On Approximating Degree-Bounded Network Design Problems

Xiangyu Guo
Dept. of Comp. Sci. and Eng.
University at Buffalo, USA
xiangyug@buffalo.edu

Guy Kortsarz*
Dept. of Comp. Sci.
Rutgers University Camden, USA
guyk@camden.rutgers.edu

Bundit Laekhanukit
ITCS,
SUFE, China
bundit@sufe.edu.cn

Shi Li
Dept. of Comp. Sci. and Eng.
University at Buffalo, USA
shil@buffalo.edu

Daniel Vaz
Operations Research Group,
TU Munich, Germany
daniel.vaz@tum.de

Jiayi Xian
Dept. of Comp. Sci. and Eng.
University at Buffalo, USA
jxian@buffalo.edu

Abstract

Directed Steiner Tree (DST) is a central problem in combinatorial optimization and theoretical computer science: Given a directed graph $G = (V, E)$ with edge costs $c \in \mathbb{R}^E_{\geq 0}$, a root $r \in V$ and k terminals $K \subseteq V$, we need to output the minimum-cost arborescence in G that contains an $r \to t$ path for every $t \in K$. Recently, Grandoni, Laekhanukit and Li, and independently Ghuge and Nagarajan, gave quasi-polynomial time $O(\log^2 k / \log\log k)$-approximation algorithms for the problem, which are tight under popular complexity assumptions.

In this paper, we consider the more general Degree-Bounded Directed Steiner Tree (DB-DST) problem, where we are additionally given a degree bound d_v on each vertex $v \in V$, and we require that every vertex v in the output tree has at most d_v children. We give a quasi-polynomial time $(O(\log n \log k), O(\log^2 n))$-bicriteria approximation: The algorithm produces a solution with cost at most $O(\log n \log k)$ times the cost of the optimum solution that violates the degree constraints by at most a factor of $O(\log^2 n)$. This is the first non-trivial result for the problem.

While our cost-guarantee is nearly optimal, the degree violation factor of $O(\log^2 n)$ is an $O(\log n)$-factor away from the approximation lower bound of $\Omega(\log n)$ from the set-cover hardness. The hardness result holds even on the special case of the *Degree-Bounded Group Steiner Tree* problem on trees (DB-GST-T). With the hope of closing the gap, we study the question of whether the degree violation factor can be made tight for this special case. We answer the question in the affirmative by giving an $(O(\log n \log k), O(\log n))$-bicriteria approximation algorithm for DB-GST-T.

*The corresponding author.

1 Introduction

Network design is a central problem in combinatorial optimization and computer science. To capture more practical situations, the more general model of network design with *degree-constraints* was suggested in the early 90's [21, 8] and has attracted researchers in both theory and practice for decades. One of the most famous examples is the *Degree-Bounded Minimum Spanning Tree* (DB-MST) problem, which models the problem of designing a multi-casting network in which each node only has enough power to broadcast to a bounded number of its neighbors. This problem has been studied in a sequence of works (see, e.g.,[15, 17, 11, 23]), leading to the breakthrough result of Goemans [11] followed by the work of Singh and Lau [23], which settled down the problem by giving an algorithm that outputs a solution with optimum cost, while violating the degree bound by an additive factor of +1 [23]. Since the works on DB-MST, many works have been dedicated to studying generalizations of the problem: the *Degree-Bounded Steiner Tree* problem, in which the goal is to find a minimum-cost subgraph that connects all the terminals, while meeting the given degree bounds, was studied in [16, 20]. The *Degree-Bounded Survivable Network Design* problem, where each pair of nodes v, w are required to have at least λ_{vw} edge-disjoint v-w paths, has also been studied in literature; see, e.g., [19, 20]. Recently, degree-bounded network design problems have also been studied in the online setting [4, 3, 5].

Besides the standard (also called point-to-point) network design problems, a degree-bounded version of the *Group Steiner Tree* problem has also been considered in the literature [4]. In this problem, we are given a graph $G = (V, E)$ with edge costs, a root vertex r, as well as a collection of groups $O_t \subseteq V$, $t \in [k]$, and the goal is to find a minimum-cost tree that connects the root r to at least one vertex of each group O_t, $t \in [k]$. The *Degree-Bounded Group Steiner Tree* problem (DB-GST) is the variant of group Steiner tree in which degree bounds are given for each vertex, and these degree bounds must be obeyed by any feasible solution. Since the group Steiner tree problem generalizes Steiner tree, its degree-bounded version generalizes the degree-bounded Steiner tree problem. Dehghani et al. [4] studied the DB-GST problem in the online setting, and gave a negative result for the problem: their result shows that it is not possible to approximate both cost and weight of the Online DB-GST problem simultaneously, even when the input graph is a star. More specifically, there exists an input demand sequence that forces any algorithm to pay a factor of $\Omega(n)$ either in the cost or in the degree violation. To date there was no non-trivial approximation algorithm for DB-GST, either in the online or offline setting, and even when all the edges have zero-cost. This was listed as an open problem by Hajiaghayi [13] at the 8th Flexible Network Design Workshop (FND 2016).

In this paper, we study a degree-bounded variant of the classic network design problem, the *Degree-Bounded Directed Steiner Tree* problem (DB-DST). Formally, in DB-DST, we are given an n-vertex directed graph $G = (V, E)$ with costs on edges, a root vertex r, a set of k terminals K, and degree bounds d_v for each vertex v. The goal is to find a minimum-cost rooted tree $T \subseteq G$ that contains a path from the root r to every terminal $t \in K$, while respecting the degree bound, i.e., the out-degree of each vertex v in T is at most d_v. Despite being a classic problem, there was no previous positive result on DB-DST as it is a generalization of DB-GST.

The barriers in obtaining any non-trivial approximation algorithm for DB-GST and DB-DST are similar. Most of the previous algorithms to these two problems either run on the metric closure of the input graph [9, 7, 22], require metric-tree embedding [9, 1, 6] or use height-reduction techniques [24, 2, 12, 10], all of which lose track of the degree of the solution subgraph.

We solve the open problem of Hajiaghayi [13], by presenting an algorithm that is a bicriteria

$(O(\log k \log n), O(\log^2 n))$-approximation algorithm for DB-DST running in quasi-polynomial-time. We say that a randomized algorithm is an (α, β)-bicriteria-approximation algorithm for DB-DST if it outputs a tree T containing an $r \to t$ path for every terminal $t \in K$ such that the number of children of every vertex v in T is at most $\beta \cdot d_v$, and the expected cost of the tree is at most α times the cost of the optimum tree that does not violate the degree constraints.

Theorem 1.1. *There is a randomized $(O(\log n \log k), O(\log^2 n))$-bicriteria approximation algorithm for the degree-bounded directed Steiner tree problem in $n^{O(\log n)}$-time.*

To the best of our knowledge, our result for DB-DST is the first non-trivial bicriteria approximation for the problem. Our technique expands upon the recent result of Grandoni, Laekhanukit and Li [12] for the Directed Steiner Tree problem. We observe that their algorithm can be easily extended to the problem with degree bounds. Nevertheless, to amend the degree-constrained problem into their framework, we are required to prove a concentration bound for the degrees, which is rather non-trivial. Notice that the $O(\log n \log k)$-approximation factor on the cost of the tree is almost tight due to the hardness of $\Omega(\log^{2-\epsilon} n)$ in [14] for Directed Steiner Tree and the slightly improved hardness of $\Omega(\log^2 n / \log\log n)$ in [12]. There is a hardness of $\Omega(\log n)$ for the degree-violation factor from the set-cover problem, even if the cost of the output tree is ignored.

While our result for DB-DST is (almost) tight on the cost guarantee, the degree violation factor $O(\log^2 n)$ is an $O(\log n)$ factor away from the approximation lower bound of $\Omega(\log n)$ from the set-cover hardness. To understand if the gap can be reduced, we study the special case of DB-DST obtained from the hardness construction in [14], namely the *Degree-Bounded Group Steiner Tree problem on trees* (DB-GST-T). In this problem, we are given an (undirected) tree $T^\circ = (V^\circ, E^\circ)$ with edge-costs, a root r, k subsets of vertices (called groups) $O_1, \ldots, O_k \subseteq V$ and a degree bound d_v for each vertex $v \in V^\circ$. The goal is to find a minimum-cost subtree $T \subseteq T^\circ$ that joins r to at least one vertex from each group O_t, for every $t \in [k]$, while respecting the degree bound, i.e., the number of children of each vertex v in T is at most d_v. We present an $(O(\log k \log n), O(\log n))$-bicriteria approximation algorithm for DB-GST-T that runs in polynomial time. So, the degree violation of our algorithm is tight and the cost-guarantee is almost tight. This improves upon the $O(\log n \log k, \log n \log k)$-bicriteria approximation algorithm due to Kortsarz and Nutov [18] who observe that the randomized rounding algorithm in [9] also gives a guarantee on degree-violation.

Theorem 1.2. *There is a randomized $\big(O(\log n \log k), O(\log n)\big)$-bicriteria approximation algorithm for the degree-bounded group Steiner tree problem on trees, running in polynomial time.*

Remark: As in [12, 10], we could save a factor of $\log\log n$ in the approximation factor for DB-DST, with a slight increase in the running time. However, this complicates the algorithmic framework. To deliver the algorithmic idea in a cleaner way, we choose to present the results with $O(\log n \log k)$ approximation ratios.

1.1 Our Techniques

Our algorithm for degree-bounded directed Steiner tree takes ingredients from both [12] and [10]. As in these papers, we consider an optimum solution, and recursively partition it into balanced sub-trees; we then assign a "state" to each of these sub-trees. The tree structure of this recursive partition, as well as all of the states, form what we call a *state tree*. We solve the problem indirectly, by finding a good state tree, which we can transform back into a corresponding good solution. The

state of a sub-tree contains a set of special vertices in the sub-tree that we call *portals*; these were used in [10] to obtain their improved approximation algorithm for DST. We construct a super-tree $\mathbf{T}^\circ$ that contains all possible state trees as sub-trees and reduce the problem considered into that of finding a good sub-tree of small cost in $\mathbf{T}^\circ$. This can be done by formulating a linear program (LP) relaxation and rounding the LP solution using a recursive procedure. The construction of the super-tree and the LP rounding techniques are similar to those in [12]. To extend the algorithm to DB-DST, we need to store the degrees of all of the portals in the state.

This algorithmic framework outputs a so-called "multi-tree": This is a tree where a vertex or an edge can appear multiple times. Repeating the procedure for $Q = O(\log n \log k)$ times, we obtain a set of Q multi-trees. This process violates the degree requirements and thus we obtain bicriteria approximation results. The analysis of this process is non-trivial as we need to introduce several techniques and prove a concentration bound on the number of times a vertex appears in a multi-tree. We summarize the techniques used to analyze this process below.

To bound the factor by which degree requirements are violated, we introduce several techniques. First, we transform the graph so that the out-degree of each vertex is at most 2. Since this clearly changes the degree bounds, we introduce a recursive process that, given a solution in the transformed graph, computes the *original degree* of each vertex, that is, the degree of the vertex in the corresponding solution in the original graph (see Section 2.1 for details). By imposing the degree bounds on these original degrees, we obtain an equivalent problem on a graph with maximum out-degree of 2.

Next, we must ensure that the bounds on the original degrees are satisfied in the state tree: to do so, the original degrees of portals are stored in the state; by construction of the state tree, edges are only added to the solution at leaves of the state tree, and all outgoing edges for a (copy of a) single vertex are chosen by the state of a single leaf. Since the original degree of a vertex can be recursively computed from the original degrees of its children, we check for bounds on original degrees on the leaves of the state tree, by only considering valid those that lead to satisfied degree bounds (see Section 3.2 for details).

Finally, we must consider the existence of multiple copies of a vertex in a multi-tree: while the step above bounds the degree of a single copy of a vertex, we must now bound the number of copies of a vertex in a multi-tree. We use the expectation of exponential random variables to bound the total number of copies of a vertex, over all Q runs, to be at most $O(\log^2 n)$ with large constant probability (see Sections 2.3 and 5.4). Combining these three steps with the techniques for DST, we obtained the claimed bicriteria approximation.

For the DB-GST-T problem, our technique comes from observing that the rounding algorithm for GST-T (no degree bounds) in [9] is indeed a generalization of random walk. As we slightly boost the branching probability by a constant factor, this (almost) does not affect the degree bound, but the probability of connecting the root vertex to each group is amplified dramatically. A drawback is that it also incurs a huge blow-up in the cost. To handle the blow-up, we stop amplifying the branching probability when the connecting probability is sufficiently large. The best (but inaccurate) way to illustrate our algorithm is by considering a random walk from the root vertex to a group O_t. We change the random process by branching into two directions simultaneously in each step, and then stop the extra branching when it generates $\Theta(\log n)$ simultaneous random walks. Since we have $O(\log n)$ simultaneous random walks, the cost incurred by the process is blown-up by a factor $O(\log n)$, but the degree-violation is blown-up by only a factor 2. At the same time, the probability of reaching the group O_t goes up by a factor $\Omega(\log n)$. Thus, if we need $O(\log k \log n)$

rounds to reach every group, then we now need only $O(\log k)$ rounds. There is no difference in the cost for running the algorithm for $O(\log k \log n)$ rounds or $O(\log k)$ rounds (with an extra $O(\log n)$ factor in the cost), but it saves a factor in the degree-violation of $O(\log n)$.

2 Preliminaries for Degree-Bounded Directed Steiner Tree

2.1 Notations and Assumptions

In our algorithm and analysis for the DB-DST problem, a tree is always an out-arborescence. Given a tree T, we use $\text{root}(T)$ to denote its root. Given T and a vertex v in T, we use $\Lambda_T(v)$ to denote the set of children of v, and $\Lambda^*_T(v)$ to denote the set of descendants of v (including v itself) in the tree T. A sub-tree T' of T is a weakly-connected sub-graph of T; such a T' must be an out-arborescence. Sometimes, we shall use left and right children to refer to the two children of a vertex in a tree; in this case, the order of the two children is important and will be clearly specified. For an edge $e = (u, v)$, we use $\text{tail}(e) = v$ to denote its tail. For a triple $\xi = (u, v, v')$ of three vertices, we use $\text{second}(\xi) = v$ and $\text{third}(\xi) = v'$ to denote the second and third parameter of ξ.

Our input digraph is G. Let $d_{\max} = \max_{v \in V} d_v$. We shall assume each terminal $t \in K$ has only one incoming edge and no outgoing edges in G. This can be assumed w.l.o.g using the following simple operation: For every terminal $t \in K$ that does not satisfy the condition, we add a new vertex t', an edge (t, t') and replace t with t' in K. We increase d_t by 1 and set $d_{t'} = 0$.

We will assume that each non-terminal $u \in V \setminus K$ has at most 2 outgoing edges in G, though this requires us to extend the problem definition slightly. We will show how to transform G into a new graph $G' = (V', E')$ where the condition above holds, and the problem is equivalent, with using a more general definition of degree constraints. We describe the transformation by focusing on some non-terminal u with $b \geq 3$ outgoing edges. We replace the star centered at u with its b outgoing edges by a gadget which is a full binary-tree rooted at u with b leaves being the out-neighbors of u. For every newly added vertex u, we set $d_u = d_{\max}$. This way every vertex in G' will have at most 2 outgoing edges. The cost of the edges in the gadget can be naturally defined.

Unfortunately, this operation changes the degree of vertices. To address this issue, we define a simple transformation function $\phi_v : \mathbb{Z} \to \mathbb{Z}$ for every $v \in V'$ as follows: If v is a vertex in the original graph ($v \in V$), then ϕ_v is identically 1 ($\phi(x) = 1$, $x \in \mathbb{Z}$). Otherwise, v is a non-root internal vertex of some gadget and we define ϕ_v to be the identity function ($\phi(x) = x$, $x \in \mathbb{Z}$). Given a tree T' of G', we can compute the *original degree* ρ_u of a vertex u in the corresponding tree T in G recursively as follows: $\rho_u = 0$ if u is a leaf, and $\rho_u = \sum_{v \in \Lambda_T(u)} \phi_v(\rho_v)$ otherwise. We require that for every v in the output tree T, the original degree ρ_v of v is at most d_v, thus ensuring that the original tree in G will satisfy the degree bounds. For simplicity of notation, we refer to this modified graph as $G = (V, E)$ instead of $G' = (V', E')$ in the remainder of the text.

2.2 Balanced Tree Partition

We shall use the following basic tool as the starting point of our algorithm design. Its proof is elementary and deferred to Appendix A.

Lemma 2.1. *Let $T = (V_T, E_T)$ be an n-vertex binary tree. Then there exists a vertex $v \in V_T$ with $n/3 < |\Lambda^*_T(v)| \leq 2n/3 + 1$.*

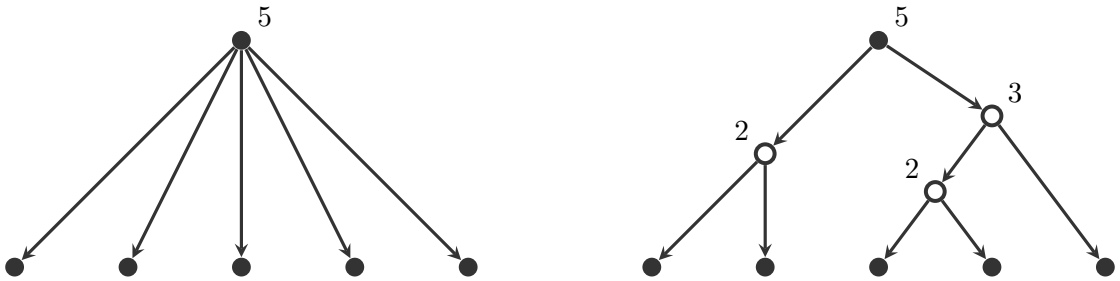

Figure 1: Example of the transformation to a graph with out-degree 2. On the left, a star consisting of the outgoing edges of a vertex is presented; on the right, the transformed subgraph. Vertices that are added by the transformation are represented as hollow circles. Each non-leaf vertex is labeled by its computed original degree.

Given a tree $T = (V_T, E_T)$ as in the lemma, we can partition it into two trees $T_1 = (V_{T_1}, E_{T_1})$ and $T_2 = (V_{T_2}, E_{T_2})$, where T_2 contains vertices in $\Lambda^*_T(v)$ and T_1 contains vertices in $V_T \setminus (\Lambda^*_T(v) \setminus \{v\})$. First assume $n \geq 4$. Since $2n/3 + 1 < n$, we know that $v \neq \text{root}(T)$, thus implying $\text{root}(T_1) = \text{root}(T) \neq \text{root}(T_2) = v$, which is a leaf in T_1. Consequently, we have $E_{T_1} \cup E_{T_2} = E_T$, $E_{T_1} \cap E_{T_2} = \emptyset$, and $V_{T_1} \cup V_{T_2} = V_T, V_{T_1} \cap V_{T_2} = \{\text{root}(T_2)\}$. Moreover, $|V_{T_1}|, |V_{T_2}| \leq 2n/3 + 1$, which is strictly less than n. Thus, T_1 and T_2 are sub-trees that form a balanced partition of (the edges of) T. We call this procedure the *balanced tree partitioning* on T.

When $n = 3$, there are 2 types of trees. If the root has two children, then we could not make both $|V_{T_1}|$ and $|V_{T_2}|$ to be smaller than 3. If the tree is a path of 2 edges, then we can choose v to be the middle vertex and the procedure partitions the tree into two edges. Later, we shall apply the balanced tree partitioning procedure recursively. We stop the recursion when the tree is either an edge, or only contains the root and its 2 children. In other words, we stop when the tree has only 1 level of edges.

2.3 Multi-Tree

We define a *multi-tree* in G as an intermediate structure. It is simply a tree over *multi-sets* of vertices and edges in G:

Definition 2.2 (Multi-Tree)**.** *Given the input digraph $G = (V, E)$, a* multi-tree *in G is a tree $T = (V_T, E_T)$ where every vertex $a \in V_T$ is associated with a label* $\text{label}(a) \in V$ *such that for every $(a, b) \in E_T$, we have* $(\text{label}(a), \text{label}(b)) \in E$.

We say that each vertex $a \in V_T$ is a copy of the vertex $\text{label}(a) \in V$ and each edge $(a, b) \in E_T$ is a copy of the edge $(\text{label}(a), \text{label}(b)) \in E$. So, we say that T is rooted at a copy of $v \in V$, if $\text{label}(\text{root}(T)) = v$, and T contains a copy of some $v \in V$ if there exists some $a \in V_T$ with $\text{label}(a) = v$. We extend the costs c_e, the functions ϕ_v and the degree bounds d_v automatically to their copies in a multi-tree. That means, for a vertex a and an edge (a, b) in a multi-tree, $d_a = d_{\text{label}(a)}, \phi_a \equiv \phi_{\text{label}(a)}$ and $c_{(a,b)} = c_{(\text{label}(a),\text{label}(b))}$. The cost of a multi-tree $T = (V_T, E_T)$ is naturally defined as $\text{cost}(T) = \sum_{e \in E_T} c_e$. Given a multi-tree T, the "original degree" ρ_a of a vertex a can be computed in the same way as before.

Definition 2.3 (Good Multi-Trees)**.** *Let $T = (V_T, E_T)$ be a multi-tree in G. We say that T is good if it is rooted at a copy of r, has leaves being copies of terminals, and the original degree of any vertex a in T is at most d_a.*

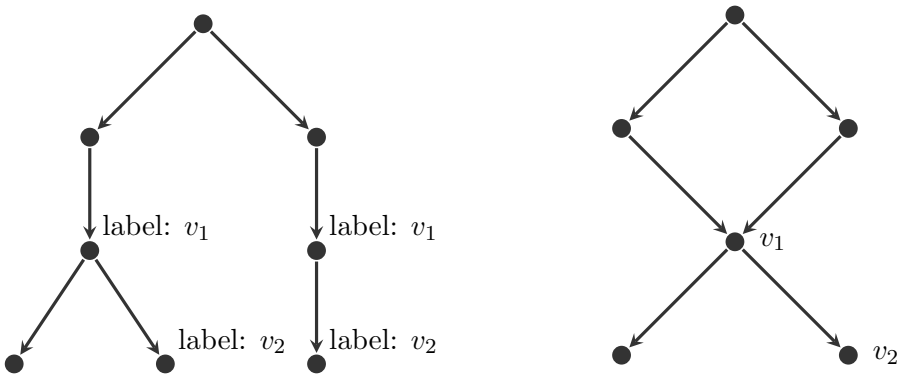

Figure 2: Example of a multi-tree and corresponding subgraph of G.

We can then state the main theorem for DB-DST, which we prove in Sections 3 to 5.

Theorem 2.4 (Main Theorem for DB-DST)**.** *There is an $n^{O(\log n)}$-time randomized algorithm that outputs a good multi-tree $T = (V_T, E_T)$ such that*

(2.4a) $\mathbb{E}_T[\mathrm{cost}(T)] \le \mathrm{opt}$*, where* opt *is the cost of the optimum solution for the instance.*

(2.4b) For every $t \in K$, we have $\Pr_T[V_T$ contains a copy of $t] \ge \Omega(1/\log n)$.

(2.4c) For some $s = \Omega\left(\frac{1}{\log n}\right)$, it holds, for every $v \in V$, that

$$\mathbb{E}\left[\exp\left(s \cdot (\textit{number of copies of } v \textit{ in } T)\right)\right] \le 1 + O\left(\tfrac{1}{\log n}\right).$$

We show that this implies Theorem 1.1.

Proof of Theorem 1.1. We run the algorithm in Theorem 2.4 Q times to obtain Q good multi-trees $T_1, T_2, \cdots, T_Q$, for some large enough $Q = O(\log n \log k)$. Our output will contain all edges that appear in the Q multi-trees. Notice that the output may not be a tree, but we can remove edges so that it becomes a tree. Applying union bound, all terminals appear in the union of the Q trees with probability at least 0.9, when Q is big enough. By Property (2.4c) in the theorem statement, we have for every v,

$$\mathbb{E}\left[\exp\left(s \cdot (\#\text{ copies of } v \text{ in } T_1, \cdots, T_Q)\right)\right] \le \left(1 + O\left(\frac{1}{\log n}\right)\right)^Q = \exp(O(\log k)).$$

The above inequality holds since the Q trees are produced independently.

Thus, if $M = O(\log n)$ is big enough, by Markov's inequality we have

$$\Pr\left[\exp\left(s \cdot (\#\text{ copies of } v \text{ in } T_1, \cdots, T_Q)\right) \ge \exp(M)\right] \le \frac{1}{10n}.$$

The event on the left side is exactly that the number of copies of v in $T_1, \cdots, T_Q$ is at least M/s.

Thus, with probability at least 0.8, every terminal t appears in one of the Q trees and every vertex v appears at most $M/s = O(\log^2 n)$ times in $T_1, T_2, \cdots, T_Q$. Taking the union of all trees and reflecting the edges in original graph G, we have a sub-graph G' of G that contains a path from r to every terminal $t \in K$. The total cost of edges in G' is at most $O(\log n \log k) \cdot \text{opt}$. For every vertex v, the out-degree of v in G' will be at most $(M/s)d_v = O(\log^2 n)d_v$. We can take an arbitrary Steiner tree T in G' as the output of the algorithm. This gives us an $(O(\log n \log k), O(\log^2 n))$-bicriteria approximation algorithm for the degree-bounded directed Steiner tree problem. The running time of the algorithm is $n^{O(\log n)}$. □

Organization The remaining part of the paper is organized as follows. In Section 3, we define states and good state trees. In Section 4, we argue that the problem of finding a small cost valid tree can be reduced to that of finding a small cost state-tree. In Section 5, we present our linear programming rounding algorithm that finishes the proof of Theorem 2.4. Section 6 is dedicated to the proof of Theorem 1.2 for the degree-bounded group Steiner tree problem on trees (DB-GST-T).

3 States and State-Trees

Given the optimum tree T^* (which is binary by our assumptions) for the DB-DST problem, we can apply the balanced tree partitioning recursively to obtain a *decomposition tree*: We start from T^* and partition it into two trees T_1 and T_2 using the balanced-tree-partitioning procedure, and then recursively partition T_1 and T_2 until we obtain sub-trees with 1 level of edges: Such a tree contains either a single edge, or two edges from the root. Then the decomposition tree is a full binary tree where each node corresponds to a sub-tree of T^*. Due to the balance condition, the height of the tree will be $O(\log n)$. Throughout the paper, we shall use $h = \Theta(\log n)$ to denote an upper bound on the height of this decomposition tree.

Thanks to its small depth, the decomposition tree becomes the object of interest. However, as each node in the tree corresponds to a sub-tree of the optimum solution T^*, it contains too much information for the algorithm to handle. Instead, we shall only extract a small piece of information from each node that we call the *state* of the node. On one hand, a state contains much less information than a sub-tree does, so we can afford to enumerate all possible states for a node. On the other hand, the states of nodes in the decomposition tree still contain enough information for us to check whether the correspondent multi-tree is good. We call the binary tree of states a *state tree*; we require in a *good state tree*, the states of nodes satisfy some consistency constraints. Then we can establish a two-direction connection between good multi-trees and good state trees.

Given a valid tree T in G and a sub-tree T' of T, we now start to make definitions related to the state of T' w.r.t T. It is convenient to think that T is the optimum tree T^* and T' is a sub-tree of $T = T^*$ obtained from the recursive balanced-partitioning procedure, since this is how we use the definitions. However, the definitions are w.r.t general T and T'; from now on till the end of Section 3, we fix any valid tree T and its sub-tree T'.

3.1 Portals

Other than root(T'), the state for T' w.r.t T contains the set of *portals* of T':

Definition 3.1. *A vertex v in T' is a* portal *in T', if v is* root(T') *or a non-terminal leaf of T'.*

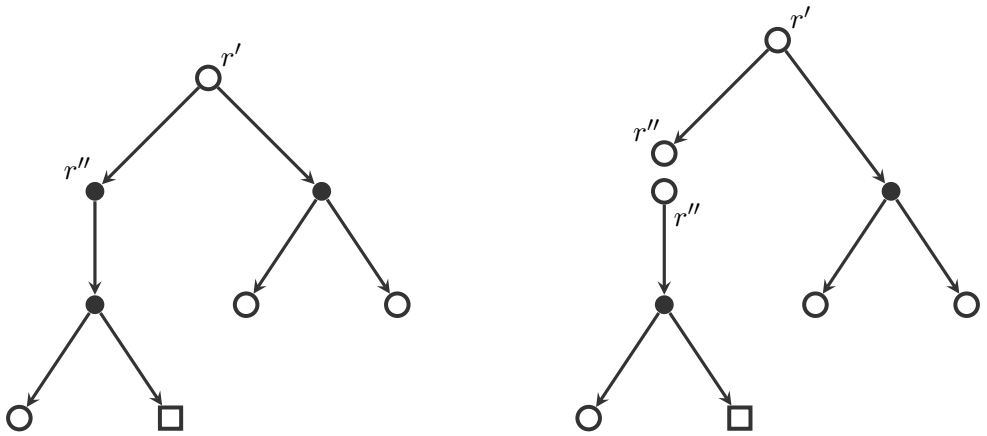

Figure 3: Example of a subtree T' and the subtrees corresponding to an allowable child-pair. Portals are drawn as hollow circles and terminals as hollow squares. The left subtree is T', the right subtrees are T'_1 (top-right) and T'_2 (bottom-right). The sets of portals S, S_1, S_2 contain the portals of T', T'_1, T'_2 respectively.

In general, the set of portals of T' can be large, but if T' is obtained from the recursive balanced-tree-partitioning procedure for T, then the number of portals can be shown to be at most $h + 1$. As we shall often use the root and set of portals together, we make the following definition:

Definition 3.2 (Root-Portals-Pair)**.** *(r', S) is called a root-portals-pair if $r' \in S \subseteq V \setminus K$.*

It is easy to see that the root-portal-pairs for an internal node of the decomposition tree and its two children satisfy some properties stated in the following definition:

Definition 3.3 (Allowable Child-Pair)**.** *Given three root-portals-pairs $(r', S), (r', S_1)$ and (r'', S_2), we say $((r', S_1), (r'', S_2))$ is an allowable child-pair of (r', S) if $r'' \notin S, S_1 \cup S_2 = S \cup \{r''\}$ and $S_1 \cap S_2 = \{r''\}$.*

The following claim motivates the definition of allowable child pairs:

Claim 3.4. *Assume $T' = (V', E')$ contains at least 2 levels of edges. Let $T'_1 = (V'_1, E'_1)$ and $T'_2 = (V'_2, E'_2)$ be the two sub-trees obtained by applying the balanced tree partitioning on T'. Let $r' = \text{root}(T') = \text{root}(T'_1)$, $r'' = \text{root}(T'_2) \neq r'$ and S, S_1, S_2 be the sets of portals in T', T'_1, T'_2 respectively. Then, $((r', S_1), (r'', S_2))$ is an allowable child-pair of (r', S).*

Proof. First, r'' is not a portal of T' since it is a non-root internal vertex in of T'. Second, it is easy to see that $S_1 = (S \cup \{r''\}) \cap V'_1$ and $S_2 = (S \cup \{r''\}) \cap V'_2$. So, $S_1 \cup S_2 = S \cup \{r''\}$ and $S_1 \cap S_2 = \{r''\}$. □

3.2 Degree Vectors

The next piece of the information in a state is a *degree vector*:

Definition 3.5. *A degree vector for a set $S \subseteq V \setminus K$ is a vector $\rho = (\rho_v)_{v \in S}$, where ρ_v is an integer in $[1, d_v]$ for every $v \in S$.*

Supposedly, ρ_v will be the original degree of v in the tree T.

Definition 3.6 (Consistency of degree vectors)**.** *Given a root-portals-pair (r', S), an allowable child-pair $((r', S_1), (r'', S_2))$ of (r', S), three degree vectors ρ, ρ^1 and ρ^2 for S, S_1 and S_2 respectively, we say ρ^1 and ρ^2 are consistent with ρ, if*

- *for every $v \in S_1 \setminus \{r''\}$, we have $\rho_v = \rho^1_v$,*
- *for every $v \in S_2 \setminus \{r''\}$, we have $\rho_v = \rho^2_v$ and*
- *$\rho^1_{r''} = \rho^2_{r''}$.*

So, the degree vectors are consistent if there is no contradictory information among them.

Definition 3.7 (Edge/Triple Agreeing with Degree Vector)**.** *Given a root-portals-pair (r', S) with $|S| \leq 2$, a degree vector ρ for S, and an edge $(r', v) \in E$ with $\{r', v\} \setminus K = S$, we say (r', v) agrees with ρ if $\rho_{r'} = (\phi_v(\rho_v)$ or $1)$, where $(\phi_v(\rho_v)$ or $1)$ denotes $\phi_v(\rho_v)$ if ρ_v is defined (i.e, if $v \in S$) and 1 otherwise.*

Similarly, given a root-portals-pair (r', S) with $|S| \leq 3$, a degree vector ρ for S, and two edges $(r', v), (r', v') \in E$ such that $\{r', v, v'\} \setminus K = S$, we say the triple (r', v, v') agrees with ρ if $\rho_{r'} = (\phi_v(\rho_v)$ or $1) + (\phi_{v'}(\rho_{v'})$ or $1)$.

Notice that in the above definition either $v \in S$ or $v \in K$. In the former case, ρ_v is defined; in the latter case ρ_v is not defined but we know ϕ_v is identically 1. The same argument holds for v'. The definition corresponds to the case when T' is a base case of the recursive balanced tree partitioning, i.e., T' contains only 1 level of edges. If T' contains an edge $e = (r', v)$, then the portal set of T' is $\{r', v\} \setminus K$. We shall have $\rho_{r'} = \phi_v(\rho_v)$ or 1. Thus, if ρ is restricted to the portal set, we have $\rho_{r'} = (\phi_v(\rho_v)$ or $1)$. Similarly, if T' contains 3 vertices (r', v, v') with r' being the root, then we must have $\rho_{r'} = (\phi_v(\rho_v)$ or $1) + (\phi_{v'}(\rho_{v'})$ or $1)$.

3.3 States and Good State-Trees

With degree vectors, we can define states and good state-trees:

Definition 3.8. *A* state *is a tuple (r', S, ρ) where (r', S) is a root-portals-pair and ρ is a degree vector for S.*

The state of the tree T' w.r.t T is the tuple (r', S, ρ) with $r' = \mathrm{root}(T')$, S being the set of portals in T', and ρ being the vector of original degrees of vertices in S w.r.t the tree T.

Definition 3.9 (Good State Trees)**.** A good state tree *is a full binary tree τ of depth at most h, where every node p is associated with a state (r'_p, S_p, ρ^p), and every leaf o is associated with either an edge $e_o \in E$ or a triple ξ_o such that the following conditions hold.*

(3.9a) $\left(r'_{\mathrm{root}(\tau)}, S_{\mathrm{root}(\tau)}\right) = (r, \{r\})$.

(3.9b) For any leaf o of τ, either e_o or ξ_o agrees with ρ^o.

(3.9c) For an internal node p in τ, letting q and o be the left and right children of p, then the pair $((r'_q, S_q), (r'_o, S_o))$ is an allowable child-pair of (r'_p, S_p) (so, $r'_q = r'_p \neq r'_o$), and ρ^q and ρ^o are consistent with ρ^p.

We say that a terminal $t \in K$ is involved *in a good state tree τ if there exists a leaf o of τ with $t = \text{tail}(e_o)$, or $t \in \{\text{second}(\xi_o), \text{third}(\xi_o)\}$.*

Given a good state tree τ, and a leaf o in τ, we define the cost $c(o)$ as follows. If e_o is defined, then we define $c(o) = c_{e_o}$; otherwise, define $c(o) = c_{(r'_o, \text{second}(\xi_o))} + c_{(r'_o, \text{third}(\xi_o))}$. The cost of a state-tree τ is defined as $\text{cost}(\tau) := \sum_{o \text{ leaf of } \tau} c(o)$.

We remark that the degree bounds are ensured by Property (3.9b), as the outgoing edges for a vertex v (or actually, for a copy of a vertex), are decided by the state of a leaf p of τ. Now, we can use the fact that the original degrees of portals are stored in ρ, and non-portals count as a single outgoing edge, together with the recursive definition of original degree, to compute the original degree of v and check the degree bounds. The checks corresponding to this process follow from Definitions 3.5 and 3.7.

4 Reduction to Finding Good State-Trees

4.1 From a Valid Tree to a Good State-Tree Involving All Terminals

In this section, we show that the decomposition tree of the optimum tree T^* can be turned into a good state tree τ^* with cost $\text{cost}(\tau^*) = \text{cost}(T^*)$ that involves all terminals. As we alluded, the state tree τ^* is constructed by taking the state for each node in the decomposition tree for T^*. Formally, it is obtained by calling gen-state-tree(T^*) (defined in Algorithm 1). In the algorithm ρ^{T^*} is the vector of original degrees of all vertices in T^*. The procedure is only for needed for the purpose of analysis; it is not a part of our algorithm.

Algorithm 1 gen-state-tree(T')

1: create a node p with $r'_p = \text{root}(T')$, $S_p =$ portals of T' and ρ^p being ρ^{T^*} restricted to S_p
2: **if** T' has only 1 level of edges **then**
3: **if** T' contains a single edge e **then** let $e_p = e$ and **return** the single node p
4: **otherwise**, T' contains two edges (r', v) and (r', v'), let $\xi_p = (r', v, v')$ and **return** p
5: apply balanced tree partitioning to decompose T' into T'_1 and T'_2
6: $\tau_1 \leftarrow$ gen-state-tree(T'_1), $\tau_2 \leftarrow$ gen-state-tree(T'_2)
7: **return** the tree τ obtained by combining p, τ_1 and τ_2 with edges $(p, \text{root}(\tau_1))$ and $(p, \text{root}(\tau_2))$, with $\text{root}(\tau_1)$ and $\text{root}(\tau_2)$ being the left and right children of p respectively

Lemma 4.1. *τ^* is a good state tree involving all terminals and* $\text{cost}(\tau^*) = \text{cost}(T^*)$.

Proof. We first show that τ^* is a good state tree, by showing that it satisfies all the properties in Definition 3.9. Property (3.9a) trivially holds by the way we define the parameters for the root recursion of gen-state-tree. Property (3.9b) holds by that each ρ^p is ρ^{T^*} restricted to S_p. Property (3.9c) follows from the same facts and Claim 3.4. $\text{cost}(\tau^*) = \sum_{e \in E_{T^*}} c_e = \text{cost}(T^*)$ since every edge in T^* counted exactly once in τ^*. □

4.2 From a Good State Tree to a Good Multi-Tree

Now we focus on the other direction of the reduction. Suppose we are given a good state tree τ, and our goal is to construct a good multi-tree T with $\text{cost}(T) = \text{cost}(\tau)$. Moreover, if a terminal

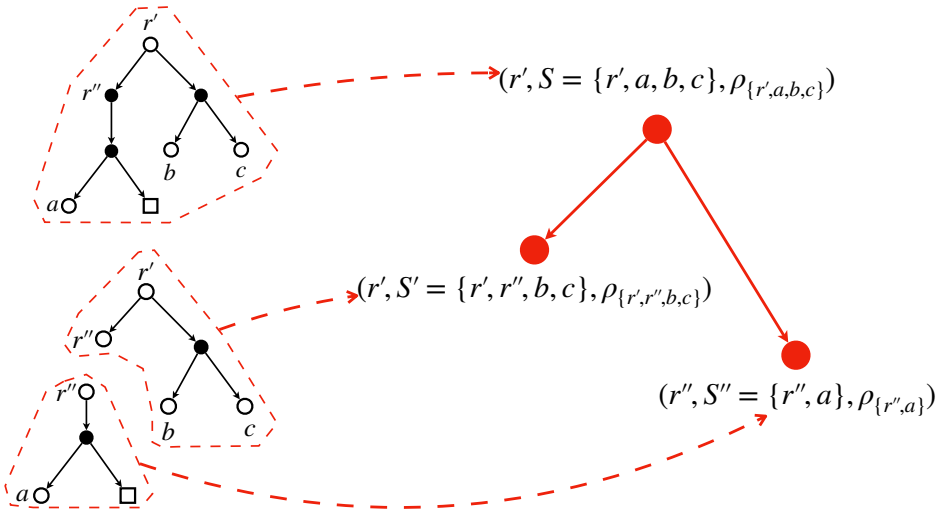

Figure 4: The figure represents the encoding of a tree as a state tree. Each state is a succinct encoding of a tree that captures its boundary information (i.e. the portals). The state tree then encodes how a tree is further decomposed to subtrees. In the example we have a one-step decomposition – an allowable child-pair, and it corresponds to an 1-level state tree with three states.

$t \in K$ is involved in τ, then T contains a copy of t.

The multi-tree T is constructed by joining the edges associated with all leaf nodes o in τ using a recursive procedure. For each node p in τ we shall construct a multi-tree T_p for p, as well as a mapping π_p from S_p to vertices in T_p. The multi-tree T_p and the mapping π_p satisfy the following properties:

(P1) For every $v \in S_p$, we have $\text{label}(\pi_p(v)) = v$, that is, $\pi_p(v)$ is a copy of v.

(P2) $\pi_p(r'_p) = \text{root}(T_p)$.

In particular, the two properties imply that $\text{root}(T_p)$ is a copy of r'_p.

The trees and mappings are constructed from the bottom to the top of the tree τ. Focus on a leaf node p with $e_p = (r', v)$. If e_p is defined, then T_p only contains a copy of the edge (r', v). π_p maps r' to the copy of r', and if $v \notin K$ (thus, $v \in S_p$), v to the copy of v in T_p. Otherwise ξ_p is defined. Then T_p contains a tree with two edges: a copy of $(r'_p, \text{second}(\xi_p))$ and a copy of $(r'_p, \text{third}(\xi_p))$. π_p can also be defined naturally.

Now consider the case that p is an internal node and let q and o be its left and right children. Then, we have $r'_p = r'_q, r'_o \notin S_p, S_q \cup S_o = S_p \cup \{r'_o\}$ and $S_q \cap S_o = \{r'_o\}$ by Property (3.9c). Then we identify $\pi_q(r'_o)$ with $\pi_o(r'_o) = \text{root}(T_o)$, and then the multi-tree T_p is the new tree containing vertices in T_q and T_o. Notice that both $\pi_q(r'_o)$ and $\pi_o(r'_o)$ are copies of r'_o; thus the obtained T_p can be well-defined. The mapping π_p is just the combination of π_q and π_o: For a vertex $v \in S_q$, let $\pi_p(v) = \pi_q(v)$; for a vertex $v \in S_o$, let $\pi_p(v) = \pi_o(v)$; since $S_q \cap S_o = \{r'_o\}$ and we identified $\pi_q(r'_o)$ with $\pi_o(r'_o)$, the mapping is well-defined. Also, it is easy to see that (P1) and (P2) holds for T_p and π_p.

Our final multi-tree for τ will be $T = T_{\text{root}(\tau)}$. It is straightforward to see that if $t \in K$ is involved in τ, then T contains a copy of t. Notice that all the ρ^p-vectors are consistent with each other, and for every leaf o, e_o or ξ_o agrees with ρ^o. Thus, aggregating all the ρ^p vectors will recover the vector ρ^T of original degrees of vertices in ρ^T. So, the multi-tree T is good since every v in T has $\rho_v^T \in [1, d_v]$. The cost of T is $\sum_{e \in E_T} c_e = \sum_{o:\text{ leaves of } \tau} c(o) = \text{cost}(\tau)$. The procedure is presented in Algorithm 2.

5 Finding a Good State Tree using LP Rounding

5.1 Extended State Trees and Construction of $\mathbf{T}^0$

With the relationship between good multi-trees and good state trees established, we can now focus on the problem of finding a good state-tree of small cost involving many terminals. We shall construct a quasi-polynomial sized tree $\mathbf{T}^\circ$ so that every good state-tree τ corresponds a sub-tree $\mathbf{T}$ of $\mathbf{T}^\circ$ satisfying some property. Roughly speaking, $\mathbf{T}^\circ$ is the "super-set" of all potential good state-trees τ. However, since the consistency conditions are defined over three states for a parent and its two children, it is more convenient to insert a "virtual" node between every internal node and its two children. Also, it is convenient to break a leaf state node o into two nodes, one containing the state information and the other containing e_o or ξ_o. Formally, for a good state-tree τ, we construct a correspondent tree $\mathbf{T}$ as follows.

1. Let $\mathbf{T}$ be a copy of τ. All nodes in $\mathbf{T}$ are called *state nodes*.
2. For every internal state node p in $\mathbf{T}$ with left and right children p_1 and p_2, we create a *virtual node* q and replace the two edges (p, p_1) and (p, p_2) with 3 edges $(p, q), (q, p_1)$ and (q, p_2); p_1 is still the left child and p_2 is the right child.
3. For every leaf state node p, we create a *base node* o and let o be the child of p. Then we move the e_p or ξ_p information from the node p to node o: If e_p is defined, then we let $e_o = e_p$ and undefine e_p; otherwise, let $\xi_o = \xi_p$ and undefine ξ_p.
4. We add a *super node* $\mathbf{r}$ and an edge from $\mathbf{r}$ to the root of $\mathbf{T}$. $\mathbf{r}$ will be the new root for $\mathbf{T}$.

We call this $\mathbf{T}$ the *extended state-tree* for τ; we say $\mathbf{T}$ is good if its correspondent τ is good. Clearly, there is a 1-to-1 correspondence between good state trees and good extended state trees.

Algorithm 2 gen-multi-tree(p, τ)

if p is a leaf **then**
  if e_p is defined **then**
    return a copy of e_p
  else if ξ_p is defined **then**
    return a copy of the star with edges $\{(r'_p, \text{second}(\xi_p)), (r'_p, \text{third}(\xi_p))\}$
else if p is a node with children q (left) and o (right) **then**
  $T_q \leftarrow$ gen-multi-tree(q, τ), $T_o \leftarrow$ gen-multi-tree(o, τ)
  Let T'_p be the union of T_q and T_o
  Obtain T_p from T'_p by unifying the root of T_o with the copy of the portal $r'_o \in S_q$ in T_q
  Return T_p

Our $\mathbf{T}^\circ$ will be the "super-set" of all potential good extended state trees $\mathbf{T}$. Formally, we create a super node $\mathbf{r}$ to be the root of $\mathbf{T}^\circ$. Then, for every $\rho_r \in [1, d_r]$, we call cnstr-$\mathbf{T}^\circ(0, r, \{r\}, \rho = (\rho_r))$ to obtain a tree and let its root be a child of $\mathbf{r}$.

Algorithm 3 cnstr-$\mathbf{T}^\circ(h', r', S, \rho)$

create a state node p with $(r'_p, S_p, \rho^p) = (r', S, \rho)$
for every $(r', v) \in E$ such that $\{r', v\} \setminus K = S$ and (r', v) agrees with ρ **do**
    create a "base node" o with $e_o = (r', v)$ and let o be a child of p
    let $c(o) = c_{(r',v)}$
for every $(r', v), (r', v') \in E$ such that $\{r', v, v'\} \setminus K = S$ and (r', v, v') agrees with ρ **do**
    create a "base node" o with $\xi_o = (r', v, v')$ and let o be a child of p
    let $c(o) = c_{(r',v)} + c_{(r',v')}$
if $h' < h$ **then**
    for every allowable child-pair $((r', S_1), (r'', S_2))$ of (r', S) **do**
        for every pair of degree vectors ρ^1 for S_1 and ρ^2 for S_2 such that ρ^1 and ρ^2 are consistent with ρ **do**
            create a "virtual node" q and let q be a child of p
            $\mathbf{T}_1 \leftarrow$ cnstr-$\mathbf{T}^\circ(h' + 1, r', S_1, \rho^1)$
            $\mathbf{T}_2 \leftarrow$ cnstr-$\mathbf{T}^\circ(h' + 1, r'', S_2, \rho^2)$
            let the left and right sub-trees of q be $\mathbf{T}_1$ and $\mathbf{T}_2$ respectively
return the tree $\mathbf{T}$ rooted at p

The following claim is immediate from the construction of $\mathbf{T}^\circ$.

Claim 5.1. *A subtree $\mathbf{T}$ of $\mathbf{T}^\circ$ with* $\mathrm{root}(\mathbf{T}) = \mathrm{root}(\mathbf{T}^\circ)$ *is a good extended state tree if and only if the following happens:*

- *The super node in $\mathbf{T}$ has exactly one child (which is a state node).*
- *Each state node in $\mathbf{T}$ has exactly one child (which is an base node or a virtual node).*
- *For each virtual node q in $\mathbf{T}$, both q's children in $\mathbf{T}^\circ$ are in $\mathbf{T}$.*

On the other hand, every good extended tree $\mathbf{T}$ of depth at most $h+1$ is a sub-tree of $\mathbf{T}^\circ$ with root being $\mathrm{root}(\mathbf{T}^\circ)$.

Also, we say that a vertex v is involved in $\mathbf{T}$ if there is an base node o in $\mathbf{T}$ with $v = \mathrm{tail}(e_o)$ or $v \in \{\mathrm{second}(\xi_o), \mathrm{third}(\xi_o)\}$. The cost of $\mathbf{T}$, denoted as $\mathrm{cost}(\mathbf{T})$, is defined the sum of $c(o)$ over all base nodes in $\mathbf{T}$. So, the problem now becomes finding a small-cost good extended state tree in $\mathbf{T}^\circ$ that involves each terminal with large probability.

5.2 LP Formulation

We formulate an LP relaxation for our task. Let $\mathbf{V}^\circ$ be the set of nodes in $\mathbf{T}^\circ$, $\mathbf{r} = \mathrm{root}(\mathbf{T}^\circ)$, and let $\mathbf{V}^\circ_{\mathrm{state}}, \mathbf{V}^\circ_{\mathrm{virt}}$ and $\mathbf{V}^\circ_{\mathrm{base}}$ be the sets of state, virtual and base nodes in $\mathbf{T}^\circ$, respectively. Notice that there is only one super node, which is the root $\mathbf{r}$. For every $v \in V$, let $\mathbf{O}_v = \left\{o \in \mathbf{V}^\circ_{\mathrm{base}} : v = \mathrm{tail}(e_o) \text{ or } v \in \{\mathrm{second}(\xi_o), \mathrm{third}(\xi_o)\}\right\}$ be the set of base nodes involving v. Let $\mathbf{T}^*$ be our

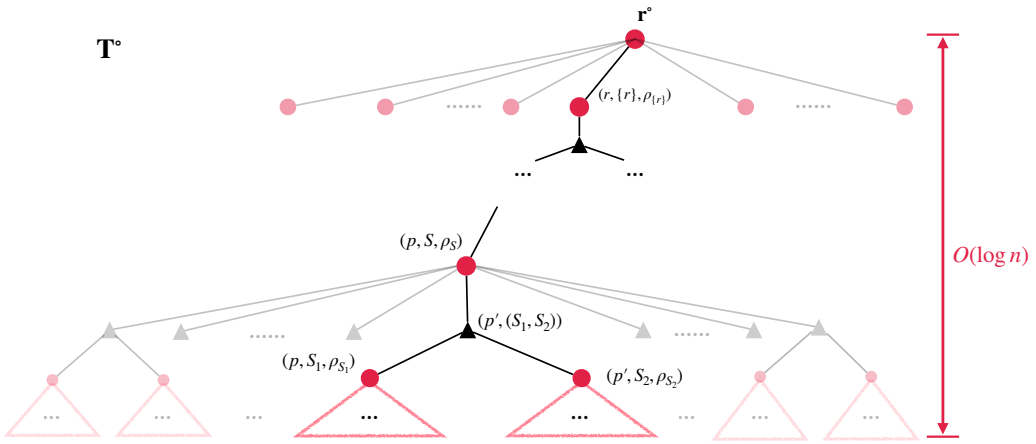

Figure 5: $\mathbf{T}^\circ$ is the union of all good extended state trees. Virtual nodes are denoted as triangles, each of which represents a way to decompose (the subtree corresponding to) its parent state node. For example, $(p', (S_1, S_2))$ means we break the tree at vertex p' (thus creating a new portal p'), and divide the portals S of the original tree (along with p') to the two new subtrees as S_1 and S_2.

target good extended state tree; this is the tree correspondent to the good state tree τ^*. Then, in our LP, we have a variable x_p for every $p \in \mathbf{V}^\circ$, that indicates whether p is in the $\mathbf{T}^*$ or not.

$$\min \quad \sum_{o \in \mathbf{V}^\circ_{\text{base}}} x_o c(o) \tag{1}$$

$$\sum_{q \in \Lambda_{\mathbf{T}^\circ}(p)} x_q = x_p, \quad \forall p \in \mathbf{V}^\circ_{\text{state}} \cup \{\mathbf{r}\} \tag{2}$$

$$x_p = x_q, \quad \forall q \in \mathbf{V}^\circ_{\text{virt}}, p \in \Lambda_{\mathbf{T}^\circ}(q) \tag{3}$$

$$x_p \in [0,1], \forall p \in \mathbf{V}^\circ \tag{4}$$

$$\sum_{o \in \Lambda^*_{\mathbf{T}^\circ}(p) \cap \mathbf{O}_t} x_o \le x_p, \forall p \in \mathbf{V}^\circ, t \in K \tag{5}$$

$$\sum_{o \in \mathbf{O}_t} x_o = 1, \quad \forall t \in K \tag{6}$$

The objective function of LP (1) is to minimize the total cost of all leaves in $\mathbf{T}^*$. (2) requires that for every state or super node p in $\mathbf{T}^*$, exactly one child of p is in $\mathbf{T}^*$. (3) requires that a virtual node q in $\mathbf{T}^*$ has both its children in $\mathbf{T}^*$. (5) says for every node p in $\mathbf{T}^*$ and every terminal $t \in K$, there is at most one descendant base node o of p that is in $\mathbf{O}_t$. In the whole tree $\mathbf{T}^*$, exactly one leaf node o has $t = \text{tail}(e_o)$ or $t \in \{\text{second}(\xi_o), \text{third}(\xi_o)\}$, for every $t \in K$ (Constraint (6)); in the LP, all the variables are between 0 and 1 (Constraint (4)).

Notice that (5) for $p = \mathbf{r}$ and any $t \in K$ and (6) for the same t imply that $x_{\mathbf{r}} = 1$. (2) and (3) imply that the x values over the nodes of a root-to-leaf path in $\mathbf{T}^\circ$ are non-increasing.

5.3 Rounding Algorithm

Given a valid solution x to LP (1), our rounding algorithm will round it to obtain set $\mathbf{V} \subseteq \mathbf{V}^\circ$, which induces a good state tree. The algorithm is very similar to that of [9] with the only one difference: For every state node or super-node p that is added to $\mathbf{V}$, we add exactly one child q of p to $\mathbf{V}$, while the algorithm of [9] makes independent decisions for each child. The algorithm is formally described in Algorithm 4. In the main algorithm, we simply call round($\mathbf{r}$). It is straightforward to

Algorithm 4 round(p)

1: **if** $p \in \mathbf{V}^\circ_{\text{state}} \cup \{\mathbf{r}\}$ **then**
2: randomly choose a child q of p according to probability vector $\left(\frac{x_q}{x_p}\right)_{q \in \Lambda_{\mathbf{T}^\circ}(p)}$
3: **return** $\{p\} \cup \text{round}(q)$
4: **else if** $p \in \mathbf{V}^\circ_{\text{virt}}$ **then**
5: **return** $\{p\} \cup \text{round}(\text{left child of } p) \cup \text{round}(\text{right child of } p)$
6: **else**
7: **return** $\{p\}$

see that the tree induced by round($\mathbf{r}$) is a good extended state tree. The following claim also holds:

Claim 5.2. *Let $p \in \mathbf{V}^\circ$ and $q \in \Lambda^*_{\mathbf{T}^\circ}(p)$. Let $\mathbf{V}$ be the random set returned by* round(p). *Then we have* $\Pr[q \in \mathbf{V}] = \frac{x_q}{x_p}$.

Applying the above claim for $p = \mathbf{r}$ and every $q \in \mathbf{V}^\circ_{\text{base}}$, we have that the expected cost of the tree induced by $\mathbf{V}$ is exactly cost(x).

The main theorem we need about the rounding algorithm is as follows:

Theorem 5.3. *Let $\mathbf{V}$ be the random set returned by* round($\mathbf{r}$). *Then, for any terminal $t \in K$, we have*

$$\Pr[\mathbf{V} \cap \mathbf{O}_t \neq \emptyset] \geq \frac{1}{h+1}.$$

Theorem 5.3 was proved [9] for the original rounding algorithm and was reproved in [22]. However, adapting the analysis to our slightly different rounding algorithm is straightforward and thus we omit the proof of the theorem here.

We now wrap up and finish the proof of the main theorem (Theorem 2.4) except for Property (2.4c), which will be proved in Section 5.4.

We solve LP(1) to obtain a solution x. Notice that $\text{cost}(x) \leq \text{cost}(\mathbf{T}^*) = \text{cost}(\tau^*) = \text{cost}(T^*)$. Let $\mathbf{V} \leftarrow \text{round}(\mathbf{r})$. Then by Claim 5.1 and the rounding algorithm, the tree $\mathbf{T}$ induced by $\mathbf{V}$ is a good extended state tree. Let τ be the good state tree correspondent to $\mathbf{T}$, and let T be the good multi-tree in G constructed using the procedure in Section 4.2. The cost of the multi-tree T is at most cost(x). By Theorem 5.3, for every $t \in K$, the probability that t is involved T is at least $1/(h+1) = \Omega(1/\log n)$.

Let us consider the running time of the algorithmic framework, which is polynomial on the size of the tree $\mathbf{T}^\circ$. First notice that if $((r', S_1), (r'', S_2))$ is an allowable child pair of (r', S), then we have $|S_1|, |S_2| \leq |S| + 1$ since $S_1 \cup S_2 = S \cup \{r''\}$. Thus, a state-node p at the h'-th level in $\mathbf{T}^\circ$ (the children of $\mathbf{r}$ have level 0 and for simplicity we do not consider super and virtual nodes when counting levels) has $|S_p| \leq h' + 1$. Thus, every state node p in $\mathbf{T}^\circ$ has $|S_p| \leq h + 1$.

Then we consider the degree of the tree $\mathbf{T}^\circ$, which is the maximum number of possible children of a state node p with $(r'_p, S_p, \rho^p) = (r', S, \rho)$. First, there are at most $n \times 2^{|S_p|} \leq n \cdot 2^{h+1}$ different allowable child pairs $((r', S_1), (r'', S_2))$ of the pair (r', S): there are at most n choices for r'' and 2^h ways to split S into S_1 and S_2. Then, for a fixed allowable child pair $((r', S_1), (r'', S_2))$ we consider the number of pairs of degree vectors $\left(\rho^1, \rho^2\right)$ such that ρ^1 and ρ^2 are consistent with ρ. This is determined by the value of $\rho^1_{r''} = \rho^2_{r''}$, which has at most $d_{\max}$ possibilities. So, the number of virtual children of a state node is at most $n \cdot 2^{h+1} \cdot d_{\max} = O(\text{poly}(n)$ since $h = O(\log n)$. The number of child base nodes of p is at most n^2. Since the height of the tree $\mathbf{T}^\circ$ is at most $O(\log n)$, its size bounded by $(\text{poly}(n))^{O(\log n)} = n^{O(\log n)}$. So the running time of the LP rounding algorithm is $n^{O(\log n)}$. This finishes the proof of Theorems 2.4 except for Property (2.4c).

5.4 Concentration Bound on Number of Copies of a Vertex Appearing in T

Finally, we prove Property (2.4c) in Theorem 2.4. To this end, we shall fix a vertex $v \in V$. For every vertex $p \in \mathbf{V}^\circ$, let $z_p = \sum_{o \in \Lambda^*_{\mathbf{T}^\circ}(p) \cap \mathbf{O}_v} x_o$. By Constraint (5), we have $z_p \leq x_p$. Let $m_p = |\Lambda^*_{\mathbf{T}^\circ}(p) \cap \mathbf{O}_v \cap \mathbf{V}|$ be the total number of nodes in $\Lambda^*_{\mathbf{T}^\circ}(p) \cap \mathbf{O}_v$ that are selected by the rounding algorithm.

As is typical, we shall introduce a parameter $s > 0$ and consider the expectation of the random exponential variables $\mathbf{e}^{sm_p}$ (we use $\mathbf{e}$ for the natural constant). We shall bound $\mathbb{E}[\mathbf{e}^{sm_p} | p \in \mathbf{V}]$ from bottom to top by induction. So, in this proof, it is more convenient for us to use a different definition of levels: the level of a node p in $\mathbf{T}^\circ$ is the maximum number of edges in a path in $\mathbf{T}^\circ$ starting from p. So, the leaves have level 0 and for an internal node p in $\mathbf{T}^\circ$, the level of p is 1 plus the maximum of the level of q over all children q of p. We define an α_i for every integer $i \geq 0$ as $\alpha_0 = \mathbf{e}^s$ and $\alpha_i = \mathbf{e}^{\alpha_{i-1}-1}, \forall i \geq 1$. Notice that $\alpha_0, \alpha_1, \cdots$ is an increasing sequence. Thus, we can induce the following lemma.

Lemma 5.4. *For any node p be in $\mathbf{T}^\circ$ of level at most i, $\mathbb{E}\left[\mathbf{e}^{sm_p} \middle| p \in \mathbf{V}\right] \leq \alpha_i^{z_p/x_p}$.*

Proof. We prove the lemma by induction on i. If $i = 0$, then p is a leaf, and thus, we have either $z_p = 0$ or $z_p = x_p$, depending on whether $p \in \mathbf{O}_v$ or not. If $z_p = 0$, then m_p is always 0, and thus, $\mathbb{E}\left[\mathbf{e}^{sm_p} \middle| p \in \mathbf{V}\right] = 1 = \alpha_0^{z_p/x_p}$. If $z_p = x_p$, then m_p is always 1 (conditioned on $p \in \mathbf{V}$), and thus, $\mathbb{E}\left[\mathbf{e}^{sm_p} \middle| p \in \mathbf{V}\right] = \mathbf{e}^s = \alpha_0^{z_p/x_p}$. So, the lemma holds if $i = 0$.

Now, let $i \geq 1$ be any integer and we assume the lemma holds for $i - 1$. We shall prove that it also holds for i. Focus on a node p of level at most i. Then all children q of p have level at most $i - 1$. If p is a virtual node, then $p \in \mathbf{V}$ implies that both children of p in $\mathbf{V}$. Since the two children are handled independently in the rounding algorithm, we have

$$\begin{aligned}\mathbb{E}\left[\mathbf{e}^{sm_p} \middle| p \in \mathbf{V}\right] &= \prod_{q \in \Lambda_{\mathbf{T}^\circ}(p)} \mathbb{E}\left[\mathbf{e}^{sm_q} \middle| p \in \mathbf{V}\right] = \prod_{q \in \Lambda_{\mathbf{T}^\circ}(p)} \left[\frac{x_q}{x_p} \cdot \mathbb{E}[\mathbf{e}^{sm_q} | q \in \mathbf{V}] + 1 - \frac{x_q}{x_p}\right] \\ &= \prod_{q \in \Lambda_{\mathbf{T}^\circ}(p)} \left[1 + \frac{x_q}{x_p}\Big(\mathbb{E}[\mathbf{e}^{sm_q} | q \in \mathbf{V}] - 1\Big)\right].\end{aligned}$$

If p is the super node or a state node, then we have $\sum_{q \in \Lambda_{\mathbf{T}^\circ}(p)} x_q = x_p$. Conditioned on $p \in \mathbf{V}$, the

rounding procedure adds exactly one child q of p to $\mathbf{V}$. Then, we have

$$\mathbb{E}\left[\mathbf{e}^{sm_p}\middle|p\in\mathbf{V}\right] = \sum_{q\in\Lambda_{\mathbf{T}^\circ}(p)}\frac{x_q}{x_p}\,\mathbb{E}\left[\mathbf{e}^{sm_q}\middle|q\in\mathbf{V}\right] = 1+\sum_{q\in\Lambda_{\mathbf{T}^\circ}(p)}\frac{x_q}{x_p}\Big(\mathbb{E}[\mathbf{e}^{sm_q}\big|q\in\mathbf{V}]-1\Big)$$
$$\leq \prod_{q\in\Lambda_{\mathbf{T}^\circ}(p)}\left[1+\frac{x_q}{x_p}\Big(\mathbb{E}[\mathbf{e}^{sm_q}|q\in\mathbf{V}]-1\Big)\right].$$

Thus, we always have

$$\mathbb{E}\left[\mathbf{e}^{sm_p}\middle|p\in\mathbf{V}\right] \leq \prod_{q\in\Lambda_{\mathbf{T}^\circ}(p)}\left[1+\frac{x_q}{x_p}\Big(\mathbb{E}[\mathbf{e}^{sm_q}|q\in\mathbf{V}]-1\Big)\right]$$
$$\leq \prod_{q\in\Lambda_{\mathbf{T}^\circ}(p)}\left[1+\frac{x_q}{x_p}\big(\alpha_{i-1}^{z_q/x_q}-1\big)\right] \qquad \text{by induction hypothesis}$$
$$\leq \exp\left[\sum_{q\in\Lambda_{\mathbf{T}^\circ}(p)}\frac{x_q}{x_p}\big(\alpha_{i-1}^{z_q/x_q}-1\big)\right] \leq \exp\left[\frac{z_p}{x_p}(\alpha_{i-1}-1)\right] = \alpha_i^{z_p/x_p}. \quad \text{since } 1+\theta\leq e^\theta \text{ for every } \theta$$

To see the second inequality in the last line, we notice that (i) $\alpha_{i-1}^\theta - 1$ is a convex function of θ that is upper bounded by $\theta(\alpha_{i-1}-1)$ in the interval $\theta\in[0,1]$, and (ii) $z_q/x_q\in[0,1]$ for every q in the summation, which allows us to bound each term by $\frac{x_q}{x_p}\frac{z_q}{x_q}(\alpha_{i-1}-1)\leq\frac{z_p}{x_p}(\alpha_{i-1}-1)$. Since $\sum_{q\in\Lambda_{\mathbf{T}^\circ}(p)}\frac{x_q}{x_p}\cdot\frac{z_q}{x_q}=\frac{z_p}{x_p}$, the quantity inside $\exp(\cdot)$ has maximum value $\frac{z_p}{x_p}(\alpha_{i-1}-1)$. The equality in the last line is by the definition of α_i. □

Let $h'=\Theta(h)=\Theta(\log n)$ be the level of the root. Now, we set $s=\ln(1+\frac{1}{2h'})$. We prove inductively the following lemma:

Lemma 5.5. *For every $i\in[0,h']$, we have $\alpha_i\leq 1+\frac{1}{2h'-i}$.*

Proof. By definition, $\alpha_0=\mathbf{e}^s=1+\frac{1}{2h'}$ and thus the statement holds for $i=0$. Let $i\in[1,h']$ and assume the statement holds for $i-1$. Then, we have

$$\alpha_i=\mathbf{e}^{\alpha_{i-1}-1}\leq\mathbf{e}^{1+\frac{1}{2h'-i+1}}\leq 1+\frac{1}{2h'-i+1}+\left(\frac{1}{2h'-i+1}\right)^2$$
$$=1+\frac{2h'-i+2}{(2h'-i+1)^2}\leq 1+\frac{1}{2h'-i}.$$

The first inequality used the induction hypothesis and the second one used that for every $\theta\in[0,1]$, we have $e^\theta\leq 1+\theta+\theta^2$. □

So, by Lemma 5.4 and 5.5, we have $\mathbb{E}[\mathbf{e}^{sm_\mathbf{r}}]\leq\alpha_{h'}^1\leq 1+\frac{1}{h'}=1+O\left(\frac{1}{\log n}\right)$. This finishes the proof of Property (2.4c) in Theorem 2.4.

6 Bicriteria-Approximation Algorithm for Degree-Bounded Group Steiner Tree on Trees

In this section, we prove Theorem 1.2, which is repeated here.

Theorem 1.2. *There is a randomized $\big(O(\log n \log k), O(\log n)\big)$-bicriteria approximation algorithm for the degree-bounded group Steiner tree problem on trees, running in polynomial time.*

We first set up some notations for the theorem. Recall that T° is the input tree, V° denotes the set of vertices of T°, and r denotes the root of T°. For simplicity, we assume the costs are on the vertices instead of edges: Every vertex $u \in V^\circ$ has a cost $c_u \geq 0$. Notice that this does not change the problem. We have k groups indexed by $[k]$. For each group $t \in [k]$, we are given a set $O_t \subseteq V^\circ$ of leaves in T°. W.l.o.g, we assume all O_t's are disjoint. Every vertex $v \in V$ is given a degree bound D_v. The goal of the problem is then to output the smallest cost subtree T of T° that satisfies the degree constraints and contains the root r and one vertex from each O_t, $t \in [k]$. Since now we only have one tree T°, we use the following notations for children and descendants: For every vertex $u \in V^\circ$, let Λ_u denote the set of children of u in T°, and Λ_u^* to denote the set of descendants of u in T° (including u itself).

Now we describe the LP relaxation we use for our problem. For every vertex $u \in T^\circ$, we use x_u to indicate whether u is chosen or not (in the correspondent integer program). LP (7) is a valid LP relaxation for the DB-GST-T problem:

$$\min \quad \sum_{u \in V^\circ} c_u x_u \qquad \text{s.t.} \tag{7}$$

$$x_v \leq x_u \qquad \forall u \in V^\circ, v \in \Lambda_u \tag{8}$$

$$\sum_{o \in O_t} x_o = 1 \qquad \forall t \in [k] \tag{9}$$

$$\sum_{o \in O_t \cap \Lambda_u^*} x_o \leq x_u \qquad \forall t \in [k], \forall u \in V^\circ \tag{10}$$

$$\sum_{v \in \Lambda_u} x_v \leq d_u \cdot x_u \qquad \forall u \in V^\circ \tag{11}$$

$$x_u \in [0,1] \qquad \forall u \in V^\circ \tag{12}$$

In the correspondent integer program, the objective we try to minimize is $\sum_{u \in V^\circ} c_u x_u$, i.e, the total cost of all verticies we choose. Constraint (8) says that if we choose a vertex v then we must choose its parent u. Constraint (9) requires for every group t, exactly one vertex in O_t is added to the tree. Constraint (10) holds since if u is chosen, at most one vertex in $\Lambda_u^* \cap O_t$ is chosen for every group t. Constraint (11) is the degree constraint. In the LP relaxation, we require each x_u to take value in $[0,1]$ (Constraint (12)). Notice that (9) and (10) for the root r imply that $x_r = 1$.

Modifying the LP solutions. Solving LP (7), we can obtain the optimum LP solution $(x_u)_{u \in V^\circ}$. In our rounding algorithm, it would be convenient if every x_u is a (non-positive) integer power of 2 that is not too small. So, we shall modify the LP solution using the following operations, which may violate many of the LP constraints slightly. For every $v \in V^\circ$ with $x_v < \frac{1}{2n}$, we change x_v to 0. This can only decrease the cost of the solution. It is easy to see that Constraints (8), (10) and (11) will not be violated. Constraint (9) may not hold any more, but we still have $\sum_{v \in O_t} x_v \geq 1 - n \times \frac{1}{2n} \geq \frac{1}{2}$ for every $t \in [k]$. We can remove all vertices v with $x_v = 0$ from the instance and thus assume $x_v \geq \frac{1}{2n}$ for every $v \in V^\circ$. Next, we increase each x_v to the smallest (non-positive) integer power of 2 that is greater than or equal to x_v. This will violate many constraints in the LP by a factor of 2. We list the properties that our new vector $(x_u)_{u \in V^\circ}$ has:

(P1) For every $u \in V^\circ$, x_u is an integer power of 2 between $\frac{1}{2n}$ and 1.

(P2) The x values along any root-to-leaf path in T° is non-increasing.

(P3) $\sum_{o\in O_t} x_o \in [\frac{1}{2}, 2]$ for every group $t \in [k]$.

(P4) $\sum_{o\in O_t\cap\Lambda_u^*} x_o \le 2x_u$ for every $t \in [k]$ and $u \in V^\circ$.

(P5) $\sum_{v\in\Lambda_u} x_v \le 2d_u x_u$ for every $u \in V^\circ$.

(P6) $\sum_{u\in V^\circ} c_u x_u \le 2 \cdot \text{opt}$, where opt is the cost of the optimum integer solution.

6.1 The rounding algorithm

We now describe our rounding algorithm. We define two important global parameters: $L := \lceil \log(2n) \rceil$ and $\gamma := \lfloor \log L \rfloor - 2$. We say an edge (u, v) with $v \in \Lambda_u$ has "hop value" 1 if $x_u < x_v$ and 0 if $x_u = x_v$. For every vertex $u \in V^\circ$, we define ℓ_u to be the sum of hop values over all edges in the path from the root to u in T°. Thus, for every $u \in V^\circ$ and $v \in \Lambda_u$, we have $\ell_v - \ell_u \in \{0, 1\}$, and $\ell_v = \ell_u$ if and only if $x_v = x_u$. By Properties (P1) and (P2), we have that $\ell_v \in [0, L]$ for every $v \in V^\circ$.

Our rounding algorithm is applied on some scaled solution x', which is defined as follows:

$$x'_u = 2^{\min\{\ell_u,\gamma\}} x_u, \text{for every } u \in V^\circ.$$

As we mentioned in the introduction, this change will increase the probability of choosing v conditioned on choosing u by a factor of 2, for some $u \in V^\circ, v \in \Lambda_u$ with $\ell_u < \ell_v \le \gamma$.

We prove one important property for x', which is necessary for us to run the recursive rounding algorithm.

Claim 6.1. *For every $u \in V^\circ$ and $v \in \Lambda_u$, we have $x'_v \le x'_u$.*

Proof. If $x_v = x_u$ then we have (u, v) has hop value 0 and thus $\ell_v = \ell_u$. In this case we have $x'_v = x'_u$ as well. Otherwise, we have $x_v \le x_u/2$ and $h_v = h_u + 1$. So, $\min\{h_v, \gamma\} \le \min\{h_u, \gamma\} + 1$ and therefore $x'_v \le x'_u$. □

Notice that $x'_r = 1$ and every x'_v is an integer power of 2 between 2^{-L} and 1. Our recursive rounding algorithm is run over x'. In the procedure recursive-rounding(u), we add u to our output tree and do the following: for every $v \in \Lambda_u$, with probability x'_v/x'_u independent of all other choices, we call recursive-rounding(v). In the root recursion, we shall call recursive-rounding(r).

Our final algorithm will repeat the recursive procedure M times independently, for a large enough $M = O(\log k)$. Let $T_1, T_2, \cdots, T_M$ be the M trees we obtained from the M repetitions. Our final tree T will be the union of the M trees.

We first analyze the expected cost of T. First focus on the tree T_1. It is easy to see that the probability u is chosen by T_1 is exactly $x'_u \le 2^\gamma x_u = O(L)x_u$. Therefore, the expected cost of T_1 is at most $O(L) \cdot \text{opt}$ by Property (P6). Therefore, the expected cost of the tree T is at most $O(ML) \cdot \text{opt} = O(L \log k) \cdot \text{opt} = O(\log n \log k) \cdot \text{opt}$.

We then analyze the degree constraints on T. Given that u is selected by T_1, the probability that we select a child of v of u is $\frac{x'_v}{x'_u} \le \frac{2x_v}{x_u}$. By Property (P5), we have $\sum_{v\in\Lambda_u} \frac{x'_v}{x'_u} \le \sum_{v\in\Lambda_u} \frac{2x_v}{x_u} \le 4d_u$. Consider all the M trees $T_1, T_2, \cdots, T_M$. Even if we condition on the event that u appears in all the M trees, the degree of u is the summation of many independent random $\{0, 1\}$-variables. The expectation of the summation is at most $4Md_u = O(\log k) \cdot d_u$. Using Chernoff bound, one can show that the probability that the degree of u is more than $O(\log n) \cdot d_u$ is at most $\frac{1}{10n}$, for some large enough $O(\log n)$ factor. Therefore, with probability at least 0.9, every node u in T has degree at most $O(\log n) \cdot d_u$. Therefore, we proved that the degree violation factor of our algorithm is $O(\log n)$, as claimed in Theorem 1.2.

6.2 Analysis of connectivity probability

It remains to show that with high probability, the tree T contains a vertex from every group. This is the goal of this section. Till the end of the section, we focus on the tree T_1 and a fixed group t. For every vertex $u \in V^\circ$, we define $\mathbf{E}_u$ to be the event that u is chosen by T_1. Our goal is to give a lower bound on $\Pr[\bigvee_{o \in O_t} \mathbf{E}_o]$, i.e, the probability that some vertex in O_t is chosen by the tree T_1.

Notice that when two adjacent nodes in T° have the same x' value, then the child is chosen whenever the parent is. Thus, we can w.l.o.g contract any sub-tree of nodes in T° with the same x' value into one single super-vertex, without changing the rounding algorithm. Notice that if two adjacent vertices $u \in V^\circ, v \in \Lambda_u$ have $\ell_u = \ell_v$ then we have $x_u = x_v$ and thus $x'_u = x'_v$. So, we contract every maximal sub-tree of vertices in T° with the same ℓ value. After this operation, for every $u \in V^\circ$, ℓ_u is exactly the level of u in the tree T°. So, for every $u \in V^\circ$ and $v \in \Lambda_v$ we have $\ell_v = \ell_u + 1$. A super-vertex is in O_t if one of its vertices before contracting is in O_t. If an internal super-vertex is in O_t, we can remove all its descendants without changing the analysis in this section. So, again we have that O_t only contains leaves.

For every vertex u, we define

$$z_u = \sum_{o \in O_t \cap \Lambda^*_u} x_o.$$

Notice that $z_u \leq 2x_u$ by Property (P4).

In the following, we shall bound $\Pr\left[\bigvee_{o \in O_t \cap \Lambda^*_u} \mathbf{E}_o \middle| \mathbf{E}_u\right]$ for every $u \in V^\circ$ from bottom to top. This is done in two stages due to the threshold γ we used when we define x' variables. First we consider the case when $\ell_u \geq \gamma$ and then we focus on the case when $\ell_u < \gamma$. The two stages are captured by Lemmas 6.2 and 6.3 respectively.

Lemma 6.2. *For a vertex u with $\ell_u \geq \gamma$, we have* $\Pr\left[\bigvee_{o \in O_t \cap \Lambda^*_u} \mathbf{E}_o \middle| \mathbf{E}_u\right] \geq \frac{1}{2(L+1-\ell_u)} \frac{z_u}{x_u}$.

Similar lemmas have been proved multiple times in many previous results. Since our parameters are slightly different, we provide the complete proof here. There are two different approaches to prove the lemma, one based on bounding the conditional second moment of the random variable for the number of chosen vertices in $O_t \cap \Lambda^*_u$, and the other based on the mathematical induction on ℓ_u, which is the one we use here.

Proof of Lemma 6.2. Suppose u is a leaf. Then $z_u/x_u = 1$ if $u \in O_t$ and $z_u/x_u = 0$ otherwise. So, we have $\Pr\left[\bigvee_{o \in O_t \cap \Lambda^*_u} \mathbf{E}_o \middle| \mathbf{E}_u\right] = \frac{z_u}{x_u}$ and the lemma clearly holds since we have $\ell_u \leq L$.

Then, we prove the lemma by induction on ℓ_u. If $\ell_u = L$ then u must be a leaf and thus the lemma holds. We assume the lemma holds for every u with $\ell_u = \ell + 1$, for some $\ell \in [\gamma, L-1]$. Then we prove the lemma for u with $\ell_u = \ell$. If u is a leaf the lemma holds and thus we assume u

is not a leaf.

$$\begin{aligned}
\Pr\Big[\bigvee_{o\in O_t\cap\Lambda_u^*}\mathbf{E}_o\big|\mathbf{E}_u\Big] &\geq 1-\prod_{v\in\Lambda_u}\left(1-\frac{x'_v}{x'_u}\cdot\frac{1}{2(L-\ell)}\frac{z_v}{x_v}\right) = 1-\prod_{v\in\Lambda_u}\left(1-\frac{x_v}{x_u}\cdot\frac{1}{2(L-\ell)}\cdot\frac{z_v}{x_v}\right)\\
&\geq 1-\prod_{v\in\Lambda_u}\exp\left(-\frac{1}{2(L-\ell)}\cdot\frac{z_v}{x_u}\right) = 1-\exp\left(-\frac{1}{2(L-\ell)}\cdot\frac{z_u}{x_u}\right)\\
&\geq \frac{1}{2(L-\ell)}\cdot\frac{z_u}{x_u}-\frac{1}{2}\left(\frac{1}{2(L-\ell)}\cdot\frac{z_u}{x_u}\right)^2 \geq \frac{1}{2(L-\ell)}\cdot\frac{z_u}{x_u}-\left(\frac{1}{2(L-\ell)}\right)^2\frac{z_u}{x_u}\\
&= \left(\frac{2(L-\ell)-1}{(2(L-\ell))^2}\right)\frac{z_u}{x_u} \geq \frac{1}{2(L+1-\ell)}\cdot\frac{z_u}{x_u}.
\end{aligned}$$

The inequality in the first line used the induction hypothesis: $\frac{x'_v}{x'_u}$ is the probability that we choose v in T_1 conditioned on that we choose u, and $\frac{1}{2(L-\ell)}\frac{z_v}{x_v}$ is the lower bound on the probability that we choose some vertex in $O_t\cap\Lambda_v^*$ conditioned on that v is chosen. The equality in the line used that $x'_u = 2^\gamma x_u$ and $x'_v = 2^\gamma x_v$. The inequality in the second line used that $1-\theta\leq e^{-\theta}$ for every real number θ. The first inequality in the third line used that $e^{-\theta}\leq 1-\theta+\frac{\theta^2}{2}$ for every $\theta\geq 0$. The second inequality in the line used Property (P4), which says $\frac{z_u}{x_u}\leq 2$. The last inequality used that $(2(L-\ell)-1)\cdot 2(L-\ell+1)\geq 4(L-\ell)^2$ since $L-\ell\geq 1$. □

The lemma implies that for every u with $\ell_u\geq\gamma$, we have $\Pr\Big[\bigvee_{o\in O_t\cap\Lambda_u^*}\mathbf{E}_o\big|\mathbf{E}_u\Big]\geq\frac{1}{2L}\cdot\frac{z_u}{x_u}$.

Now we analyze the probability for u with $\ell_u\leq\gamma$. Recall that $\gamma=\lfloor\log L\rfloor-2$ and thus we have $2^\gamma\in(L/8,L/4]$. Let $\alpha_\gamma=\frac{1}{2L}$ and for every $\ell\in[0,\gamma-1]$, define $\alpha_\ell=2\alpha_{\ell+1}-4\alpha_{\ell+1}^2$. It is easy to see that for every $\ell\in[0,\gamma]$, we have $\alpha_\ell\leq\frac{2^{\gamma-\ell}}{2L}$. Then, we have for every $\ell\in[0,\gamma-1]$,

$$\alpha_\ell = 2\alpha_{\ell+1}-4\alpha_{\ell+1}^2 = 2\alpha_{\ell+1}(1-2\alpha_{\ell+1})\geq 2\alpha_{\ell+1}\left(1-2\times\frac{2^{\gamma-\ell-1}}{2L}\right) = 2\alpha_{\ell+1}\left(1-\frac{2^{\gamma-\ell-1}}{L}\right).$$

Therefore, we have

$$\alpha_0\geq 2^\gamma\prod_{\ell=1}^{\gamma}\left(1-\frac{2^{\gamma-\ell-1}}{L}\right)\alpha_\gamma\geq\frac{2^\gamma}{2L}\prod_{\ell=1}^{\gamma}e^{-2^{\gamma-\ell}/L}\geq\frac{2^\gamma}{2L}e^{-2^\gamma/L}=\Omega(1).$$

The second inequality used that $1-\theta\geq e^{-2\theta}$ for every $\theta\in(0,1/2)$. The last equality used that $\gamma=\lfloor\log L\rfloor-2$ and thus $2^\gamma=\Theta(L)$.

With the α values defined, we prove the following lemma via mathematical induction:

Lemma 6.3. *For every vertex $\ell_u=\ell\leq\gamma$, we have* $\Pr\Big[\bigvee_{o\in O_t\cap\Lambda_u^*}\mathbf{E}_o\big|\mathbf{E}_u\Big]\geq\alpha_\ell\frac{z_u}{x_u}$.

Proof. The lemma holds if $\ell=\gamma$ as we mentioned. So, we assume $\ell<\gamma$ and the lemma holds with ℓ replaced by $\ell+1$. If u is a leaf, then we have $\Pr\Big[\bigvee_{o\in O_t\cap\Lambda_u^*}\mathbf{E}_o\big|\mathbf{E}_u\Big]=\frac{z_u}{x_u}$ and the lemma holds.

So again we assume u is not a leaf. Then,

$$\begin{aligned}\Pr\Big[\bigvee_{o\in O_t\cap\Lambda^*_u}\mathbf{E}_o\big|\mathbf{E}_u\Big] &\geq 1-\prod_{v\in\Lambda_u}\left(1-\frac{x'_v}{x'_u}\alpha_{\ell+1}\frac{z_v}{x_v}\right)=1-\prod_{v\in\Lambda_u}\left(1-\frac{2x_v}{x_u}\alpha_{\ell+1}\frac{z_v}{x_v}\right)\\ &\geq 1-\prod_{v\in\Lambda_u}\exp\left(-2\alpha_{\ell+1}\frac{z_v}{x_u}\right)=1-\exp\left(-2\alpha_{\ell+1}\frac{z_u}{x_u}\right)\\ &\geq 2\alpha_{\ell+1}\frac{z_u}{x_u}-\frac{1}{2}\left(2\alpha_{\ell+1}\frac{z_u}{x_u}\right)^2\geq 2\alpha_{\ell+1}\frac{z_u}{x_u}-(2\alpha_{\ell+1})^2\frac{z_u}{x_u}=\alpha_\ell\frac{z_u}{x_u}.\end{aligned}$$

To see the equality in the first line, we notice that $x'_u=2^\ell x_u$ and $x'_v=2^{\ell+1}x_v$ for every $v\in\Lambda_u$. Many other inequalities used the same arguments as in Lemma 6.2. □

Applying the lemma for the root r of T°, we have that $\Pr\left[\bigvee_{o\in O_t}\mathbf{E}_o\right]\geq\alpha_0\cdot\frac{z_r}{x_r}\geq\alpha_0\cdot\frac{1}{2}=\Omega(1)$.

Now we consider all the M trees $T_1,T_2,\cdots,T_M$ together. The probability that O_t is not chosen by any of the M trees is at most $(1-\Omega(1))^M\leq\frac{1}{10k}$, if our $M=O(\log k)$ is big enough. Thus the probability that T, the union of all trees $T_1,T_2,\cdots,T_M$, contains an r-to-O_t path for every t, is at least 0.9.

Acknowledgement X. Guo, S. Li and J. Xian are partially supported by NSF grants CCF-1566356, CCF- 1717134, CCF-1844890. B. Laekhanukit is partially supported by Science and Technology Innovation 2030 –"New Generation of Artificial Intelligence" Major Project No.(2018AAA0100903), NSFC grant 61932002, Program for Innovative Research Team of Shanghai University of Finance and Economics (IRTSHUFE) and the Fundamental Research Funds for the Central Universities and by the 1000-talent award by the Chinese Government. Daniel Vaz has been supported by the Alexander von Humboldt Foundation with funds from the German Federal Ministry of Education and Research (BMBF).

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.

[2] 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.

[3] Sina Dehghani, Soheil Ehsani, Mohammad Taghi Hajiaghayi, Vahid Liaghat, Harald Räcke, and Saeed Seddighin. Online weighted degree-bounded steiner networks via novel online mixed packing/covering. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 42:1–42:14, 2016.

[4] Sina Dehghani, Soheil Ehsani, MohammadTaghi Hajiaghayi, and Vahid Liaghat. Online degree-bounded steiner network design. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 164–175, Philadelphia, PA, USA, 2016. Society for Industrial and Applied Mathematics.

[5] Sina Dehghani, Soheil Ehsani, MohammadTaghi Hajiaghayi, Vahid Liaghat, and Saeed Seddighin. Greedy algorithms for online survivable network design. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 152:1–152:14, 2018.

[6] 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.

[7] 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.

[8] Martin Fürer and Balaji Raghavachari. Approximating the minimum-degree steiner tree to within one of optimal. *J. Algorithms*, 17(3):409–423, 1994.

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

[10] Rohan Ghuge and Viswanath Nagarajan. A quasi-polynomial algorithm for submodular tree orienteering in directed graphs. *CoRR*, abs/1812.01768, 2018.

[11] Michel X. Goemans. Minimum bounded degree spanning trees. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '06, pages 273–282, Washington, DC, USA, 2006. IEEE Computer Society.

[12] Fabrizio Grandoni, Bundit Laekhanukit, and Shi Li. $O(\log^2 k \ / \ \log \log k)$-approximation algorithm for directed steiner tree: a tight quasi-polynomial-time algorithm. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 253–264, 2019.

[13] Mohammad Taghi Hajiaghayi. Open problems on bounded-degree network design from 8-th workshop on flexible network design, amsterdam, 2016. Announcement, 2016.

[14] 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.

[15] Jochen Könemann and R. Ravi. A matter of degree: Improved approximation algorithms for degree-bounded minimum spanning trees. *SIAM J. Comput.*, 31(6):1783–1793, 2002.

[16] Jochen Könemann and R. Ravi. Quasi-polynomial time approximation algorithm for low-degree minimum-cost steiner trees. In *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science, 23rd Conference, Mumbai, India, December 15-17, 2003, Proceedings*, pages 289–301, 2003.

[17] Jochen Könemann and R. Ravi. Primal-dual meets local search: Approximating msts with nonuniform degree bounds. *SIAM J. Comput.*, 34(3):763–773, 2005.

[18] Guy Kortsarz and Zeev Nutov. Bounded degree group steiner tree problems. In *IWOCA'20, to appear*, 2020.

[19] Lap Chi Lau, Joseph Naor, Mohammad R. Salavatipour, and Mohit Singh. Survivable network design with degree or order constraints. *SIAM J. Comput.*, 39(3):1062–1087, 2009.

[20] Lap Chi Lau and Mohit Singh. Additive approximation for bounded degree survivable network design. *SIAM J. Comput.*, 42(6):2217–2242, 2013.

[21] R. Ravi, Madhav V. Marathe, S. S. Ravi, Daniel J. Rosenkrantz, and Harry B. Hunt III. Many birds with one stone: multi-objective approximation algorithms. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 438–447, 1993.

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

[23] Mohit Singh and Lap Chi Lau. Approximating minimum bounded degree spanning trees to within one of optimal. *J. ACM*, 62(1):1:1–1:19, 2015.

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

A Omitted Proofs

Proof of Lemma 2.1. We assume $n \geq 4$; otherwise, if $n = 3$, then we have $2n/3 + 1 = 3$, and root(T) satisfies the condition. Our goal is to find a vertex u with $n/3 < |\Lambda^*(u)| \leq 2n/3 + 1$. Start from $u = \text{root}(T)$ in the tree, and thus, we have $\Lambda^*(u) > 2n/3 + 1$. Let v be the child of u with the biggest $|\Lambda^*(v)|$. So, $|\Lambda^*(v)| \geq (|\Lambda^*(u)| - 1)/2 > n/3$. We then replace u with v. So $|\lambda^*(u)|$ has decreased but the condition $|\Lambda^*(u)| > n/3$ is maintained. Thus, if we repeat the process, we will eventually find a u with $n/3 < |\Lambda^*(u)| \leq 2n/3 + 1$. □