$. Then, for every $e \in \mathbb{D}$, we have*

$$\mathbb{E}[x_e^{\mathrm{new}} | x_e^{\mathrm{old}}, \mathcal{E}] = x_e^{\mathrm{old}}.$$

Proof. Conditioned on that the i-th iteration of $\mathsf{solve}(u, \cdot, \cdot)$ does not exist, the equality holds trivially. So we condition on that the iteration exists. Let L' be the L' passed to $\mathsf{solve}(u, \cdot, \cdot)$; then the ℓ handled in the i-th iteration is determined by L' and i. So,

$$\mathbb{E}\left[x_e^{\mathrm{new}} \middle| x^{\mathrm{old}}, L'\right] = \sum_{v \in \Lambda_u} x^{\mathrm{old}}_{(v,\ell)} \cdot \frac{x^{\mathrm{old}}_{\{e,(v,\ell)\}}}{x^{\mathrm{old}}_{(v,\ell)}} = \sum_{v \in \Lambda_u} x^{\mathrm{old}}_{\{e,(v,\ell)\}} = x^{\mathrm{old}}_{\{e,(u,\ell)\}} = x_e^{\mathrm{old}}.$$

The first equality is by the random process for choosing v and the definition of the conditioning operation. The second-to-last equality follows from Constraint (7), and the last equality follows from $x^{\mathrm{old}}_{(u,\ell)} = 1$ and Property (2.2b).

Also, given x^{old} and L', the random process in the i-th iteration of $\mathsf{solve}(u, \cdot, \cdot)$ does not depend on the random numbers generated before x^{old}, and thus does not depend on $\mathcal{E}$. Therefore, $\mathbb{E}\left[x_e^{\mathrm{new}} \middle| x^{\mathrm{old}}, L', \mathcal{E}\right] = x_e^{\mathrm{old}}$. Deconditioning over L' and the components inside x^{old} other than x_e^{old} gives $\mathbb{E}\left[x_e^{\mathrm{new}} \middle| x_e^{\mathrm{old}}, \mathcal{E}\right] = x_e^{\mathrm{old}}$. □

Claim 4.9. *Let $u \in V^{\mathrm{int}}, v \in \Lambda_u, x^{\mathrm{old}} = x^{(u,sh)}$ and $x^{\mathrm{new}} = x^{(v,0)}$. Let $\mathcal{E}$ be any event determined by the random numbers generated before $x^{\mathrm{old}} = x^{(u,sh)}$. Then, for any event e inside $T^0[v]$, we have*

$$\mathbb{E}\left[x_e^{\mathrm{new}} \middle| x_e^{\mathrm{old}}, \mathcal{E}\right] = x_e^{\mathrm{old}}.$$

Proof. Again we can condition on that the recursion $\mathsf{solve}(u,\cdot,\cdot)$ exists. Consider the iteration of Loop 8 for v in $\mathsf{solve}(u,\cdot,\cdot)$. We have

$$\mathbb{E}\left[x_e^{\text{new}} \middle| x^{\text{old}}, \mathcal{E}\right] = x_v^{\text{old}} \times \frac{x_{\{e,v\}}^{\text{old}}}{x_v^{\text{old}}} = x_{\{e,v\}}^{\text{old}} = x_e^{\text{old}}.$$

The first equality holds since we make the recursive call for v with probability x_v^{old}; given x^{old}, this is independent of $\mathcal{E}$. The last equality comes from that event e is inside $T^0[v]$ and thus $\hat{x}_e \le \hat{x}_v$ for every $\hat{x} \in \mathcal{P}$; Property (2.2c) gives the equality.

Again, deconditioning over the components inside x^{old} other than x_e^{old} gives $\mathbb{E}\left[x_e^{\text{new}} \middle| x_e^{\text{old}}, \mathcal{E}\right] = x_e^{\text{old}}$. □

Corollary 4.10. *For every $v \in V(T^0)$, we have $\Pr[v \in \tilde{V}] = x_v^*$.*

Proof. Let $u_1 = \text{root}(T^0), u_2, \cdots, u_t = v$ be the path from $\text{root}(T^0)$ to v in T^0. Applying Claims 4.8 and 4.9, we can obtain that the sequence $x_v^{(u_1,0)}, x_v^{(u_1,1)}, \cdots, x_v^{(u_1,sh)}, x_v^{(u_2,0)}, x_v^{(u_2,1)}, \cdots, x_v^{(u_2,sh)}, \cdots, x_v^{(u_{t-1},0)}, x_v^{(u_{t-1},1)}, \cdots, x_v^{(u_{t-1},sh)}, x_v^{(u_t,0)}$ forms a martingale. This holds since all variables before a variable $x^{(u',i)}$ in the sequence are determined only by random numbers generated before $x^{(u',i)}$. Thus $\Pr[v \in \tilde{V}] = \mathbb{E}\left[x_v^{(v,0)}\right] = x^{(\text{root}(T^0),0)} = x_v^*$ as $x_v^{(\text{root}(T^0),0)}$ is deterministic. □

Then it is immediately true that the expected cost of $\tilde{T}$ is small.

Corollary 4.11. $\mathbb{E}[\text{cost}(\tilde{T})] \le \text{opt}$.

Proof. $\mathbb{E}[\text{cost}(\tilde{T})] = \sum_{v \in V(T^0)} \Pr[v \in \tilde{V}] \cdot c_v = \sum_{v \in V(T^0)} x_v^* c_v \le \text{opt}.$ □

Bounding Probability of Label $\ell \in K$ Appearing in $\tilde{T}$ To finish the proof of Theorem 4.2, it suffices to show that the probability that a label $\ell \in K$ is provided by $\tilde{T}$ with high probability. *Till the end of the proof, we shall fix a label $\ell \in K$.*

Let $t_\ell = \left|\{v \in \tilde{V} \cap V^{\text{leaf}} : a_v = \ell\}\right|$ be the number of nodes in $\tilde{V} \cap V^{\text{leaf}}$ with label ℓ. Our goal is to prove that $t_\ell \ge 1$ with high probability. The proof is almost the same as the counterpart in [30]; we include it here for completeness.

Lemma 4.12. $\mathbb{E}[t_\ell] = 1$.

Proof. By Corollary 4.10, we have

$$\mathbb{E}[t_\ell] = \mathbb{E}\left[\left|\{v \in \tilde{V} \cap V^{\text{leaf}} : a_v = \ell\}\right|\right] = \sum_{v \in V^{\text{leaf}}: a_v = \ell} \Pr[v \in \tilde{V}] = \sum_{v \in V^{\text{leaf}}: a_v = \ell} x_v^* = x^*_{(\text{root}(T^0),\ell)} = 1,$$

where the second-to-last equality follows from Claim 4.3, and the last equality is by (9). □

Lemma 4.13. *For every $w \in V^{\text{leaf}}$ with $a_w = \ell$, we have $\mathbb{E}[t_\ell | w \in \tilde{V}] \le h + 1$.*

Proof. Assume w is at depth h' in the tree T^0. We partition the set $\{w' \in V^{\mathrm{leaf}} \setminus \{w\} : a_{w'} = \ell\}$ of leaves into h' sets $U_0, U_1, \cdots, U_{h'-1}$ according to the LCA of w' and w: w' is in U_i if the LCA of w' and w has depth i in the tree T^0 (the root $\mathrm{root}(T^0)$ has depth 0). Notice that $w' \neq w$ and thus the LCA has depth between 0 and $h'-1$. We show that for every $i = 0, 1, \cdots, h'-1$,

$$\mathbb{E}\left[|U_i \cap \tilde{V}| \middle| w \in \tilde{V}\right] \leq 1. \tag{10}$$

Summing up the inequality over all $i = 0, 1, \cdots, h'-1$ and taking w itself into account implies $\mathbb{E}[t_\ell | w \in \tilde{V}] \leq h' + 1 \leq h + 1$.

Thus, it remains to prove (10). We fix an $i \in \{0, 1, \cdots, h'-1\}$ and let u be the ancestor of w with depth i. Focus on any $w' \in U_i$; thus u is the LCA of w' and w. Let $(S_v)_{v \in \Lambda_u}$ be the vector $(S_v)_{v \in \Lambda_u}$ before Loop 8 in $\mathsf{solve}(u, \cdot, \cdot)$.

Given $\{S_v\}_{v \in \Lambda_u}$ and $x^{(u,sh)}$, the two events $w \in \tilde{V}$ and $w' \in \tilde{V}$ are independent. Thus,

$$\begin{aligned}\Pr\left[w' \in \tilde{V} \middle| \{S_v\}_{v \in \Lambda_u}, x^{(u,sh)}, w \in \tilde{V}\right] &= \Pr\left[w' \in \tilde{V} \middle| \{S_v\}_{v \in \Lambda_u}, x^{(u,sh)}\right] \\ &= \mathbb{E}\left[x_{w'}^{(w',0)} \middle| \{S_v\}_{v \in \Lambda_u}, x^{(u,sh)}\right] = x_{w'}^{(u,sh)}.\end{aligned}$$

To see the third equality, consider the path $u, u_1, u_2, \cdots, u_t = w'$ from u to w' in T^0. Then Claims 4.8 and 4.9 imply that conditioned on $\{S_v\}_{v \in \Lambda_u}$ and $x^{(u,sh)}$, the sequence $x^{(u_1,0)}$, $x^{(u_1,1)}, \cdots, x^{(u_1,sh)}, x^{(u_2,0)}, x^{(u_2,1)} \cdots, x^{(u_{t-1},sh)}, x^{(u_t,0)}$ is a martingale.

Summing up over all $w' \in U_i$, we have

$$\mathbb{E}\left[|U_i \cap \tilde{V}| \middle| \{S_v\}_{v \in \Lambda_u}, x^{(u,sh)}, w \in \tilde{V}\right] = \sum_{w' \in U_i} x_{w'}^{(u,sh)} = \sum_{w' \in U_i} x_{(w',\ell)}^{(u,sh)} \leq x_{(u,\ell)}^{(u,sh)} \leq 1.$$

The first inequality used Claim 4.3 and $U_i \subseteq \Lambda_u^{\mathrm{leaf}}$. Deconditioning gives (10). □

Lemma 4.14. *For every $\ell \in K$, we have $E[t_\ell | t_\ell \geq 1] \leq h + 1$.*

Proof. In the following, w and w' in summations are over all nodes in V^{leaf} with label ℓ.

$$\begin{aligned}\mathbb{E}[t_\ell | t_\ell \geq 1]^2 \leq \mathbb{E}[t_\ell^2 | t_\ell \geq 1] &= \sum_{w,w'} \Pr[w \in \tilde{V}, w' \in \tilde{V} | t_\ell \geq 1] \\ &\qquad \text{(by Jansen's inequality and the definition of } t_\ell) \\ &= \sum_w \Pr[w \in \tilde{V} | t_\ell \geq 1] \sum_{w'} \Pr[w' \in \tilde{V} | w \in \tilde{V}, t_\ell \geq 1] \\ &= \sum_w \Pr[w \in \tilde{V} | t_\ell \geq 1] \, \mathbb{E}[t_\ell | w \in \tilde{V}] \\ &\qquad \text{(by the definition of } t_\ell \text{ and that } w \in \tilde{V} \text{ implies } t_\ell \geq 1) \\ &\leq (h+1) \sum_w \Pr[w \in \tilde{V} | t_\ell \geq 1] \qquad \text{(by Lemma 4.13)} \\ &= (h+1) \, \mathbb{E}[t_\ell | t_\ell \geq 1] \qquad \text{(by the definition of } t_\ell).\end{aligned}$$

This implies $\mathbb{E}[t_\ell | t_\ell \geq 1] \leq h + 1$. □

Corollary 4.15. $\Pr[t_\ell \geq 1] \geq \dfrac{1}{h+1}$ *for every $\ell \in K$.*

Proof. Notice that $1 = \mathbb{E}[t_\ell] = \mathbb{E}[t_\ell | t_\ell \geq 1] \cdot \Pr[t_\ell \geq 1]$. The corollary follows from Lemma 4.14. □

Thus we have finished the proof of Theorem 4.2.

5 Discussion and Open Problems

In this paper we close the gap on the approximability of DST for the class of quasi-polynomial-time algorithms. However, there is still a huge gap between the lower and upper bounds on approximation ratios for the class of polynomial-time algorithms. In particular, it has been an open problem that perplexes many researchers whether DST admits a polylogarithmic approximation algorithm that runs in polynomial-time. There are both positive and negative evidences that suggest DST may or may not admit such algorithm. On one hand, Rothvoß [30] observes that despite an algorithm based on hierarchical techniques (i.e., Sum-of-Squares) runs in super polynomial-time due to the size of the lifted linear program, the rounding algorithm itself reads only a polynomial number of variables of the fractional solution with high probability. This also applies to all the LP techniques including the folklore path-tree formulation (please see, e.g., [24]). Thus, some may believe that DST admits polylogarithmic approximation algorithms that run in polynomial-time. On the other hand, the factor n^{ϵ}/ϵ that appears in the approximation ratio shows that same behavior as in other problems whose trade-off between approximation ratio and running-time are tight under the Exponential-Time Hypothesis, e.g., *Dense CSP* [27] and *Densest k-Subgraph* [26][3] Our result removes the factor $1/\epsilon$ from the approximation ratio, suggesting that DST may have a different behavior than the other problems mentioned above. Nevertheless, our technique does not yield a good trade-off between approximation ratio and running-time as it requires exactly quasi-polynomial-time to remove such factor. It seems that there is still a major barrier in answering the open question.

Acknowledgement. We would like to thank Uriel Feige for useful discussions over two years, and we would like to thank Jittat Fakcharoenphol for useful discussion on the balanced tree separator.

F. Grandoni is partially supported by the SNSF Grant 200021_159697/1 and the SNSF Excellence Grant 200020B_182865/1.

B. Laekhanukit is supported by the National 1000-Youth Award by the Chinese government. Parts of this work was done when Laekhanukit was at the Weizmann Institute of Science, partially supported by ISF grant #621/12 and I-CORE grant #4/11, while he was visiting the Simons Institute for the Theory of Computing, which was partially supported by the DIMACS/Simons Collaboration on Bridging Continuous and Discrete Optimization through NSF grant #CCF-1740425, and while he was at the Max-Plack Institute for Informatics.

S. Li is supported by NSF grant #CCF-1566356 and #CCF-1717134. Some critical parts of this work were done while Li was visiting the Institute for Theoretical Computer Science at Shanghai University of Finance and Economics.

References

[1] Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 184–193, 1996.

[3] In [27], the trade-off is slightly weaker, say $O(n^{\epsilon^3}/\epsilon)$-approximation ratio versus $n^{1/\epsilon}$-running time.

[2] MohammadHossein Bateni, Moses Charikar, and Venkatesan Guruswami. Maxmin allocation via degree lower-bounded arborescences. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 543–552, 2009.

[3] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1):6:1–6:33, 2013.

[4] Parinya Chalermsook, Fabrizio Grandoni, and Bundit Laekhanukit. On survivable set connectivity. In *SODA*, pages 25–36, 2015.

[5] Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed steiner problems. *J. Algorithms*, 33(1):73–91, 1999.

[6] Chandra Chekuri and Martin Pál. A recursive greedy algorithm for walks in directed graphs. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 245–253, 2005.

[7] Joseph Cheriyan, Bundit Laekhanukit, Guyslain Naves, and Adrian Vetta. Approximating rooted steiner networks. *ACM Transactions on Algorithms*, 11(2):8:1–8:22, 2014.

[8] Eden Chlamtac. Approximation algorithms using hierarchies of semidefinite programming relaxations. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 691–701, 2007.

[9] Marek Cygan, Fabrizio Grandoni, and Monaldo Mastrolilli. How to sell hyperedges: The hypermatching assignment problem. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 342–351, 2013.

[10] Alina Ene, Deeparnab Chakrabarty, Ravishankar Krishnaswamy, and Debmalya Panigrahi. Online buy-at-bulk network design. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 545–562. IEEE Computer Society, 2015.

[11] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004.

[12] Zachary Friggstad, Jochen Könemann, Young Kun-Ko, Anand Louis, Mohammad Shadravan, and Madhur Tulsiani. Linear programming hierarchies suffice for directed steiner tree. In *Integer Programming and Combinatorial Optimization - 17th International Conference, IPCO 2014, Bonn, Germany, June 23-25, 2014. Proceedings*, pages 285–296, 2014.

[13] Naveen Garg, Goran Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group steiner tree problem. *J. Algorithms*, 37(1):66–84, 2000.

[14] Shashwat Garg, Janardhan Kulkarni, and Shi Li. Lift and project algorithms for precedence constrained scheduling to minimize completion time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, New Orleans, Louisiana, USA, January 6-8, 2019*.

[15] Fabrizio Grandoni and Bundit Laekhanukit. Surviving in directed graphs: a quasi-polynomial-time polylogarithmic approximation for two-connected directed steiner tree. In Hatami et al. [18], pages 420–428.

[16] Anupam Gupta, Ravishankar Krishnaswamy, and R. Ravi. Tree embeddings for two-edge-connected network design. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1521–1538, 2010.

[17] Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 585–594. ACM, 2003.

[18] Hamed Hatami, Pierre McKenzie, and Valerie King, editors. *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*. ACM, 2017.

[19] Christopher S. Helvig, Gabriel Robins, and Alexander Zelikovsky. An improved approximation scheme for the group steiner problem. *Networks*, 37(1):8–20, 2001.

[20] Camille Jordan. Sur les assemblages de lignes. *Journal fr die reine und angewandte Mathematik*, 70:185–190, 1869.

[21] Rohit Khandekar, Guy Kortsarz, and Zeev Nutov. Approximating fault-tolerant group-steiner problems. *Theorerical Computer Science*, 416:55–64, 2012.

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

[23] Bundit Laekhanukit. Parameters of two-prover-one-round game and the hardness of connectivity problems. In *SODA*, pages 1626–1643, 2014.

[24] Bundit Laekhanukit. Approximating directed steiner problems via tree embedding. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 74:1–74:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[25] Elaine Levey and Thomas Rothvoss. A (1+epsilon)-approximation for makespan scheduling with precedence constraints using LP hierarchies. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 168–177, 2016.

[26] Pasin Manurangsi. Almost-polynomial ratio eth-hardness of approximating densest k-subgraph. In Hatami et al. [18], pages 954–961.

[27] Pasin Manurangsi and Prasad Raghavendra. A birthday repetition theorem and complexity of approximating dense csps. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 78:1–78:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[28] Dana Moshkovitz. The projection games conjecture and the np-hardness of ln n-approximating set-cover. *Theory of Computing*, 11:221–235, 2015.

[29] Gabriel Robins and Alexander Zelikovsky. Tighter bounds for graph steiner tree approximation. *SIAM J. Discrete Math.*, 19(1):122–134, 2005.

[30] Thomas Rothvoß. Directed steiner tree and the lasserre hierarchy. *CoRR*, abs/1111.5473, 2011.

[31] Alexander Zelikovsky. An 11/6-approximation algorithm for the network steiner problem. *Algorithmica*, 9(5):463–470, 1993.

[32] Alexander Zelikovsky. A series of approximation algorithms for the acyclic directed steiner tree problem. *Algorithmica*, 18(1):99–110, 1997.

[33] Leonid Zosin and Samir Khuller. On directed steiner trees. In David Eppstein, editor, *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA.*, pages 59–63. ACM/SIAM, 2002.

A The Missing Proofs from Section 2

We provide in the section the proof of Lemma 2.1.

Lemma 2.1 (Balanced-Tree-Partition)**.** *For any $n \geq 3$, for any n-vertex tree T rooted at a vertex r, there exists a vertex $v \in V(T)$ such that T can be decomposed into two trees T_1 and T_2 rooted at r and v, respectively, in such a way that $E(T_1) \uplus E(T_2) = E(T)$, $V(T_1) \cup V(T_2) = V(T)$ and $V(T_1) \cap V(T_2) = \{v\}$ and $|V(T_1)|, |V(T_2)| < 2n/3 + 1$. In other words, T_1 and T_2 are sub-trees that form a balanced partition of (the edges of) T.*

This can be proved via the well-known Tree-Separator Theorem:

Theorem A.1 (Tree-Separator Theorem [20])**.** *For any n-vertex tree T, there is a vertex $v \in V(T)$ such that removing v from T results in a graph where each (connected) component contains at most $n/2$ vertices.*

Proof of Lemma 2.1. The proof follows straightforward from Theorem A.1. We may assume that T is an out-arborescence rooted at a vertex r. We pick a vertex v as in Theorem A.1 (it could be the case that $v = r$). Then we have weakly connected subgraphs of $T \setminus v$, say $H_1, \ldots, H_q$. It is not hard to see that each subgraphs H_i, for $i \in [q]$, is an arborescence.

We start from $T' = \emptyset$. As long as there is an $H_i \not\subseteq T'$ such that $|V(H_i)| + |V(T')| < 2n/3$, we add H_i to T'. Let T_1 contain v, T' and the edges between v and T'; let T_2 contain v, sub-graphs H_i that are not in T', and the edges joining v and these sub-graphs.

It follows from the construction that both T_1 and T_2 are sub-arborescences of T that have only v as a common vertex and that $T_1 \cup T_2 = T$. Renaming T_1 and T_2 in the end of the proof if necessary so that T_1 is rooted at r. Notice that $|V(T')| < 2n/3$, implying that $V(T_1) < 2n/3 + 1$. It is sufficient to show that $|V(T')| > n/3 - 1$, which will imply $V(T_2) < 2n/3 + 1$ since $|V(T')| + |V(T_2)| = n$.

Suppose $|V(T')| < n/3 - 1$. Then every component H_i not included in T' must contain more than $n/3 + 1$ vertices. So there are at most two such components. Also, there can not be just one such component since otherwise it has size more than $2n/3 > n/2$, a contradiction. So, there are exactly two components not in T', and one of them, say H_i, has at most $\dfrac{n - 1 - |V(T')|}{2}$ vertices. But then $|V(T')| + |V(H_i)| \leq \dfrac{n - 1 + |V(T')|}{2} \leq 2n/3 - 1 < 2n/3$, a contradiction. □

Claim 2.2. *For some $x \in \mathrm{SA}(\mathcal{P}, R)$ with $R \geq 2$, the following statements hold:*

(2.2a) $x_S \geq x_{S'}$ *for every* $S \subseteq S' \in \binom{[n]}{\leq R}$.

(2.2b) If $x_i = 1$ for some $i \in [n]$, then $x_{\{i,i'\}} = x_{i'}$ for every $i' \in [n]$.

(2.2c) If every $\hat{x} \in \mathcal{P}$ has $\hat{x}_i \leq \hat{x}_{i'}$, then $x_{\{i,i'\}} = x_i$.

Letting x' be obtained from x by conditioning on some event $i \in [n]$, the following holds:

(2.2d) $x'_i = 1$.

(2.2e) $x' \in \mathrm{SA}(\mathcal{P}, R - 1)$.

(2.2f) If $x_{i'} \in \{0, 1\}$ for some $i' \in [n]$, then $x'_{i'} = x_{i'}$.

Proof. Let $x \in \mathrm{SA}(\mathcal{P}, R)$ for some $R \geq 2$.

(2.2a) Consider the case where $S' = S \cup \{i\}$ for some $i \notin S$. Linearizing the constraint $x_i \leq 1$ multiplied by $\sum_{i' \in S} x_i$ gives the constraint $x_{S'} \leq x_S$.

(2.2b) Multiplying $1 - x_i \geq 0$ and $1 - x_{i'} \geq 0$ and linearizing the product gives the constraint $1 - x_i - x_{i'} + x_{\{i,i'\}} \geq 0$. Then $x_i = 1$ implies $x_{i'} \leq x_{\{i,i'\}}$. But $x_{i'} \geq x_{\{i,i'\}}$; thus $x_{i'} = x_{\{i,i'\}}$.

(2.2c) $x_i \leq x_{i'}$ is implied by the constraints for the basic polytope. Multiplying both sides by x_i and linearizing the constraint gives $x_i \leq x_{\{i,i'\}}$; but $x_i \geq x_{\{i,i'\}}$ by (2.2a). Thus $x_i = x_{\{i,i'\}}$.

Now, let x' be obtained from x by conditioning on some event $i \in [n]$.

(2.2d) By definition of the conditioning operation, we have $x'_i = \frac{x_{\{i\}\cup\{i\}}}{x_i} = \frac{x_i}{x_i} = 1$.

(2.2e) $x'_\emptyset = \frac{x_{\emptyset\cup\{i\}}}{x_i} = \frac{x_i}{x_i} = 1$. (1) on x' for j, S and T is implied by (1) on x for $j, S \cup \{i\}$ and T.

(2.2f) If $x_{i'} = 0$, then $x'_{i'} = \frac{x_{\{i',i\}}}{x_i} = 0$ since $x_{\{i',i\}} \leq x_{i'} = 0$. Consider the case $x_{i'} = 1$. Property (2.2b) says $x_{\{i,i'\}} = x_i$, implying $x'_{i'} = \frac{x_{\{i,i'\}}}{x_i} = 1$. □

B Missing Proofs from Section 3

Claim 3.2. *τ^* is a full binary decomposition tree of height $O(\log k)$ and cost* opt *that involves all vertices in K. Moreover, for every $v \in K$, there is exactly one leaf β of τ^* with* $\mathrm{tail}(e_\beta) = v$.

Proof. Clearly, τ^* is a full binary tree. It has depth $O(\log k)$ since $|V(T^*)| \leq 2k$ and the size of $|V(T)|$ goes down by a constant factor in each level of the recursion for cstr-opt-dcmp-tree. It is easy to see that $(e_\beta)_{\beta \text{ is leaf of } \tau^*}$ is a 1-to-1 mapping from leaves of τ^* to $E(T^*)$, where an edge $(u,v) \in E(T^*)$ corresponds to a leaf β of τ^* with $e_\beta = (u,v)$. This holds as $E(T_1)$ and $E(T_2)$ produced in Step 4 form a partition of $E(T)$, and in Step 1 the leaf-node β created has $e_\beta = (u,v)$. Thus, $\mathrm{cost}(\tau^*) = \sum_{\beta \text{ a leaf of } \tau^*} c(e_\beta) = \sum_{e \in E(T^*)} c(e) = \mathrm{opt}$. Since every terminal $v \in K$ has in-degree exactly 1 in T^*, there is exactly one leaf $\beta \in V(\tau^*)$ with $\mathrm{tail}(e_\beta) = v$. In particular, all terminals in K are involved in τ^*.

A simple observation is that any tree τ^* returned by the procedure cstr-opt-dcmp-tree(T) will have $\mu_{\mathrm{root}(\tau)} = \mathrm{root}(T)$. Properties (3.1a) and (3.1b) hold trivially. Thus, to show that τ^* is indeed a decomposition tree, it suffices to prove Property (3.1c).

Focus on a node α created in Step 3 in some recursion of cstr-opt-dcmp-tree; we shall prove Property (3.1c) for this α. Focus on the moment before we return the tree in Step 6 in the recursion. Let $\alpha_1 = \mathrm{root}(\tau_1)$ and $\alpha_2 = \mathrm{root}(\tau_2)$ be the two children of α. Then we have $\mu_{\alpha_1} = \mathrm{root}(T_1) = \mathrm{root}(T) = \mu_\alpha$ and $\mu_{\alpha_2} = \mathrm{root}(T_2)$. If $\mathrm{root}(T_2) = \mathrm{root}(T)$, then $\mu_{\alpha_2} = \mu_\alpha$ and there is nothing to prove. Thus, we assume $\mathrm{root}(T_2) \neq \mathrm{root}(T)$. Then $\mathrm{root}(T_2) \in V(T_1)$, and it has exactly one incoming edge in T_1. By our construction, there will be a leaf node $\beta \in V(\tau_1)$ with e_β being the edge and thus $\mathrm{tail}(e_\beta) = \mathrm{root}(T_2)$. So $\mu_{\alpha_2} = \mathrm{root}(T_2)$ is involved in τ_1. Thus, Property (3.1c) holds. □

Lemma 3.3. *Given a decomposition tree τ that involves all terminals in K, we can efficiently construct a directed Steiner tree T in G connecting r to all terminals in K with cost at most* $\mathrm{cost}(\tau)$.

Proof. We simply let T contain the edges e_α for all leaves α of τ. Then the cost of T is exactly $\mathrm{cost}(\tau)$. We shall show that T contains a path from r to every terminal $v \in K$. At the end of the proof, we can remove edges in T so that T forms an out-arborescence rooted at r.

Given a node α of τ, let $H_\alpha := (V, \{e_\beta : \beta \text{ is a leaf of } \tau[\alpha]\})$. We will show the following:

For every $\alpha \in V(\tau)$, H_α contains a path from μ_α to every vertex v involved in $\tau[\alpha]$. (*)

Since $E(T) = E(H_{\text{root}(\tau)})$, $\mu_{\text{root}(\tau)} = r$ and every terminal $v \in K$ is involved in τ, applying (*) to root(τ) gives that T contains a path from r to every terminal in K, which finishes our proof.

We prove (*) by induction from the bottom to the top of the tree τ. If α is a leaf, then H_α contains the edge e_α, only head(e_α) and tail(e_α) are involved, and $\mu_\alpha = \text{head}(e_\alpha)$. Thus, (*) holds.

Now consider an internal node α in τ, and assume (*) holds for every child α' of α. Focus on any vertex v involved in $\tau[\alpha]$. If $v = \mu_\alpha$, then trivially there is a path from μ_α to v in H_α. Otherwise, $v = \text{tail}(e_\beta)$ for some leaf β of $\tau[\alpha]$. Let α_2 be the child of α such that $\beta \in V(\tau[\alpha_2])$. By induction hypothesis, there is a path from μ_{α_2} to v in H_{α_2}. If $\mu_{\alpha_2} = \mu_\alpha$, there is a path from μ_α to v in $H_{\alpha_2} \subseteq H_\alpha$. Otherwise, by Property (3.1c), there is a child α_1 of α such that $\mu_{\alpha_1} = \mu_\alpha$ and μ_{α_2} is involved in $\tau[\alpha_1]$. By induction hypothesis, there is a path from $\mu_\alpha = \mu_{\alpha_1}$ to μ_{α_2} in H_{α_1}. Since H_α contains both H_{α_2} and H_{α_1}, there is a path from μ_α to v in H_α. So, (*) holds. □

Claim 3.6. $\mathbf{T}^0$ *is a rooted tree with* $n^{O(\log^2 k/\log\log k)}$ *vertices and height* $O(\bar{h}/g) = O(\log k/\log\log k)$, *where* $n = |V(G)|$.

Proof. The height of T^0 is easily seen to be $O(\bar{h}/g) = O(\log k/\log\log k)$. The number of children of a p-node is dominated by the number of different twigs with a specific μ value for the root. This is at most $2^{2^g} \cdot n^{2\cdot 2^g} \le n^{2^{g+2}} = n^{O(\log k)}$. [4] The number of children of a q-node is at most $2^g = O(\log k)$. Thus, the number of nodes in T^0 is at most $\left((\log k) n^{O(\log k)}\right)^{O(\log k/\log\log k)} = n^{O(\log^2 k/\log\log k)}$. □

Lemma 3.7. $\mathbf{T}^*$ *is a label-consistent sub-tree of* $\mathbf{T}^0$ *with cost exactly* $\text{cost}(\tau^*) = \text{opt}$. *Moreover, all global labels in* K *are supplied by* $\mathbf{T}^*$.

Proof. We define a collapsing operation over a rooted tree $\mathbf{T}$ as follows. Given an internal node in $\mathbf{T}$ with exactly one child, collapsing the node means removing the node and directly connect its child to its parent. If the node is the root of $\mathbf{T}$, we then simply remove the root. It is easy to see that $\mathbf{T}^*$ satisfies the following properties:

(A1) Every p-node in $\mathbf{T}^*$ has exactly one child which is a q-node.

(A2) If a q-node is in $\mathbf{T}^*$, then all its children in $\mathbf{T}^0$ are in $\mathbf{T}^*$.

(A3) Let $\tilde{\mathbf{T}}$ be the tree obtained from $\mathbf{T}^*$ by collapsing all p-nodes. Then $\tilde{\mathbf{T}}$ is *isomorphic* to $\mathbf{H}$: replacing each node q in $\tilde{\mathbf{T}}$ with η_q gives $\mathbf{H}$.

With this correspondence, it is obvious that $\text{cost}(\mathbf{T}^*) = \text{cost}(\tau^*) = \text{opt}$: this holds since the cost of a q node is exactly the total cost of leaves of τ^* that are in η_q. We then show that $\mathbf{T}^*$ is indeed label-consistent. Notice that each p-node in $\mathbf{T}^*$ has exactly 1 child in $\mathbf{T}^*$, and so the demand label for a p-node is satisfied. For a node q in $\mathbf{T}^*$, all the demand labels added to dem(q) in Loop 6 are satisfied since all children of q are included in $\mathbf{T}^*$.

Now focus on a label ℓ' added to dem(q) in an iteration of Loop 12; let $\eta, \alpha, \alpha_1, \alpha_2$ be the values of the correspondent variables in the end of the iteration. Since we assumed a label ℓ' was created

[4] We first describe the shape of the tree. The perfect binary-tree of depth g contains $2^g - 1$ internal nodes we just need to specify whether each internal node has children or not. Now each node can have n^2 different choices for its μ and e values.

and added to $\mathrm{dem}(q)$ in this iteration, we have $\mu_{\alpha_2} \neq \mu_\alpha$. As τ^* is a valid decomposition tree, there is a leaf β' of $\tau^*[\alpha_1]$ such that $\mathrm{tail}(e_{\beta'}) = \mu_{\alpha_2}$. If this leaf β' is in η then it is in $\eta[\alpha_1]$; in this case the label ℓ' can not be created. So, β' is not in η, which means there is a leaf β of $\eta[\alpha_1]$ with e_β undefined, a twig $\eta' \in \mathbf{H}[\eta]$ with β' being a leaf of η'. By the correspondence between $\mathbf{T}^*$ and $\mathbf{H}$ in (A3), there is a leaf β in $\eta[\alpha_1]$ with e_β undefined, and a node $q' \in V(\mathbf{T}^q_\beta) \cap V(\mathbf{T}^*)$ such that $\eta_{q'}$ contains a leaf β' with $e_{\beta'}$ defined and $\mathrm{tail}(e_{\beta'}) = \mu_{\alpha_2}$. Thus, the label ℓ' will be satisfied by this q'.

Finally, all the global demand labels K are provided by $\mathbf{T}^*$: for every terminal v, τ^* contains a leaf β with $\mathrm{tail}(e_\beta) = v$ this β will appear in some twig η and the node q with $\eta_q = \eta$ will provide the label v. □

Lemma 3.8. *Given any feasible solution* $\mathbf{T}$ *to the LCST instance* $\mathbf{T}^0$*, in time* $\mathrm{poly}(|V(\mathbf{T})|)$ *we can construct a decomposition tree* τ *with* $\mathrm{cost}(\tau) = \mathrm{cost}(\mathbf{T})$*. Moreover, if a global label* $v \in K$ *is supplied by* $\mathbf{T}$*, then* τ *involves* v*.*

Proof. We pick the twigs η_q over all nodes q in $\mathbf{T}$. For a technical issue, we also pick a singular *root-twig* α with $\mu_\alpha = r$. Then our decomposition tree τ is constructed by taking the collection $\mathcal{C}$ of twigs we picked, identifying some pairs of nodes in these twigs naturally. We shall make sure that when we identify two nodes, they will have the same μ-value and they do not have e values.

Focus on a node p in $\mathbf{T}$. Let q be the parent of p and β be the leaf of η_q such that $p = \mathrm{root}(\mathbf{T}^q_\beta)$; If $p = \mathrm{root}(\mathbf{T}^0)$, then q is not defined and we let β be the node in the root-twig. Then for every child q' of p in $\mathbf{T}$, we identify $\mathrm{root}(\eta_{q'})$ with β. Clearly we have $\mu_{\mathrm{root}(\eta_{q'})} = u_p = \mu_\beta$. e_β is not defined since otherwise $\mathbf{T}^q_\beta$ does not exist. $e_{\mathrm{root}(\eta_{q'})}$ is not defined either since $\eta_{q'}$ is a non-singular twig.

This finishes the construction of τ. We need to show that τ is a decomposition tree. $\mathrm{root}(\tau)$ is the root of the root-twig and thus we have $\mu_{\mathrm{root}(\tau)} = r$. For each leaf node β with e_β undefined in any twig η_q in our collection $\mathcal{C}$, $p := \mathrm{root}(\mathbf{T}^q_\beta)$ must be in $\mathbf{T}$ as all children of q should be in $\mathbf{T}$ in order to make it label-consistent. The label for p must be satisfied by one of its children. Thus we must have identified β with the root of some non-singular twig in $\mathcal{C}$. Thus e values are defined for exactly the set of leaves of τ. Clearly, for a leaf β' of τ we have $\mu_{\beta'} = \mathrm{head}(e_{\beta'})$; so Property (3.1b) also holds.

We then prove Property (3.1c) for an internal node α of τ, and a child α_2 of τ such that $\mu_{\alpha_2} \neq \mu_\alpha$. The edge (α, α_2) must be in η_q for some q in $\mathbf{T}$. Let α_1 be the other child of α in η_q. So, we have $\mu_{\alpha_1} = \mu_\alpha$ by the definition of a twig. If there is a leaf β of $\eta_q[\alpha_1]$ with e_β defined and $\mathrm{tail}(e_\beta) = \mu_{\alpha_2}$, then Property (3.1c) holds for this α and α_2. Otherwise in the iteration of Loop 12 in cstr-label-tree for this α, α_1 and α_2, we have created a label ℓ'. In order for this ℓ' to be satisfied, there must be a leaf β of $\eta_q[\alpha_1]$ with e_β undefined, and a twig $\eta' \in \mathbf{H}[\eta_q]$ that contains a leaf β' with $e_{\beta'}$ defined and $\mathrm{tail}(e_{\beta'}) = \mu_{\alpha_2}$. Thus this β' will be a leaf node of $\tau[\alpha_1]$; thus Property (3.1c) holds.

The cost of $\mathbf{T}$ is exactly $\mathrm{cost}(\tau)$ since every q-node of $\mathbf{T}$ is correspondent to a twig η_q with cost being the cost of leaves in η_q. If a global demand label $v \in K$ is provided by $\mathbf{T}^*$, then some node q with η_q containing a leaf β with $\mathrm{tail}(e_\beta) = v$ will be in $\mathbf{T}$, and our τ will contain the leaf β and thus v will be involved in τ. □

C Hardness of DST for the Class of Quasi-Polynomial-Time Algorithms

In this section, we present the hardness result for the Directed Steiner Tree problem for the class of quasi-polynomial-time algorithms.

Our hardness result is a refinement of the hardness construction of Halperin and Krauthgamer [17]. To avoid repeating all the proofs in [17], it suffices for us to consider the size of the construction. It is worth remarking that the hardness result of Halperin and Krauthgamer is designed for an instance of the *group Steiner tree problem* (GST) on a tree. To be formal, GST is defined as follows.

Definition C.1. In GST, we are given an n-vertex **undirected** graph G with edge-costs, a root vertex r and a collection of subsets $S_1, \ldots, S_k$ of vertices (*groups*); the goal is to find a minimum-cost subgraph that contains a path from the root to at least one vertex of each group.

It can be seen that GST is a special case of DST. One can reduce GST to DST by first making G bi-directed by making two copies for each of G, one for each direction, and then add a terminal t_i, for each group S_i, with zero cost edges directed from every vertex of S_i to t_i. Thus, we will focus on the construction and the size of the tree constructed in [17]. The parameter that we are interested in is the number of the group (as we claim the lower bound of $\Omega(\log^2 k/\log\log k)$).

The starting point of the reduction is the *Label-Cover* problem defined below.

Definition C.2 (Label-Cover (a.k.a. Projection Game))**.** Let $G = (U, W; E)$ be a bipartite (directed) graph on n vertices and m edges. Let Σ be a set of labels (or alphabet). Each edge $(u, w) \in E$ (where $u \in U$ and $w \in W$) of the graph G is associated with a projection $\pi_{uw} : \Sigma \to \Sigma$. A labeling f is an assignment $f : U \cup W \to \Sigma$ that assigns one label from Σ to each vertex of G. The labeling f is said to *cover* an edge $(u, w) \in E(G)$ if $\pi_{uw}(f(u)) = f(w)$. The goal in the Label-Cover problem is to find a labeling that covers the maximum number of edges.

To the best of our knowledge, the hardness factor $\log^{2-\epsilon} k$, for any $\epsilon > 0$, is the best one could prove under the standard assumption NP $\not\subseteq$ ZPTIME$(n^{\text{polylog}(n)})$. To show a strong hardness, we need to assume a strongly: (1) The *Exponential-Time Hypothesis* (ETH) for k-SAT and (2) The *Projection Games Conjecture*.

Hypothesis C.3 ((randomized) Exponential-Time Hypothesis for k-SAT)**.** *For any constant $k > 0$, there exists a constant c_k such that k-SAT admits no (randomized) $2^{c_k n}$-time algorithm. In particular, there is no $2^{o(n)}$-time algorithm that solves SAT.*

Hypothesis C.4 (Projection Games Conjecture [28])**.** *There exists a constant $c > 0$ such that, for every $\epsilon > 1/n^c$, a SAT instance ϕ on input of size n can be efficiently reduced to a Label-Cover instance on a* poly$(1/\epsilon)$*-regular bipartite graph with $n^{1+o(1)}$ veritces over a set of label of size* poly$(1/\epsilon)$ *in such a way that*

- **Yes-Instance:** *If ϕ is satisfiable, then there exists a labeling that covers all the edges of G.*
- **No-Instance:** *If ϕ is not satisfiable, then there exists no labeling that covers more than ϵ fraction of the edges of G.*

Combining the two hypotheses, we may assert that, for any $1 < \varepsilon \leq c$ (for c from Hypothesis C.4), there is no 2^{n^ε}-time algorithm that approximates the Label-Cover problem to within a factor of n^ε. In fact, we do not need the full power of ETH and may weaken the assumption as below.

Hypothesis C.5 (ETH for Projection Games)**.** *Unless* NP $\subseteq \cap_{\varepsilon>0}$ZPTIME$(2^{n^\varepsilon})$*, there exists a constant $0 < \epsilon^* $ such that, for any constant $0 < \epsilon \leq \epsilon^*$, there exist constants c_ϵ, d_ϵ and $\delta_\epsilon \leq \epsilon$ depending on ϵ such that, the Label-Cover problem on a n^{d_ϵ}-regular bipartite graph G with n vertices and a set of labels of size n^{c_ϵ} admits no $2^{n^{\delta_\epsilon}}$-time algorithm that distinguishes the following two cases:*

- **Yes-Instance:** *There exists a labeling that covers all the edges of G.*
- **No-Instance:** *There exists no labeling that covers more than $1/n^\epsilon$ fraction of the edges of G.*

In particular, the Label-Cover problem admits no $2^{n^{\delta_\epsilon}}$-time n^ϵ-approximation algorithm for any $0 < \epsilon \leq \epsilon^*$ unless every NP problem admits a randomzied algorithm that runs in time $2^{n^{o(1)}}$.

Applying Hypothesis C.5 to the proof in [17] with slightly different parameters setting immediately gives us the approximation hardness of $\Omega(\log^2 k/\log\log k)$ for the directed Steiner tree problem. To avoid overwhelming readers with too much information (and avoid repeating the proof in [17]), we re-state the reduction in [17] as below.

Theorem C.6 ([17])**.** *Consider an instance ψ of the Label-Cover problem on a Δ-regular n-vertex bipartite graph with a set of labels Σ of size σ. For any parameter $1 \leq h \leq O(\log^2 n)$, there exists a randomized reduction from ψ to an instance of the Group Steiner Tree problem on a tree T with costs on edges and with k groups such that $|V(T)| = (\sigma n)^h$ and $k = \Delta n^h$. Moreover, with high probability, the following holds:*

- **Yes-Instance:** *If there exists a labeling that covers all the edges of G, then there exists a feasible solution $T' \subseteq T$ to the Group Steiner Tree problem with $\text{cost}(T') = h^2$.*
- **No-Instance:** *If there is no labeling that covers more than γ fraction of the edges of G, then every feasible solution $T' \subseteq T$ to the Group Steiner Tree problem must have cost at least $\text{cost}(T') \geq \min\{\gamma^{-1/2}h, \Omega(h\log k)\}$.*

We will now prove our hardness result, which can be considered as a corollary of Theorem C.6.

Theorem C.7. *Suppose Hypothesis C.5 is true, i.e., $\text{NP} \not\subseteq \bigcap_{\varepsilon>0} \text{ZPTIME}(2^{n^\varepsilon})$ and the Projection Games Conjectures holds. Then there exists no quasi-polynomial-time algorithm for the Directed Steiner Tree problem on a graph with N vertices (resp., the Group Steiner tree problem on a tree with N vertices) that yields an approximation ratio of $o(\log^2 k/\log\log k)$ or $o(\log^2 N/\log\log N)$.*

Proof. Let h be a parameter as in Theorem C.6 (which we will specify later). We first take an instance of the Label-Cover problem on a bipartite graph $G = (U, W; E)$ on n vertices and with the set of labels Σ from Hypothesis C.5. Thus, we have an instance of the Label-Cover problem with parameter $\Delta = n^{d_\epsilon}$ (which is the degree of G) and the set of labels of size $\sigma = n^{c_\epsilon}$, and $\gamma = 1/n^\epsilon$. (The parameter γ is usually called the soundness error in literature.)

Now we choose the parameter $h = n^z$, for some constant $0 < z < 1$, that will be specified later. Observe that

$$k = \Delta n^h \implies \log k = h \cdot \log(\Delta \cdot n) = h \cdot (\log\Delta + \log n) = h \cdot \Theta(\log n) \implies h = \frac{\log k}{\Theta(\log n)}$$

Moreover, it is not hard to see that $\log n = \Theta(\log\log k)$ because

$$k = \Delta n^h = n^{h+d_\epsilon} = 2^{(n^z+d_\epsilon)\log_2 n} \implies \log\log k = \Theta(\log n)$$

Therefore, we have the hardness gap of $\Omega(\log^2 k/\log\log k)$ as claimed. Observe that $\log|V(T)| = \Theta(\log k)$. Thus, we have the same hardness gap for both in terms of k and that of $N = |V(T)|$.

Next we prove the running-time lower bound. Assume for a contrary that there exists an algorithm for GST on the tree T that runs in time $O(|V(T)|^{\log^\zeta |V(T)|})$, for some constant $\zeta > 0$, and

yields approximation guarantee $o(\log^2 k/\log\log k)$. Then by setting $z < \delta_\epsilon/(3\zeta)$, we would have an algorithm that runs in time

$$\begin{aligned} O(|V(T)|^{\log^\zeta |V(T)|}) &= O(((\sigma n)^h)^{(h\log(\sigma n))^\zeta}) & &= O((\sigma n)^{(h^2\log(\sigma n))^\zeta}) & &= 2^{(O(h^2\log^2(\sigma n)))^\zeta} \\ &= 2^{(O(h\log(\sigma n))^{2\zeta}} & &= 2^{(O(n^{\delta_\epsilon/(3\zeta)}\log(n^{1+c_\epsilon}))^{2\zeta}} & &< 2^{n^{\delta_\epsilon}} \end{aligned}$$

This running time contradicts the statement of Hypothesis C.5. □

D Hardness of the Label-Consistent problem on General Graphs

This section provides the proof for the hardness of the generalization of the Label-Consistent Subtree problem to general graphs, which we may call the *Label-Consistent Subgraph* problem (LCSG).

Definition D.1. In LCSG, the input is an undirected graph $G = (V, E)$ with vertex (or edge) costs with a root vertex r, a set of labels L and a set of global labels $K \subseteq L$. Each vertex of G is associated with a demand function $\text{dem}(v) \subseteq L$ and a service function $\text{ser}(v) \subseteq L$. The goal in LCSG is to find a minimum-cost subgraph $H \subseteq G$ such that the following two properties hold:

- For every global label $t \in K$, there is a path from r to t in H.
- For every vertex $v \in V(H)$ and every label $\ell \in \text{dem}(v)$, there exists a path from v to a vertex w with $\ell \in \text{ser}(v)$.

We will now show that LCSG is at least as hard as the minimum Label-Cover problem. The definition of the minimum Label-Cover problem is slightly different from that of the (maximum) Label-Cover problem; here we are allowed to assign multiple labels to each vertex, but we have to cover all the edges. The formal definition of the minimum Label-Cover problem is defined as below.

Definition D.2 (Minimum Label-Cover (a.k.a. Min-Rep [22])**.** Let $G = (U, W; E)$ be a bipartite (directed) graph on n vertices and m edges. Let Σ be a set of labels (or alphabet). Each edge $(u, w) \in E$ (where $u \in U$ and $w \in W$) of the graph G is associated with a projection $\pi_{uw} : \Sigma \to \Sigma$. A multi-labeling f is an assignment $f : U \cup W \to 2^\Sigma$ that assigns a set of labels from Σ to each vertex of G. The multi-labeling f is said to *cover* an edge $(u, w) \in E(G)$ if there exists a label $a \in f(u)$ and $b \in f(w)$ such that $\pi_{uw}(f(u)) = f(w)$. The cost of the multi-labeling f is $\sum_{v \in U \cup W} |f(v)|$. The goal in the Label-Cover problem is to find a multi-labeling with minimum-cost that covers all the edges.

We remark that the standard hardness of the Label-Cover problem is between the case that the optimal solution is a labeling versus the case that the optimal solution is a multi-labeling.

Theorem D.3 (Hardness of the Label-Consistent problem on General Graphs)**.** *There exists a polynomial-time reduction from an instance ψ of the Label-Cover problem on a bipartite graph $G = (U, W; E)$ with n vertices, m edges and with the set of labels Σ to an instance I of the Label-Consistent Subgraph problem on a graph G' on a set of label L' of size $m + \Sigma$. Moreover, the following holds.*

- **Yes-Instance:** *Suppose there is a labeling f that covers all the edges of G, then there exists a solution to the instance I of LCSG with cost n.*
- **No-Instance:** *Suppose there is no multi-labeling f with cost γn that covers all the edges of G, then any feasible solution to the instance I of LCSG must have cost at least γn.*

In particular, LCSG is at least as hard as the Label-Cover problem with perfect completeness.

Proof. First, take an instance of the Label-Cover problem consisting of a graph $G = (U, W; E)$ with the constraints π_{uw} on edges $(u, w) \in E(G)$ and a set of label L. We first construct a graph G' by adding a root vertex r. Then we add to G' a set of vertices $U' = \{u_a : u \in U, a \in L\}$ and $W' = \{w_b : w \in W, b \in L\}$. For each vertex $u \in U$ (resp., $w \in W$), we denote by $U'(u) = \{u_a : a \in L\}$ (resp., $W'(w) = \{w_b : b \in L\}$) the set of vertices corresponding to a vertex u (resp., w) in G.

We add edges joining r to every vertex of U', and we add an edge $u_a w_b$ to G' if $(u, w) \in E(G)$ and $\pi_{uw}(a) = b$. We set cost of each vertex in $U' \cup W'$ to be one.

Now we define the set of labels of the instance I of LCSG. Let the set of all labels be $L = (U \cup W') \cup \Sigma$, and the set of global labels be $K = U$. We assign the service label $\text{ser}(u_a) = \{u\}$ and the demands $\text{dem}(u_a) = \{w_b \in W' : (u, w) \in E(G) \wedge \pi_{uw}(a) = b\}$, for all vertices $u_a \in U'$. Next we assign the service labels $\text{dem}(w_b) = \{w_b\}$, for all vertices $w_b \in W'$; these vertices have no demands (i.e., $\text{dem}(w_b) = \emptyset$). This completes the construction.

Completeness. Suppose there is a labeling f that covers all the edges of G. Then we choose the root vertex r, all the vertices $u_a \in U'$ such that $f(u) = a$, and all the vertices $w_b \in W'$ such that $f(w) = b$. We denote such a subgraph by $H' \subseteq G'$. By feasibility of f, we know that, for every vertex $u_a \in V(H')$ and for every edge $(u, w) \in E(G)$, there exists a vertex $w_b \in V(H')$ such that $\pi_{uw}(a) = b$; moreover, by construction, H' must contain a path (r, u_a, w_b). Consequently, for every global label $u \in K$, the graph H' has an r, u_a-path, for $a = f(u)$, and for every demand label $w_b \in \text{dem}(u_a)$, we have a u_a, w_b-path (which is just a single edge). Thus, the graph H' is label-consistent and must be a feasible solution to the LCSG instance I with the same cost as f.

Soundness. Suppose there is no multi-labeling of cost less than γn that covers all the edges of G. Then we claim that every feasible solution to the instance I of LCSG must have cost at least γn. Suppose to a contrary that there exists a subgraph $H' \subseteq G'$ that is feasible to the instance I of LCSG, but H' has cost less than γn. Then we can obtain a feasible multi-labeling f by assigning $f(u) = V(H') \cap U'(u)$, for all $u \in U$, and $f(w) = V(H') \cap W'(w)$, for all $w \in W$. We know that, for every vertex $u_a \in V(H')$ and for all edges $(u, w) \in E(G)$, H' must contain a u_a, w_b-path such that $b = f_{uw}(a)$. This means that $a \in f(u)$, $b \in f(w)$ and that $f_{uw}(a) = b$, for every edge $(u, w) \in E(G)$, i.e., f covers all the edges of G. It is not hard to see that f has the same cost as H', i.e., f has cost less than than γn. But, this is a contradiction since any multi-labeling that covers all the edges of G must have cost at least γn. □