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

Fabrizio Grandoni
IDSIA
Switzerland
fabrizio@idsia.ch

Bundit Laekhanukit
Institute for Theoretical Computer Science
Shanghai University of Finance and Economics
China
bundit@sufe.edu.cn

Shi Li
Department of Computer Science and Engineering
University at Buffalo
USA
shil@buffalo.edu

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 subgraph 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 $n^{\mathrm{poly}\log(k)}$. By assuming the Projection Game Conjecture and $\mathrm{NP} \not\subseteq \bigcap_{0<\epsilon<1}\mathrm{ZPTIME}(2^{n^\epsilon})$, 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 *Group Steiner Tree on Trees with Dependency Constraint (GSTTD)* problem. Compared to the classic Group Steiner Tree on Trees problem, in GSTTD we are additionally given some dependency constraints among the nodes in the output tree that must be satisfied. The GSTTD 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 GSTTD 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 GSTTD. 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 guarantee 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.

CCS CONCEPTS

• **Theory of computation**; • **Design and analysis of algorithms**; • **Approximation algorithms analysis**; • **Routing and network design problems**;

KEYWORDS

Directed Steiner Tree, Quasi-Polynomial Time, Sherali-Adams Hierarchy

ACM Reference Format:
Fabrizio Grandoni, Bundit Laekhanukit, and Shi Li. 2019. $O(\log^2 k/\log\log k)$-Approximation Algorithm for Directed Steiner Tree: A Tight Quasi-Polynomial-Time Algorithm. In *Proceedings of the 51st Annual ACM SIGACT Symposium on the Theory of Computing (STOC '19), June 23–26, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3313276.3316349

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 subgraph $H \subseteq G$ 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, 28, 30]), the best known polynomial-time approximation algorithm for this problem could achieve only an $O\left(\frac{k^\epsilon \log k}{\epsilon^2}\right)$ approximation ratio in time $O(n^{1/\epsilon})$ for any $0 < \epsilon \le 1/\log_2 k$, 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

[1]The original paper claims an $O(i^2 k^{1/i})$-approximation in time $O(n^i)$ and an $O(\log^2 k)$-approximation in quasi-polynomial time; however, their result was based on the initial statement of the Zelikovsky's height-reduction theorem in [31], which was

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 [32], Sum-of-Squares (a.k.a. Lasserre) hierarchy [29], Sherali-Adams and Lovász-Schrijver hierarchies [13]. 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 [19] that unless NP $\subseteq$ ZPTIME($n^{\mathrm{polylog}(n)}$), 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 [7]. 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, 13, 29] rely on the well-known Zelikovsky's height-reduction theorem [21, 31]. 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 [19] carries over to GST on trees. We remark that algorithms for many related problems (see, e.g., [11, 16]) 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* $O\left(\frac{\log^2 k}{\log\log k}\right)$*-approximation algorithm for DST with running time* $n^{O(\log^5 k)}$.

By analyzing the proofs in [19], we also show that this bound is asymptotically tight under stronger assumptions; the proof is deferred to the full version of the paper.

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

Our upper bound is based on two main ingredients. The first one is a quasi-polynomial-time approximation-preserving reduction to a novel *Group Steiner Tree on Trees with Dependency Constraints (GSTTD)* problem. In GSTTD we are given an instance of GST on a tree, and additionally we are given many dependency rules of the form (v, S), where v is a vertex in the input tree and S is a subset of descendants of v. The rule (v, S) requires that if v is selected in the output tree, then at least one vertex in S must be selected. The dependency rules will be used to guarantee that a feasible solution induces a valid solution to the original DST problem. In our reduction the tree has size $n^{\mathrm{poly}\log(k)}$ and height $h = O(\log k/\log\log k)$, with k terminals. For a comparison, Zelikovsky's height-reduction theorem [31], 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 GSTTD instances arising from the previous reduction. Here we exploit the LP-hierarchy framework developed by Rothvoß [29] (and later simplified by Friggstad et al. [13]). We define a proper LP relaxation for the problem, and solve an R-level Sherali-Adams lifting of this LP for a parameter $R = \mathrm{poly}\log k$. 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; they correspond to the set of relevant dependency rules that are not satisfied yet. 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 always satisfies the dependency rules. 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 [29], the probability that each terminal 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, 9, 10, 15, 25]).

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 [7], and second is the hierarchy based LP-rounding techniques by Rothvoß [29].

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 a 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).

later found to contain a subtle flaw and was restated by Helvig, Robin and Zelikovsky [21].

We imitate the recursive greedy algorithm by recursively splitting the optimal solution via balanced tree separators. The same approach as in [7], 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 dependency rules specifically to solve this issue.

The dependency rules 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 [29]. While the framework was originally developed for the Sum-of-Squares hierarchy, it was shown by Friggstad et al. [13] 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 dependency rules into account.

1.3 Related Work

We already mentioned some main results about DST and GST. For GST there is a polynomial-time algorithm by Garg et al. [14] 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, 12], 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 [7]; however, their algorithm runs in quasi-polynomial-time. Chekuri and Pal also mentioned the slight improvement of $O(\log n/\log\log n)$ for GST in quasi-polynomial-time using enumeration. One would wish to achieve the same approximation ratio for the case that the input is a tree; nevertheless, all the known $O(\log n/\log\log n)$-approximation algorithms [6, 18] run in quasi-polynomial-time, and it has been an open problem whether there exists a polynomial-time algorithm that yields such approximation ratio (this would imply an improvement for the general case as well).

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. [8] showed that ℓ-DST admits no $2^{\log^{1-\varepsilon} n}$-approximation, for any $\varepsilon > 0$, unless NP $\subseteq$ DTIME($2^{\mathrm{polylog}(n)}$). 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 [16] (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. [17] 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 [22]. 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($2^{\mathrm{polylog}(n)}$).

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 $\mathrm{root}(T)$ 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 $\mathrm{head}(e) = u$ and $\mathrm{tail}(e) = v$ 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 instances obtained from reductions.

Group Steiner Tree on Trees with Dependency Constraint. A new problem we introduce is the Group Steiner Tree on Trees with Dependency Constraint (GSTTD) problem. The input consists of a rooted tree T^0 of size $N = |V(T^0)|$ and height h, a node cost vector $c \in \mathbb{R}_{\geq 0}^{V(T^0)}$, a set K of k terminals, and a collection of k subsets (called *groups*) $\{\mathcal{K}_t\}_{t\in K}$, one for each terminal.

If our goal is to find the minimum-cost subtree T of T^0 containing $\mathrm{root}(T^0)$ and at least one vertex from each group $\mathcal{K}_t$, then the problem is exactly GST on trees. In GSTTD, additionally we are given a set L of *dependency rules*. Each rule is of the form (v, S), where $v \in V(T^0)$ and S is a subset of *descendants* of v in T^0. The rule (v, S) requires that if v is chosen by T, then at least one vertex in S must be chosen by T. We say this rule is *associated* with the vertex v. The goal of the GSTTD problem is then the same as that of GST on trees, with the requirement that T must obey all the rules.

In Section 4, we give an $O(h \log k)$-approximation algorithm in running time $(shN)^{O(sh^2)}$-time for the GSTTD problem, where s is the maximum number of rules associated with any given vertex. Thus, we require s to be small in order to derive a quasi-polynomial-time algorithm; fortunately, this is the case for the GSTTD instance reduced from DST.

Balanced Tree Partition. A main tool in our reduction is the following standard balanced-tree-partition lemma, whose proof will be deferred to the full version of the paper.

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:

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

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 :

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

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 polytope 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. The following standard facts hold.

Claim 2.2. *For any $x \in \mathrm{SA}(\mathcal{P}, R)$ 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.

All the proofs that are omitted due to space constraints will appear in the full version of the paper.

3 REDUCING DIRECTED STEINER TREE TO GROUP STEINER TREE ON TREES WITH DEPENDENCY CONSTRAINT

In this section, we present a reduction from DST to GSTTD. 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 minimum 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 GSTTD 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 a dependency rule.

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 GSTTD 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

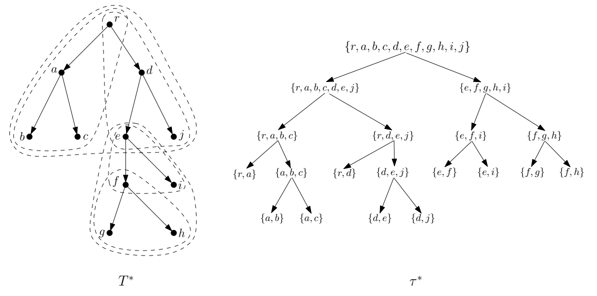

Figure 1: An example for construction of τ^*. For each node τ^*, the set denotes the vertices in the sub-tree of T^* correspondent to the node; the μ value of the node is the first element in the set. For a leaf node, its e value is the edge from the first element to the second element in the set.

that $v = \text{tail}(e_\beta)$. 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 $O(\log k)$ that involves all terminals. 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 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 $|V(T^*)| \le 2k$. 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 $\tau^* \leftarrow$ 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.

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

```
1: if $T$ consists of a single edge $(u, v)$ then
2:     return a node $\beta$ with $\mu_\beta = u$ and $e_\beta = (u, v)$
3: else
4:     create a node $\alpha$ with $\mu_\alpha = \text{root}(T)$
5:     apply Lemma 2.1 to find two rooted trees $T_1$ and $T_2$ with root($T_1$) = root($T$)
6:     $\tau_1 \leftarrow$ cstr-opt-dcmp-tree($T_1$)
7:     $\tau_2 \leftarrow$ cstr-opt-dcmp-tree($T_2$)
8:     return tree rooted at $\alpha$ with two sub-trees $\tau_1$ and $\tau_2$
```

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

From Decomposition Tree to Directed Steiner Tree. For the other direction, we proof the following lemma in the full version of the paper:

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 GSTTD problem.

3.2 Construction of GSTTD Instance

Let $\bar{h}$ be the $O(\log k)$ term in Claim 3.2 that upper bounds the height of τ^*. To save the factor of $\log\log k$ in the approximation ratio, we shall "collapse" every $g := \lceil \log_2 \log_2 k \rceil$ levels of a decomposition tree into one level. 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 GSTTD instance $\mathbf{T}^0$ is constructed by calling $\mathbf{T}^0 \leftarrow$ cstr-gsttd-inst($r, 0$), where cstr-gsttd-inst is defined in Algorithm 2. See Figure 2 for illustration of one recursion of cstr-gsttd-inst.

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: replace η with a node q with $\eta_q = \eta$, and insert a virtual parent p of q with $u_p = \mu_{\text{root}(\eta)}$ between this q and its actual parent. Then idea behind the construction of $\mathbf{T}^0$ is that no matter what the decomposition tree τ is, we can find copy of the resulting tree in $\mathbf{T}^0$. So there are two types of nodes in $\mathbf{T}^0$: (1) p-nodes are those created in Step 1, which correspond to the virtual parents created and (2) q-nodes are those created in Step 4, which correspond to the actual twigs of $\mathbf{H}$. 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 p-nodes will be useful when we define the dependency rules L, which are used to guarantee that $\mathbf{T}$ will correspond to a valid τ. In particular, the rules created in Step 14 guarantee that if p is selected then so is at least one child of p (for the optimum solution τ^*, exactly one is chosen). The rules created in Step 9 for a fixed q guarantees that if q is selected, then all children of q must be selected, while the rules created in Step 13 guarantee Property (3.1c) of τ. The collection of groups $\{\mathcal{K}_v\}_{v \in K}$ are constructed in Step 6, where we add a node q to a group $\mathcal{K}_v$ if η_q contains a leaf β with $\text{tail}(e_\beta) = v$.

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

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 $n = |V(G)|$.*

It is easy to see that a node p will be associated with exactly one rule created in Step 14, while a node q can be associated with up to $O(2^g) = O(\log k)$ rules. So, the parameter s, i.e, the maximum number of rules a node is associated with, is $O(\log k)$.

Algorithm 2 cstr-gsttd-inst(u, j)

```
1:  create a new node $p$ with $c_p = 0$ and $u_p = u$
2:  if $j = \lceil \bar{h}/g \rceil$ then return $p$
3:  for each possible non-singular twig $\eta$ with $\mu_{\mathrm{root}(\eta)} = u$ do
4:      create a new child $q$ of $p$ with $c_q = \sum_{\text{leaf } \beta \text{ of } \eta: e_\beta \text{ defined}} c(e_\beta)$ and $\eta_q = \eta$
5:      for every leaf $\beta$ of $\eta$ with $e_\beta$ defined do
6:          if $\mathrm{tail}(e_\beta) \in K$ then add the node $q$ to the group $\mathcal{K}_{\mathrm{tail}(e_\beta)}$
7:      for every leaf $\beta$ of $\eta$ with $e_\beta$ undefined do
8:          $\mathbf{T}^q_\beta \leftarrow$ cstr-gsttd-inst$(\mu_\beta, j+1)$, let $\mathrm{root}(\mathbf{T}^q_\beta)$ be a child of $q$
9:          add a new rule $\left(q, \left\{\mathrm{root}(\mathbf{T}^q_\beta)\right\}\right)$ to $L$
10:     for every internal node $\alpha$ of $\eta$ do
11:         let $\alpha_1$ be a child of $\alpha$ with $\mu_{\alpha_1} = \mu_\alpha$ and $\alpha_2$ be the other child
12:         if $\mu_{\alpha_2} \neq \mu_\alpha$ and $\nexists$ leaf $\beta$ of $\eta[\alpha_1]$ with $e_\beta$ defined and $\mathrm{tail}(e_\beta) = \mu_{\alpha_2}$ then
13:             add the rule $(q, S)$ to $L$, where
                $$S := \left\{q' : \exists \beta \in \eta_{\alpha_1} \text{ with } e_\beta \text{ undefined}, q' \in \mathbf{T}^q_\beta \text{ contains a leaf } \beta' \text{ with } e_{\beta'} \text{ defined and } \mathrm{tail}(e_{\beta'}) = \mu_{\alpha_2}\right\}.$$
14: add the rule $(p, \{q : q \text{ is a child of } p\})$ to $L$
15: return the tree rooted at $p$
```

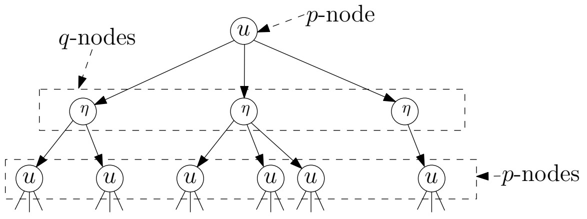

Figure 2: Illustration for one recursion of cstr-gsttd-inst**. Each p-node has a u_p value, and each q-node is associated with a twig η_q with $\mu_{\mathrm{root}(\eta_q)}$ being the u value of its parent p-node. Each child p' of q corresponds to a leaf β of η_q with e_β undefined.**

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

From Decomposition Tree to GSTTD. To show that there is a valid 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 $\mathrm{root}(\eta')$ is a leaf in η. So, $\mathbf{H}$ has depth at most $\lceil \bar{h}/g \rceil$.

$\mathbf{T}^*$ can be found naturally by calling cstr-opt-gsttd$(\mathrm{root}(\mathbf{T}^0), \mathrm{root}(\mathbf{H}))$ (with $\mathbf{T}^*$ being empty initially), where cstr-opt-gsttd 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 $u_p = \mu_{\mathrm{root}(\eta)}$: The root recursion satisfy this condition since $u_{\mathrm{root}(\mathbf{T}^0)} = \mu_{\mathrm{root}(\mathrm{root}(\mathbf{H}))} = r$; in

Algorithm 3 cstr-opt-gsttd(p, η)

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

Step 4, we also have $u_{\mathrm{root}(\mathbf{T}^q_\beta)} = \mu_\beta = \mu_{\mathrm{root}(\eta')}$. 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.

Lemma 3.7. $\mathbf{T}^*$ *is a sub-tree of* $\mathbf{T}^0$ *obeying all the dependency rules and having cost exactly* $\mathrm{cost}(\tau^*) = \mathrm{opt}$. *Moreover, for every $t \in K$, $\mathbf{T}^*$ contains exactly one node in $\mathcal{K}_t$.*

From GSTTD to Decomposition Tree. The following lemma gives the other direction.

Lemma 3.8. *Given any feasible solution* $\mathbf{T}$ *to the GSTTD 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 group $\mathcal{K}_t$, for $t \in K$, is spanned by* $\mathbf{T}$, *then τ involves t.*

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 GSTTD instance, and k is the number of groups.

Theorem 3.9. *There is an $O(h \log k)$-approximation algorithm in running time $(shN)^{O(sh^2)}$ for the Group Steiner Tree on Trees with Dependency Constraint problem where s is the maximum number of rules associated with a given vertex.*

With this theorem at hand, we 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 GSTTD 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 GSTTD instance, we have $s = O(\log k)$. By Claim 3.2 and Lemma 3.7, there is a solution $\mathbf{T}^*$ to the GSTTD 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 an $O(\log^2 k/\log\log k)$-approximation for DST in running time $n^{O(\log^5 k)}$, finishing the proof of Theorem 1.1.

4 APPROXIMATION ALGORITHM FOR GROUP STEINER ON TREES WITH DEPENDENCY CONSTRAINT

The goal of this section is to prove Theorem 3.9, which is repeated below.

Theorem 3.9. *There is an $O(h \log k)$-approximation algorithm in running time $(shN)^{O(sh^2)}$ for the Group Steiner Tree on Trees with Dependency Constraint problem where s is the maximum number of rules associated with a given vertex.*

Since we are not dealing with the original DST problem any more, we use T^0, T (instead of $\mathbf{T}^0, \mathbf{T}$) for trees and u, v (instead of p, q) for nodes in this section: T^0 is the input tree while T is the tree we need to output. Let V^{leaf} and V^{int} respectively be the sets of leaves and internal nodes of T^0. For every node $v \in V^{\text{int}}$, let Λ_v be the set of children of v. For every $v \in V(T^0)$, let $\Lambda_v^{\text{leaf}} = V(T^0[v]) \cap V^{\text{leaf}}$ be the set of descendants of v that are leaves.

For convenience of description, we shall introduce two types of *labels*: each rule in L corresponds to a *rule label* and each terminal $t \in K$ corresponds to a *terminal label*. So, we also view L and K as the set of rule labels and terminal labels respectively. For each rule $\ell = (v, S) \in L$, we say that v is demanding the rule label ℓ, and all the vertices in S are supplying ℓ. For every terminal $t \in K$, we also say all vertices in $\mathcal{K}_t$ are supplying the terminal label t. For every node $v \in V(T^0)$, let $\text{dem}(v) \subseteq L$ be the set of labels that v is demanding. Thus, we have $s = \max_{v \in V(T^0)} |\text{dem}(v)|$.

We can assume that the demand labels are only at the internal nodes: a rule associated with a leaf is either useless or can never be satisfied if the leaf is contained in T. We can assume only leaves can supply labels and each leaf is supplying exactly one label. A leaf that does not supply a label can be removed. For a non-leaf v supplying some labels, we can attach many leaves of cost 0 to v and let each leaf supply one label. Similarly, if a leaf v supplies more than one labels, we can attach many new leaves to v and let each of them supply one label. With this property, we can define $a_v \in L$ be the unique label that v supplies, for every $v \in V^{\text{leaf}}$.

Thus, a subtree T of T^0 with $\text{root}(T) = \text{root}(T^0)$ is valid if the following two conditions are satsfied.

- (dependency rules) All the demand rule labels in T are supplied by T: if for every $u \in V(T) \cap V^{\text{int}}$ and $\ell \in \text{dem}(u)$, there is a node $v \in V(T) \cap \Lambda_u^{\text{leaf}}$ with $a_v = \ell$.
- (terminal conditions) All the terminal labels are supplied by T: for every $t \in K$, there is a $v \in V(T) \cap V^{\text{leaf}}$ with $a_v = t$.

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 $\text{cost}(T)$, is defined as $\text{cost}(T) := \sum_{v \in V(T)} c_v$. Thus, the goal of GSTTD is to find the minimum cost valid subtree T of T^0.

A small note is that we changed the height and size of T^0 slightly when we attach new leaves to T^0. We consider these changes now. 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 attached leaves to the nodes in the original T^0. So, we have $h \leq h' + 1$. The number of internal nodes in the new T^0 is at most N'. If a node has many leaf children v with the same a_v, we only need to keep the one with the smallest cost. Since each u has $|\text{dem}(u)| \leq s$ 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 $s(h'+1)N' + k$. So $N \leq s(h'+1)N' + N' + k = O(sh'N')$.

Let T^* be the optimum tree for the given instance. Let $\text{opt} = \text{cost}(T^*)$ be the cost of the T^*.[2] We can assume the following:

(4.1a) For every label $\ell \in L \cup K$, at most one node in T^* supplies ℓ.

Indeed, if there are multiple such nodes v, we can keep any one without breaking the validity of the tree. Notice that for a terminal label $t \in K$, exactly one node supplies t by terminal conditions.

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 tree $\tilde{T}$ obeying the dependency rules such that, $\mathbb{E}\left[c(\tilde{T})\right] \leq$ opt, and for every $t \in K$, $\tilde{T}$ supplies t with probability at least $\frac{1}{h+1}$.*

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' obeys the dependency rules. 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

$$\Pr\left[T' \text{ supplies all labels in } K\right] \geq 1/2. \tag{2}$$

We repeatedly run the above procedure until T' supplies all labels in K and output T'. 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 to run the procedure twice.

[2] We remark that it is easy to check whether a valid solution exists or not: an $u \in V^{\text{int}}$ is useless if for some $\ell \in \text{dem}(u)$ there is no $v \in \Lambda_u^{\text{leaf}}$ 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 supplies all the terminal labels. So, we can assume that the instance has a valid solution.

Thus, we obtain an $O(h \log k)$-approximation for GSTTD. 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 [29] for GST on trees. We solve the lifted LP relaxation for the GSTTD 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 $L' \subseteq L$ of rule 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 always obeys the dependency rules. Finally, we show that each terminal label $v \in K$ appears in $\tilde{T}$ with large probability, using a technique that is very similar to that of [29]. [3]

4.1 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 $\mathbb{D} = V(T^0) \cup (V(T^0) \times (L \cup K))$. 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 $(u, \ell) \in V(T^0) \times (L \cup K)$ happens if and only if $u \in V(T^*)$ and $T^*[u]$ supplies ℓ (i.e, $\Lambda_u^{\text{leaf}} \cap V(T^*)$ contains a v with label $a_v = \ell$; such a node is unique if it exists by Property (4.1a)). 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:

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

$$x_{(u,\ell)} \le x_u, \qquad \forall u \in V(T^0), \ell \in L \cup K \tag{4}$$

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

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

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

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

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

(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^* obey the dependency rules, and (6) holds trivially. (7) follows from Property (4.1a). (8) holds trivially and (9) follows from the terminal conditions.

[3]Notice that in previous work, we either apply the basic LP relaxation for the quasi-polynomial size instances on trees obtained by expanding the original instance, or apply LP/SDP hierarchy on the original instance on general graphs [13, 29]. Our algorithm does both: it applies Sherali-Adams hierarchy on the expanded instance.

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 $\ell' \in L \cup K$, 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. □

4.2 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 $\sum_{v \in V(T^0)} c_v x_v^* \le \text{opt}$ 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 start with $\tilde{V} = \emptyset$ and then call solve(root(T^0), dem(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: subtree $\tilde{T}$ of T^0 obeying dependency rules

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

Algorithm 5 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 $(v, \ell)$
8: **for** every $v \in \Lambda_u$, with probability $x_v$ **do**
9:     solve($v, S_v \cup \text{dem}(v)$, $x$ conditioned on event $v$)
```

In the recursive algorithm 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 $\tilde{T}$ obeys the dependency rules and has small expected cost; moreover, every label $t \in K$ 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 $\sum\limits_{v\in\Lambda_u} x_{v,\ell} = 1$, 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 obeys the dependency rules.*

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 $|\mathrm{dem}(v)| \leq s$.

Inside each recursion of solve, the number of nested conditioning operations is $|L'|+1 \leq s(\mathrm{depth}(u)+1) \leq s(h+2)$. 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 $\mathrm{solve}(u,\cdot,\cdot)$. 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 the 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 $\mathrm{solve}(u',\cdot,\cdot)$ for some ancestor u' of u, or in $\mathrm{solve}(u,\cdot,\cdot)$ 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*

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

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

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

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 solve$(u,\cdot,\cdot)$ does not depend on the random numbers generated before x^{old}, and thus does not depend on $\mathcal{E}$. Therefore, $\mathbb{E}\left[x_e^{\text{new}}\middle|x^{\text{old}},L',\mathcal{E}\right]=x_e^{\text{old}}$. Deconditioning over L' and the components inside x^{old} other than x_e^{old} gives

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

□

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

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

Proof. Again we can condition on the event that the recursion solve$(u,\cdot,\cdot)$ exists. Consider the iteration of Loop 8 for v in 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 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. $\mathbb{E}[\text{cost}(\tilde{T})]=\sum_{v\in V(T^0)}\Pr[v\in\tilde{V}]\cdot c_v=\sum_{v\in V(T^0)}x_v^*c_v\le$ opt. □

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

Let $m_t=\left|\{v\in\tilde{V}\cap V^{\text{leaf}}:a_v=t\}\right|$ be the number of nodes in $\tilde{V}\cap V^{\text{leaf}}$ with label t. Our goal is to prove that $m_t\ge1$ with high probability. The proof is almost the same as the counterpart in [29]; we include it here for completeness.

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

Proof. By Corollary 4.10, we have

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

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 $a_w=t$, we have $\mathbb{E}[m_t|w\in\tilde{V}]\le h+1$.*

Proof. Assume w is at depth h' in the tree T^0. We partition the set $\{w'\in V^{\text{leaf}}\backslash\{w\}:a_{w'}=t\}$ 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 root(T^0) has depth 0). Notice that $w'\ne w$ and thus the LCA has depth between 0 and $h'-1$. We show that for every $i=0,1,\cdots,h'-1$,

$$\mathbb{E}\left[|U_i\cap\tilde{V}|\middle|w\in\tilde{V}\right]\le1.\tag{10}$$

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

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 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 $x^{(u_1,0)}$, $x^{(u_1,1)},\cdots,x^{(u_1,sh)},x^{(u_2,0)},x^{(u_2,1)}\cdots,x^{(u_{m-1},sh)},x^{(u_t,0)}$ is a martingale.

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

$$\mathbb{E}\left[|U_i\cap\tilde{V}|\middle|\{S_v\}_{v\in\Lambda_u},x^{(u,sh)},w\in\tilde{V}\right]=\sum_{w'\in U_i}x_{w'}^{(u,sh)}$$
$$=\sum_{w'\in U_i}x_{(w',t)}^{(u,sh)}\le x_{(u,t)}^{(u,sh)}\le1.$$

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

Lemma 4.14. *For every $t\in K$, we have $E[m_t|m_t\ge1]\le h+1$.*

Proof. In the following, w and w' in summations are over all nodes in V^{leaf} with label t.

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

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

Corollary 4.15. $\Pr[m_t \geq 1] \geq \frac{1}{h+1}$ *for every* $t \in K$.

Proof. Notice that $1 = \mathbb{E}[m_t] = \mathbb{E}[m_t|m_t \geq 1] \cdot \Pr[m_t \geq 1]$. 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 [29] 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 the 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][4]. 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.

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

ACKNOWLEDGMENTS

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. We would like to thank Pasin Manurangsi for pointing out the inapproximability results of Dense CSPs and Densest k-Subgraphs.

F. Grandoni is partially supported by the SNSF Grants 200021_159697/1 and 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. 1996. 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*. 184–193. https://doi.org/10.1109/SFCS.1996.548477

[2] MohammadHossein Bateni, Moses Charikar, and Venkatesan Guruswami. 2009. 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*. 543–552. https://doi.org/10.1145/1536414.1536488

[3] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. 2013. Steiner Tree Approximation via Iterative Randomized Rounding. *J. ACM* 60, 1 (2013), 6:1–6:33. https://doi.org/10.1145/2432622.2432628

[4] Parinya Chalermsook, Fabrizio Grandoni, and Bundit Laekhanukit. 2015. On Survivable Set Connectivity. In *SODA*. 25–36.

[5] Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. 1999. Approximation Algorithms for Directed Steiner Problems. *J. Algorithms* 33, 1 (1999), 73–91. https://doi.org/10.1006/jagm.1999.1042

[6] Chandra Chekuri, Guy Even, and Guy Kortsarz. 2006. A greedy approximation algorithm for the group Steiner problem. *Discrete Applied Mathematics* 154, 1 (2006), 15–34. https://doi.org/10.1016/j.dam.2005.07.010

[7] Chandra Chekuri and Martin Pál. 2005. 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*. 245–253. https://doi.org/10.1109/SFCS.2005.9

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

[9] Eden Chlamtac. 2007. 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*. 691–701. https://doi.org/10.1109/FOCS.2007.13

[10] Marek Cygan, Fabrizio Grandoni, and Monaldo Mastrolilli. 2013. 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*. 342–351. https://doi.org/10.1137/1.9781611973105.25

[11] Alina Ene, Deeparnab Chakrabarty, Ravishankar Krishnaswamy, and Debmalya Panigrahi. 2015. Online Buy-at-Bulk Network Design. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, Venkatesan Guruswami (Ed.). IEEE Computer Society, 545–562. https://doi.org/10.1109/FOCS.2015.40

[12] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. 2004. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.* 69, 3 (2004), 485–497. https://doi.org/10.1016/j.jcss.2004.04.011

[13] Zachary Friggstad, Jochen Könemann, Young Kun-Ko, Anand Louis, Mohammad Shadravan, and Madhur Tulsiani. 2014. 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*. 285–296. https://doi.org/10.1007/978-3-319-07557-0_24

[14] Naveen Garg, Goran Konjevod, and R. Ravi. 2000. A Polylogarithmic Approximation Algorithm for the Group Steiner Tree Problem. *J. Algorithms* 37, 1 (2000), 66–84. https://doi.org/10.1006/jagm.2000.1096

[15] Shashwat Garg, Janardhan Kulkarni, and Shi Li. [n. d.]. 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*.

[16] Fabrizio Grandoni and Bundit Laekhanukit. 2017. Surviving in directed graphs: a quasi-polynomial-time polylogarithmic approximation for two-connected directed Steiner tree, See [20], 420–428. https://doi.org/10.1145/3055399.3055445

[17] Anupam Gupta, Ravishankar Krishnaswamy, and R. Ravi. 2010. 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*. 1521–1538. https://doi.org/10.1137/1.9781611973075.124

[18] Mohammad Taghi Hajiaghayi, Rohit Khandekar, and Guy Kortsarz. [n. d.]. Approximating Group Steiner Tree via Configuration LP. ([n. d.]). Personal Communication.

[19] Eran Halperin and Robert Krauthgamer. 2003. Polylogarithmic inapproximability. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, Lawrence L. Larmore and Michel X. Goemans (Eds.). ACM, 585–594. https://doi.org/10.1145/780542.780628

[20] Hamed Hatami, Pierre McKenzie, and Valerie King (Eds.). 2017. *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*. ACM. https://doi.org/10.1145/3055399

[21] Christopher S. Helvig, Gabriel Robins, and Alexander Zelikovsky. 2001. An improved approximation scheme for the Group Steiner Problem. *Networks* 37, 1 (2001), 8–20. https://doi.org/10.1002/1097-0037(200101)37:1<8::AID-NET2>3.0.CO;2-R

[22] Rohit Khandekar, Guy Kortsarz, and Zeev Nutov. 2012. Approximating fault-tolerant group-Steiner problems. *Theorerical Computer Science* 416 (2012), 55–64.

[23] Bundit Laekhanukit. 2014. Parameters of Two-Prover-One-Round Game and The Hardness of Connectivity Problems. In *SODA*. 1626–1643.

[24] Bundit Laekhanukit. 2016. Approximating Directed Steiner Problems via Tree Embedding. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy (LIPIcs)*, Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi (Eds.), Vol. 55. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 74:1–74:13. https://doi.org/10.4230/LIPIcs.ICALP.2016.74

[25] Elaine Levey and Thomas Rothvoss. 2016. 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*. 168–177. https://doi.org/10.1145/2897518.2897532

[26] Pasin Manurangsi. 2017. Almost-polynomial ratio ETH-hardness of approximating densest k-subgraph, See [20], 954–961. https://doi.org/10.1145/3055399.3055412

[27] Pasin Manurangsi and Prasad Raghavendra. 2017. A Birthday Repetition Theorem and Complexity of Approximating Dense CSPs. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland (LIPIcs)*, Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl (Eds.), Vol. 80. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 78:1–78:15. https://doi.org/10.4230/LIPIcs.ICALP.2017.78

[28] Gabriel Robins and Alexander Zelikovsky. 2005. Tighter Bounds for Graph Steiner Tree Approximation. *SIAM J. Discrete Math.* 19, 1 (2005), 122–134. https://doi.org/10.1137/S0895480101393155

[29] Thomas Rothvoß. 2011. Directed Steiner Tree and the Lasserre Hierarchy. *CoRR* abs/1111.5473 (2011). http://arxiv.org/abs/1111.5473

[30] Alexander Zelikovsky. 1993. An 11/6-Approximation Algorithm for the Network Steiner Problem. *Algorithmica* 9, 5 (1993), 463–470. https://doi.org/10.1007/BF01187035

[31] Alexander Zelikovsky. 1997. A Series of Approximation Algorithms for the Acyclic Directed Steiner Tree Problem. *Algorithmica* 18, 1 (1997), 99–110. https://doi.org/10.1007/BF02523690

[32] Leonid Zosin and Samir Khuller. 2002. On directed Steiner trees. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA.*, David Eppstein (Ed.). ACM/SIAM, 59–63. http://dl.acm.org/citation.cfm?id=545381.545388