Degrees and Network Design: New Problems and Approximations

Michael Dinitz ✉
Johns Hopkins University, Baltimore, MD, USA

Guy Kortsarz ✉
Rutgers University, Camden, NJ, USA

Shi Li ✉
Nanjing University, Jiangsu, China

─── **Abstract** ───

While much of network design focuses mostly on cost (number or weight of edges), node degrees have also played an important role. They have traditionally either appeared as an objective, to minimize the maximum degree (e.g., the Minimum Degree Spanning Tree problem), or as constraints that might be violated to give bicriteria approximations (e.g., the Minimum Cost Degree Bounded Spanning Tree problem). We extend the study of degrees in network design in two ways. First, we introduce and study a new variant of the Survivable Network Design Problem where in addition to the traditional objective of minimizing the cost of the chosen edges, we add a constraint that the ℓ_p-norm of the node degree vector is bounded by an input parameter. This interpolates between the classical settings of maximum degree (the ℓ_∞-norm) and the number of edges (the ℓ_1-degree), and has natural applications in distributed systems and VLSI design. We give a constant bicriteria approximation in both measures using convex programming. Second, we provide a polylogarithmic bicriteria approximation for the Degree Bounded Group Steiner problem on bounded treewidth graphs, solving an open problem from [17] and [12].

1 Introduction

The overarching theme of network design problems is to find "inexpensive" subgraphs that satisfy some type of connectivity constraints. The notion of "inexpensive" is often either the number of edges (unweighted cost) or the sum of edge costs (weighted cost). However, it has long been recognized that in many applications *vertex degrees* matter as much (or more) than cost. This is particularly true in the context of networking and distributed systems, where the degree of a node often corresponds to the "load" on that node, as well as in VLSI design. So there has been a significant amount of work on handling degrees, either instead of or in addition to cost, which has led to many seminal papers and results. With degrees as an objective, these include the well known local search approach of Fürer and Raghavachari [8] for the Minimum Degree Spanning Tree problem and the Minimum Degree Steiner Tree problem. With degrees as a constraint, these include the iterative rounding [16] approach of Singh and Lau [23] for the Minimum-Cost Bounded-Degree Spanning Tree problem, as well as many extensions (most notably to Survivable Network Design with degree bounds [20], but see [19] for many other examples).

In this paper we extend the study of degrees in network design in two ways. First, we introduce what is (to the best of our knowledge) a new class of problems. Instead of bounding the cost and individual degrees as in [23, 20], our objective is to obtain minimum cost while satisfying a bound on the ℓ_p-norm of the node degree vector. This interpolates between the maximum degree (the ℓ_∞-norm) and the total number of edges or unweighted cost (the ℓ_1-degree). Second, we solve a well known open problem: We give a poly-logarithmic bicriteria approximation for the Group Steiner Tree problem with degree bounds on bounded treewidth graphs.

ℓ_p-**Objective.**   While the maximum degree is often a reasonable objective, as is minimizing the total cost (either with or without degree bounds), there are many natural situations where none of these approaches are fully satisfactory. If we simply ignore the degrees and focus on cost (weighted or unweighted), then we might end up with a solution with highly imbalanced degrees, leading to large load at particular nodes. If we ignore costs and simply optimize the maximum degree, then we might return a solution with far more edges than are needed: if the structure of the graph forces some node to have large degree, then if we simply try to minimize the maximum degree we will not even try to make the degrees of other nodes small. Finally, optimizing under individual degree bounds implicitly assumes that nodes "really have" these degree bounds, i.e., they come from some external constraint. But this is of course not always the case: often we do not have real bounds on individual nodes, but rather a more vague desire to "keep degrees small".

Hence we want some way of making sure that the maximum degree is small, but also encouraging few edges. A natural function that simultaneously accomplishes both of these goals is the ℓ_p-norm of the degree vector, i.e., the function $\left(\sum_{v \in V} (\deg_v)^p\right)^{1/p}$ for $p \geq 1$ (and in particular for $p = 2$), where $\deg_v$ is the degree of v in the output subgraph. When $p = 1$ this is simply (twice) the number of edges (i.e., the unweighted cost), and when $p = \infty$ this is the maximum degree. But for intermediate values of p, it discourages very large degrees (in particular the maximum degree) since $p > 1$ implies that large degrees have a larger effect on the norm than smaller degrees, while still also being effected on a non-trivial way by the smaller degrees. So we can either use the ℓ_p-norm as an objective function, or we can use it as a constraint that is far more flexible than having simple degree constraints at every node.

This intuition, that the ℓ_p-norm takes into account both the maximum and the distribution simultaneously, is one reason why the ℓ_p-norm has been an important objective function for combinatorial problems. For example, the Set Cover problem was studied under the ℓ_p norm of the vector of number of elements assigned to each set [10]. It was also extensively studied in scheduling problems (see for example [1, 2, 18, 15]). To the best of our knowledge, the ℓ_p norm has not been studied in the context of network design, with the notable recent exception of *graph spanners* [6, 5], where the direct applications of spanners to distributed systems led to exactly this motivation. Similarly, MST with ℓ_p norm is very important for VLSI design, since in many such settings we are forced to use spanning trees and hence the number of edges is fixed. So minimizing the ℓ_p norm will likely derive a balanced degree vector which is of key importance for these VLSI application (see, for example, [24, 22]).

Motivated by the above discussion, we introduce and give the first approximation for the Survivable Network design problem with low cost under a bound on the ℓ_p norm of the degree vector.

Group Steiner Tree with Degree Bounds.   In addition to the study of ℓ_p-norm problems, we also make significant progress on a known open problem: approximating Group Steiner Tree with degree bounds on bounded treewidth graphs. The Group Steiner Tree problem

(without degree bounds) is a classical optimization problem [9] which has played a central role in network design. In this problem there is a designated root node r, and a collection of (not necessarily disjoint) *groups* of vertices. The goal is to find a subtree which connects at least one vertex from each group to r, while minimizing the total cost of all edges in the subtree. The Degree Bounded Group Steiner problem was first raised by Hajiaghayi in [13] (in the 8th Workshop on Flexible Network Design), motivated by the online version of the problem and applications to VLSI design. In particular, while low cost is highly desirable, this cost is payed only once, while later the VLSI circuit is applied (evaluated) constantly. Low degrees imply that the computation of the value of the circuit can be done faster. See a discussion of why low degrees are important for Group Steiner in [17].

Unfortunately, despite significant recent interest in this problem [12, 17], progress has been elusive. In particular, polylogarithmic bicriteria approximations were not even known for simple classes such as *series-parallel* graphs, i.e., for graphs with treewidth 2. We go far beyond series-parallel graphs, and give results for bounded treewidth graphs.

1.1 Our Results and Techniques

We begin in Section 2 with a study of the ℓ_p-Survivable Network Design problem. We are given the input graph $G = (V, E)$, with edge costs $c \in \mathbb{R}^E_{\geq 0}$. There is a connection requirement vector $r \in \mathbb{Z}^{\binom{V}{2}}_{\geq 0}$, a number $p \geq 1$ and a bound A on the ℓ_p norm of the degree vector of the output graph. The goal of the problem is to find the minimum-cost subgraph H of G satisfying the following:

- **(connection requirements)** for every $u, v \in V$ with $u \neq v$, there are at least $r_{u,v}$ edge disjoint paths between u and v in H, and
- **(degree constraint)** $\left(\sum_{v \in V} d^p_H(v)\right)^{1/p} \leq A$, where $d_H(v)$ is the degree of v in H.

We assume the input instance is feasible; that is, there is a valid sub-graph H satisfying both requirements. Let opt be the minimum cost of a valid subgraph H. The main theorem we prove for the problem is the following:

▶ **Theorem 1.** *There is a (randomized) algorithm which, given an instance of ℓ_p-SURVIVABLE NETWORK DESIGN, outputs a subgraph H satisfying the connection requirements and which has the following properties.*
- *The expected cost of H is at most $2 \cdot$ opt.*
- *The expectation of the ℓ_p-norm of the degree vector is at most $2^{1/p} 5^{1-1/p} \cdot A$.*

For the special case of ℓ_p-Spanning Tree problem, where $r_{uv} = 1$ for all $u, v \in V$, we improve the expected cost to at most opt (rather than $2 \cdot$ opt) and the expectation of the ℓ_p-norm of the degree vector to at most $2^{1-1/p} \cdot A$.

Our main approach is to leverage the fact that the ℓ_p-norm is *convex*. This allows us to write a convex relaxation for the problem, which can then be solved efficiently using standard convex programming techniques. We then round this solution using an iterative rounding approach. Making this work requires overcoming a number of issues, possibly the trickiest of which is handling fractional degrees that are less than 1. Note that a fractional solution could have many nodes with very small fractional degree (e.g., $1/n$). Due to the structure of the ℓ_p-norm, such small values contribute far less to the ℓ_p-norm than they "should" (in an integral solution). To get around this, we actually *change* the ℓ_p-constraint in a way that acts differently for values less than 1, while still maintaining convexity. With this change in place, we can solve the relaxation, interpret the fractional degrees "as if" they are true degree bounds, and then round using existing results on iterative rounding for degree-bounded network design.

We then move to our second problem, Group Steiner Tree with Degree Bounds on bounded treewidth graphs. In the problem, we are given a graph $G = (V, E)$ with treewidth tw, a cost vector $c \in \mathbb{R}_{\geq 0}^E$, a root r, and k sets $S_1, S_2, \cdots, S_k$. We are additionally given a degree bound $\mathrm{db}_v \in \mathbb{Z}_{>0}$ for every $v \in V$. The goal of the problem is to choose a minimum-cost subgraph H of G such that for every $t \in k$, H contains a path from r to some vertex in S_t, and $d_H(v) \leq \mathrm{db}_v$ for every $v \in V$. By minimality, the optimum H is always a tree. We solve an open problem from [12] and [17] by giving a polylogarthmic bicriteria algorithm as long as the treewidth is bounded. In particular, we prove the following theorem.

▶ **Theorem 2.** *There is an $n^{O(\mathrm{tw}\log\mathrm{tw})}$-time randomized algorithm for the Group Steiner Tree with Degree Bounds problem on bounded treewidth graphs which has $O(\log^2 n)$ approximation ratio and $O(\log^2 n)$-degree violation.*

In order to achieve this result, we reduce the problem to a "tree labeling" problem in Section 3, which has been used in prior results [11, 12, 21]. There is a rooted full binary tree, and we need to give a label ℓ_u for each node u in the tree from a subset L_u of potential labels. For every internal node u with two children v and v' there are some consistency constraints on the labels, which say that the triple $(\ell_u, \ell_v, \ell_{v'})$ must be from some given subset $\Gamma_u \in L_u \times L_v \times L_{v'}$. Then we have some covering constraints, each specified by a set S of labels: the constraint requires that at least one node has its label in S. Finally, we have many cost constraints. For each such constraint, a label is given a cost, and we require that the total cost of all labels used is at most 1. For this problem we give a randomized algorithm that outputs a labeling that satisfies all consistency constraints, and approximately satisfies the covering and cost constraints with reasonable probability, assuming the given instance is feasible. It runs in polynomial time when the depth of the tree is $O(\log n)$ and each L_u has $O(1)$-size. The main techniques of the algorithm are adaptations of the LP-rounding algorithm in [12] for their degree-bounded network design problem. We introduce the tree labeling problem as a host for these techniques, and adapt them for the problem.

Due to space limitations, we leave the reduction of Group Steiner Tree with Degree Bounds on bounded treewidth graphs to this tree labeling problem to Appendix B, and give a high-level idea here. Let tw be the treewidth of the graph; it is known from [3] that we can assume the decomposition tree of G is an $O(\log n)$-depth binary tree, with bag size $O(\mathrm{tw})$. This decomposition tree will be the tree in the tree-labeling instance. For each bag in the tree, a label will contain the set of edges we take from the bag, and some connectivity information on the vertices in the bag. We define the consistency constraints so that if they are satisfied, then the connectivity information is correct. A group being connected can be captured by a covering constraint in the tree labeling instance, and the edge cost constraint and degree constraints can be formulated as cost constraints in the instance. Using the algorithm for the tree labeling instance, we obtain a tree with small cost that satisfies degree bounds approximately, and connects a group with reasonable probability. The final output then is obtained by running the procedure many times and taking the union.

1.2 Other Related Work

For the survivable network design problem without any degree constraints, the classic result of Jain [16] gives a 2-approximation algorithm using the iterative rounding method. In [9] an $O(\log^2 n)$ approximation is given for the Group Steiner problem on tree inputs, and an $O(\log^3 n)$ for the Group Steiner problem (without degree constraints) for bounded treewidth graphs. The approximation for trees is almost the best possible, unless NP problems can be solved in quasi-polynomial time [14]. [12] gave a bicriteria approximation for the Group

Steiner Tree Problem with degree bounds on tree inputs, with approximation ratio $O(\log^2 n)$ and degree violation $O(\log n)$). Both bounds are nearly optimal [14, 7]. In [4] the authors gave an $O(\log^2 n)$-approximation ratio for Group Steiner problem on bounded treewidth graphs (without degree bounds). In [17] an $O(\log^2 n)$ approximation is given for the Group Steiner problem with minimum maximal degree, but without costs.

Notations. Given a graph H and a vertex v in H, we shall use $\delta_H(v)$ to denote the set of edges in H incident to v, and $d_H(v) = |\delta_H(v)|$ to denote its degree. Given a rooted tree T and a vertex v in T, we use $\Lambda_T(v)$ to denote the set of children of v in T, and $\Lambda_T^*(v)$ to denote the set of descendants of v in T (including v itself). When H and T are clear from the context, we shall omit them in the subscript. For example, this happens when $H = G$ is the input graph. For a real vector z over some domain, and a subset S of elements in the domain, we define $z(S) := \sum_{i \in S} z_i$ to denote the sum of z values of elements in S.

2 ℓ_p-Survivable Network Design

In this section, we give our iterative rounding algorithm for ℓ_p-survivable network design problem. Recall that we are given a graph $G = (V, E)$ with cost vector $c \in \mathbb{R}^E_{\geq 0}$, a connection requirement vector $r \in \mathbb{Z}^{\binom{V}{2}}_{\geq 0}$, and a bound A on the ℓ_p norm of the degree vector.

▶ **Definition 3.** *We say a polytope $\mathcal{P} \in [0,1]^E$ is good if it is upward-closed [1] and the following holds: For every vector $x \in \{0,1\}^E$, we have that $x \in \mathcal{P}$ if and only if the graph $(V, \{e \in E : x_e = 1\})$ satisfies the connection requirements.*

Notice that the above definition does not capture the degree constraints. This is done using the following definition. For a real vector $B \in [1, \infty]^V$, we define $\mathcal{Q}_B := \{x \in [0,1]^E : \forall v \in V, x(\delta(v)) \leq B_v\}$ to be the set of all vectors satisfying the degree bounds defined by B.

▶ **Definition 4.** *Let $\alpha \geq 1$ and $\beta \geq 0$ be two real numbers and $\mathcal{P}$ be a good polytope. We say $\mathcal{P}$ is (α, β)-integral if for every $B \in [1, \infty]^V$, every non-integral extreme point x of $\mathcal{P} \cap \mathcal{Q}_B$ satisfies at least one of the following two properties:*
(4a) *there exists an edge $e \in E$ with $1/\alpha \leq x_e < 1$,*
(4b) *there is a vertex $v \in V$ such that $x(\delta(v)) = B_v$ and $|\{e \in \delta(v) : x_e > 0\}| \leq B_v + \beta$.*

It is well known that for Survivable Network Design there is a $(2,3)$-integral polytope $\mathcal{P}$ [20]. For the special case of spanning tree, i.e, $r \equiv 1$, there is a $(1,1)$-integral polytope [23].

We will use these polytopes in our algorithm, and will show that that their existence implies good approximation algorithms. More formally, we prove the following theorem.

▶ **Theorem 5.** *Assuming the existence of an (α, β)-integral polytope, there is a randomized algorithm which outputs a subgraph H of G satisfying the connection requirements. The expected cost of H is at most $\alpha \cdot \mathrm{opt}$ and the expectation of the p-norm of degree vector is at most $\alpha^{1/p}(\alpha + \beta)^{1-1/p}A$; recall that opt is the value of the instance.*

Note that this theorem, together with the existence of a $(2,3)$-integral polytope for the general case and a $(1,1)$-integral polytope for the spanning tree case, imply Theorem 1. So we focus on proving Theorem 5.

---

[1] This means for every $x \in \mathcal{P}$ and $x' \in [0,1]^E$ with $x' \geq x$, we have $x' \in \mathcal{P}$.
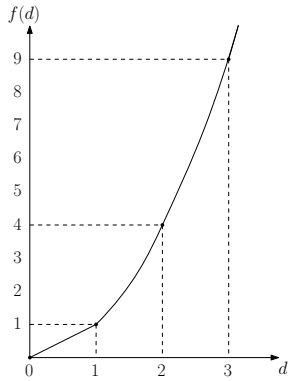
(a) The function f for $p = 2$.

1: Solve LP(1) to obtain a solution x
2: let $B_v \leftarrow \max\{x(\delta(v)), 1\}$ for every $v \in V$
3: **while** true **do**
4:     randomly choose an extreme point x' of $\mathcal{P} \cap \mathcal{Q}_B$
    such that $\mathbb{E}[x'] = x$
5:     $x \leftarrow x'$
6:     **if** x is integral **then return** x
7:     **if** case a happens for some $e = (u, v) \in E$ **then**
8:         $x_e \leftarrow 1, B_v \leftarrow x(\delta(v)), B_u \leftarrow x(\delta(u))$
9:     **else**   ▷ case b happens for some v
10:        $B_v \leftarrow \infty$

(b) Iterative Rounding Algorithm for Network Design.

■ **Figure 1** The function f and the iterative rounding algorithm.

2.1 The Convex Program

Define a function $f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ as follows: $f(x) = \begin{cases} x & \text{if } x \in [0, 1] \\ x^p & \text{if } x > 1 \end{cases}$. Figure (1a) shows this function for $p = 2$. This is a convex function for $p \geq 1$.

Let $\mathcal{P}$ be an (α, β)-integral polytope. The following is our convex programming relaxation for the problem:

$$\min \sum_{e \in E} c_e x_e \qquad \text{s.t.} \qquad x \in \mathcal{P}, \qquad \sum_{v \in V} f(x(\delta(v))) \leq A^p. \tag{1}$$

Recall that using our notation, $x(\delta(v))$ is the sum of x values of edges incident to v in G. (1) is a convex program and can be solved efficiently. Since the indicator vector of the optimum subgraph H satisfies all the constraints, the value of the convex program is at most opt.

We note that if we instead used the function $f(x) = x^p$ (i.e., without handling the $0 \leq x \leq 1$ case separately), we would still have a convex relaxation of our problem. However, it is not hard to show that this relaxation has an extremely large integrality gap (even if we are allowed to violate the ℓ_p-norm constrain by a polylogarithmic factor). Treating $0 \leq x \leq 1$ differently from $x > 1$ is one of the key ideas in our approximation algorithm.

2.2 The Iterative Rounding Algorithm

Our iterative rounding algorithm is described in Figure (1b). In Step 1, we solve the convex relaxation (1) to obtain an extreme solution x, which can be done in polynomial time using standard techniques. Then in Step 2 we define $B_v = \max\{x(\delta(v)), 1\}$ for every v to be the upper bound on the degree of v. So, before Loop 3, we have $x \in \mathcal{P} \cap \mathcal{Q}_B$. We shall maintain this property before and after each iteration of the loop.

In each iteration of Loop 3, we randomly choose a vertex point x' of $\mathcal{P} \cap \mathcal{Q}_B$ such that $\mathbb{E}[x'] = x$ (Step 4) and then update x to be the x' (Step 5). This is possible since at the beginning of the iteration we have $x \in \mathcal{P} \cap \mathcal{Q}_B$. If x is integral, we then return x in Step 6. If we did not return, by that $\mathcal{P}$ is (α, β)-integral, either a or b happens. In the former case, we update x_e to 1, and change B_v and B_u for the two end vertices u, v of e so that we still have $B_{v'} = \max\{x(\delta(v')), 1\}$ for every $v' \in V$ (Step 8). In the latter case, we change B_v to ∞ so that there will be no degree constraint for v from now on. Notice that in either case, we maintain the invariant that $x \in \mathcal{P} \cap \mathcal{Q}_B$ as $\mathcal{P}$ is upward-closed.

Notice that once x_e becomes 0 or 1 in some iteration, it will remain unchanged. This holds since for $\mathbb{E}[x'_e] = x_e \in \{0,1\}$ to hold, we must always have $x'_e = x_e$. When the algorithm terminates, it returns an integral x which satisfies the connectivity requirements. This holds since we have $x \in \mathcal{P}$ and $\mathcal{P}$ is good. The algorithm will terminate in $O(|E|)$ iterations since in every iteration, we either fixed the value of some x_e to 1, or changed some B_v from a finite number to ∞.

2.3 Analysis of the Algorithm

We now begin to analyze the algorithm. As discussed, the algorithm will terminate with a subgraph which satisfies the connectivity requirements. To prove Theorem 5, we need to analyze the total cost and the ℓ_p-norm of the degrees.

In Step 8, we say that we *round* the edge e. In Step 10, we say we *relax* the vertex v. At any time of the algorithm, we define a vector $\bar{x} \in [0,1]^E$ as follows. If e has not been rounded yet, then let $\bar{x}_e = x_e$. Otherwise, let $\bar{x}_e$ be the value of x_e right before Step 8 in which we round e. Thus, from the moment, $\bar{x}_e$ remains unchanged.

Let T be the number of iterations we run Loop 3; notice that this is a random variable. For every integer $t \in [0, T]$, we let $x^t, \bar{x}^t, B^t$ to be the values of $x, \bar{x}, B$ at the end of the t-th iteration of the Loop 3. So x^T is the output of the algorithm.

▶ **Observation 6.** *The following statements are true.*

(6a) *During Loop 3, we always have $\bar{x}_e \leq x_e \leq \alpha \bar{x}_e$ for every $e \in E$.*

(6b) *Assume $x^0(\delta(v)) < 1$ for a vertex $v \in V$. Then at the first moment when $x(\delta(v)) \geq 1$ holds, we have $\bar{x}(\delta(v)) \leq 1$.*

(6c) *$\bar{x}(\delta(v))$ does not change from the first moment $x(\delta(v)) \geq 1$ holds, until the moment v is relaxed, or the end of the algorithm if this does not happen.*

Proof. $\bar{x}_e \leq x_e$ by the definition of $\bar{x}_e$. Moreover, $x_e \leq \alpha \bar{x}_e$ as if $x_e > \bar{x}_e$, then $x_e = 1$ and $x_e \geq \frac{1}{\alpha}$. So a holds.

To prove b, we consider two scenarios. In the first scenario, the moment is after Step 5 in some iteration. In this scenario, $\bar{x}(\delta(v)) = x(\delta(v)) = 1$ since $B_v = 1$ at the moment. In the second scenario, the moment is after we round some edge $e \in \delta(v)$ in Step 8. In this case $\bar{x}(\delta(v))$ is the same as $x(\delta(v))$ before the step, which is strictly less than 1.

c holds since we maintained $B_v = x(\delta(v))$ from the moment $x(\delta(v))$ becomes at least 1. If $\bar{x}_e \neq x_e$ at some time, it must be the case that $x_e = 1$. In this case, both x_e and $\bar{x}_e$ will not change in the future. ◀

We can now analyze the expected cost of the algorithm. First, though, we will need a structural result.

▶ **Lemma 7.** *For every edge $e \in E$, the sequence $\bar{x}_e^0, \bar{x}_e^1, \cdots, \bar{x}_e^T$ is a martingale.*

Proof. Focus on an iteration $t \geq 1$ and edge $e \in E$, and we fix the sequence $\bar{x}_e^0, \bar{x}_e^1, \cdots, \bar{x}_e^{t-1}$. For simplicity we use $\mathbb{E}'[\cdot]$ to denote $\mathbb{E}[\cdot | \bar{x}_e^0, \bar{x}_e^1, \cdots, \bar{x}_e^{t-1}]$. We need to prove $\mathbb{E}'[\bar{x}_e^t] = \bar{x}_e^{t-1}$.

If we rounded e in iteration t or before, then $\bar{x}_e^t = \bar{x}_e^{t-1}$ happens with probability 1. So, we can assume that e has not been rounded by the end of iteration t. In this case, $\bar{x}_e^{t-1} = x_e^{t-1}$.

So, in iteration t, either a happens for some $e' \neq e$, or b happens. In either case, we have $\mathbb{E}'[\bar{x}_e^t] = \mathbb{E}'[x_e^t] = x_e^{t-1} = \bar{x}_e^{t-1}$ by the way we define the distribution for x' in Step 4. Therefore, $\bar{x}_e^0, \bar{x}_e^1, \cdots, \bar{x}_e^T$ is a martingale. ◀

▶ **Corollary 8.** $\mathbb{E}\left[\sum_{e \in E} c_e x_e^T\right] \leq \alpha \sum_{e \in E} c_e x_e^0$.

Proof.

$$\mathbb{E}\left[\sum_{e \in E} c_e x_e^T\right] \leq \alpha \, \mathbb{E}\left[\sum_{e \in E} c_e \bar{x}_e^T\right] = \alpha \sum_{e \in E} c_e \bar{x}_e^0 = \alpha \sum_{e \in E} c_e x_e^0.$$

The inequality is by a and the first equality used Lemma 7. ◀

Now that we understand the expected cost, it only remains to analyze the degree constraint. From now on we fix a vertex $v \in V$. We upper bound $x^T(\delta(v))$, which will in turn give an upper bound on $\mathbb{E}\left[(x^T(\delta(v)))^p\right]$. The main lemma we prove is

▶ **Lemma 9.** *For every $v \in V$, we have $\mathbb{E}\left[(x^T(\delta(v)))^p\right] \leq \alpha(\alpha + \beta)^{p-1} \cdot f(x^0(\delta(v)))$.*

Proof. We first consider the case $x^0(\delta(v)) \geq 1$. Let t be the iteration in which v is relaxed, or let $t = T$ if v is not relaxed during the algorithm. By Property c, $\bar{x}(\delta(v))$ does not change until the end of iteration t. Then, we have $x^T(\delta(v)) \leq x^t(\delta(v)) + \beta \leq \alpha\bar{x}^t(\delta(v)) + \beta = \alpha\bar{x}^0(\delta(v)) + \beta = \alpha x^0(\delta(v)) + \beta$. Notice that this happens with probability 1.

Notice that $\mathbb{E}[x^T(\delta(v))] \leq \alpha \, \mathbb{E}[\bar{x}^T(\delta(v))] = \alpha\bar{x}^0(\delta(v)) = \alpha x^0(\delta(v))$ by Lemma 7. We have:

$$\mathbb{E}\left[(x^T(\delta(v)))^p\right] \leq \frac{\alpha x^0(\delta(v))}{\alpha x^0(\delta(v)) + \beta}(\alpha x^0(\delta(v)) + \beta)^p = \alpha x^0(\delta(v))(\alpha x^0(\delta(v)) + \beta)^{p-1},$$

$$\frac{\mathbb{E}\left[\left(x^T(\delta(v))\right)^p\right]}{f(x^0(\delta(v)))} \leq \frac{\alpha x^0(\delta(v))(\alpha x^0(\delta(v)) + \beta)^{p-1}}{(x^0(\delta(v)))^p} = \alpha\left(\alpha + \frac{\beta}{x^0(\delta(v))}\right)^{p-1} \leq \alpha(\alpha + \beta)^{p-1}.$$

Now we consider the second case: $x^0(\delta(v)) < 1$. We prove that with probability 1, we have $x^T(\delta(v)) \leq \alpha + \beta$. Assume $x(\delta(v)) \geq 1$ happens at some time of the algorithm. By b, at the first moment when $x(\delta(v)) \geq 1$, we have $\bar{x}(\delta(v)) \leq 1$. By c, from the moment until the moment v becomes relaxed (or until the end of the algorithm if v is never relaxed), $\bar{x}(\delta(v))$ does not change. Therefore, immediately after v becomes relaxed, we have $\bar{x}(\delta(v)) \leq 1$. Thus $x^T(\delta(v))$ is at most the value of $x(\delta(v)) + \beta$ at this moment, which is at most $\alpha\bar{x}(\delta(v)) + \beta \leq \alpha + \beta$. If $x(\delta(v)) \geq 1$ never happens, then $x^T(\delta(v)) < 1 \leq \alpha + \beta$.

As in the first case, we have $\mathbb{E}[x^T(\delta(v))] \leq \alpha x^0(\delta(v))$. So

$$\mathbb{E}\left[(x^T(\delta(v)))^p\right] \leq \frac{\alpha x^0(\delta(v))}{\alpha + \beta}(\alpha + \beta)^p = \alpha x^0(\delta(v))(\alpha + \beta)^{p-1},$$

$$\frac{\mathbb{E}\left[(x^T(\delta(v)))^p\right]}{f(x^0(\delta(v)))} \leq \frac{\alpha x^0(\delta(v))(\alpha + \beta)^{p-1}}{x^0(\delta(v))} = \alpha(\alpha + \beta)^{p-1}.$$

So, we always have $\mathbb{E}\left[(x^T(\delta(v)))^p\right] \leq \alpha(\alpha + \beta)^{p-1}f(x^0(\delta(v)))$. This implies $\mathbb{E}[\sum_v (x^T(\delta(v)))^p] \leq \alpha(\alpha + \beta)^{p-1}A^p$. ◀

Corollary 8 and Lemma 9 imply Theorem 5, which in turn implies Theorem 1.

3   A Tree Labeling Problem

In this section, we describe the tree labeling problem to which we reduce the Group Steiner Tree problem with degree bounds on bounded-treewidth graphs; the problem has served as a building block in many prior results [11, 12, 21]. We are given a full binary tree $\mathbf{T} = (\mathbf{V}, \mathbf{E})$ rooted at $\mathbf{r} \in \mathbf{V}$.[2] For every vertex $u \in \mathbf{V}$, we are given a finite set L_u of *labels* for u; we

---

[2] It is not important to require the binary tree to be full; our algorithm works when some internal nodes have only one child. Assuming every internal node have 2 children is only for notational convenience.

assume L_u's are disjoint and let $L := \bigcup_{u \in \mathbf{V}} L_u$. The output is a labeling $\vec{\ell} = (\ell_u \in L_u)_{u \in \mathbf{V}}$ of the vertices $\mathbf{V}$, that satisfies the constraints described below.

- **(consistency constraints)** For every internal node u of $\mathbf{T}$ with two children v and v', we are given a set $\Gamma_u \subseteq L_u \times L_v \times L_{v'}$. A valid labeling $\vec{\ell}$ must satisfy $(\ell_u, \ell_v, \ell_{v'}) \in \Gamma_u$.
- **(covering constraints)** We are given k subsets $S_1, S_2, \cdots, S_k \subseteq L$. A valid labeling $\vec{\ell}$ needs to satisfy that for every $t \in [k]$, $\ell(\mathbf{V}) \cap S_t \neq \emptyset$, where $\ell(\mathbf{V})$ is defined as $\{\ell_u : u \in \mathbf{V}\}$. In words, $\ell(\mathbf{V})$ needs to intersect every S_t.
- **(cost constraints)** We are given $m \geq 0$ linear constraints defined by the costs $(c_\ell^i \in [0, 1])_{i \in [m], \ell \in L}$. For every $i \in [m]$, a valid labeling $\vec{\ell}$ needs to satisfy $\sum_{u \in \mathbf{V}} c_{\ell_u}^i \leq 1$. In words, there are m types of resource, and we have 1 unit of each type. Setting the label of u to ℓ will use c_ℓ^i units of type i-resource.

We say a labeling $\vec{\ell} = (\ell_u \in L_u)_{u \in \mathbf{V}}$ is *consistent* if it satisfies the consistency constraints. Given a consistent labeling $\vec{\ell}$, we say it covers group S_t if $\ell(\mathbf{V}) \cap S_t \neq \emptyset$. We define its type-$i$ cost to be $\mathrm{cost}^i(\vec{\ell}) := \sum_{u \in \mathbf{V}} c_{\ell_u}^i$. So a valid labeling $\vec{\ell}$ for the instance is a consistent one that covers all groups, and has $\mathrm{cost}^i(\vec{\ell}) \leq 1$ for every $i \in [m]$.

Given a label tree instance, we let $n = |\mathbf{V}|$, D be the height of $\mathbf{T}$ (the maximum number of edges in a root-to-leaf path in $\mathbf{T}$) and $\Delta = \max_{u \in \mathbf{V}} |L_u|$ be the maximum size of any L_u. The main theorem we prove is the following:

▶ **Theorem 10.** *Assume we are given a feasible label tree instance* $(\mathbf{T} = (\mathbf{V}, \mathbf{E}), \mathbf{r}, (L_u)_u,$ $(\Gamma_u)_u, (S_t)_{t \in [k]}, A \in [0, 1]^{m \times L})$, *i.e., there is a valid labeling. There is a randomized algorithm that in time* $\mathrm{poly}(n) \cdot \Delta^{O(D)}$ *outputs a consistent labeling* $\vec{\ell}$ *such that the following holds.*

(10a) *For every* $t \in [k]$, *we have* $\Pr[\vec{\ell} \text{ covers group } S_t] \geq \frac{1}{D}$.

(10b) *For every* $i \in [m]$, *we have* $\mathbb{E}\left[\exp\left(\ln(1 + \frac{1}{2D}) \cdot \mathrm{cost}^i(\vec{\ell})\right)\right] \leq 1 + \frac{1}{D}$.

Property b gives a tail concentration bound on $\mathrm{cost}^i(\vec{\ell})$. The remaining part of this section is dedicated to the proof of Theorem 10.

3.1 Construction of a super-tree T°

In this section, we construct a rooted tree $T^\circ = (V^\circ, E^\circ)$ of size $O(n)\Delta^{O(D)}$ such that a consistent labeling of $\mathbf{T}$ corresponds to what we call *a consistent sub-tree*. So we can reduce the problem to finding the latter object. The root of T° is r. Each internal node of T° is either a *selector node*, or a *copier node*; their meanings will be clear soon. Each node $p \in V^\circ$ is *associated with* a node u in T. Each non-root selector node or leaf node is *associated with* a label $\ell \in L_u$. We shall use p and q and their variants to denote nodes in T°, and u and v and their variants to denote nodes in $\mathbf{T}$.

The algorithm for constructing T° is described in Algorithm 1, which calls the procedure construct-tree described in Algorithm 2. See Figure 2 for the illustration of the construction of T° from $\mathbf{T}$. For a node $p \in V^\circ$, we use $\Lambda(p)$ denotes the set of children of p in T°, and $\Lambda^*(p)$ denotes the set of descendants of p in T°, including p itself.

■ **Algorithm 1** Main algorithm for the construction of T°.

1: create a node r associated with $\mathbf{r}$ as the root of T°, and let r be a *selector node*
2: **for** every $\ell \in L_\mathbf{r}$ **do**:
3:     create a child p of r, associated with node $\mathbf{r}$ and label ℓ
4:     call construct-tree$(p, \mathbf{r}, \ell)$
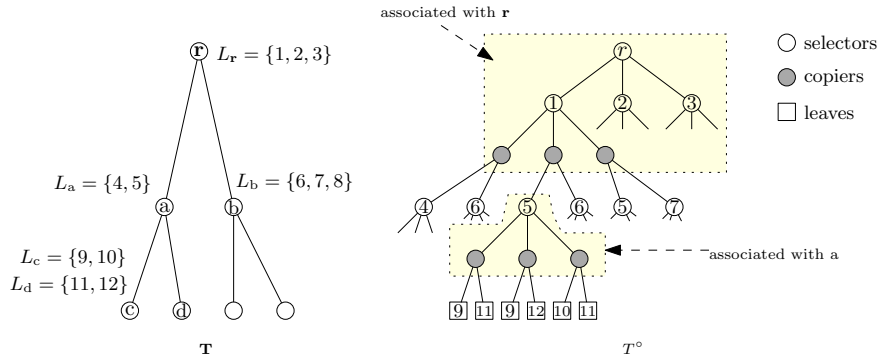
Figure 2 An example for the construction of T°. The tree on the left side is $\mathbf{T}$, and the tree on the right side is $\mathbf{T}^\circ$. The labels of the nodes in $\mathbf{T}$ are shown besides them. In T°, selectors, copiers and leaves are denoted as empty circles, solid circles and empty squares respectively. The nodes in the two yellow polygons are associated with $\mathbf{r}$ and a respectively. The numbers in the circles and squares indicate the labels associated with the nodes. In the example, the triples in $\Gamma_\mathbf{r}$ with the first coordinate being 1 are $(1, 4, 6), (1, 5, 6)$ and $(1, 5, 7)$. The triples in $\Gamma_\mathbf{a}$ with the first coordinate being 5 are $(5, 9, 11), (5, 9, 12)$ and $(5, 10, 11)$.

Algorithm 2 construct-tree(p, u, ℓ). $\triangleright\ p \in V^\circ, u \in \mathbf{V}, \ell \in L_u$

1: **if** u has no children **then return** $\triangleright\ p$ is a leaf node.
2: let p be a *selector node*, let v and v' be the two children of u in $\mathbf{T}$
3: **for** every $\ell' \in L_v, \ell'' \in L_{v'}$ such that $(\ell, \ell', \ell'') \in \Gamma_u$ **do**
4:     create a child p' of p, associated with u, let p' be a *copier node*,
5:     create two children q and q' of p', associate q with node v and label ℓ', associate q' with node v' and label ℓ''
6:     call construct-tree(q, v, ℓ') and construct-tree(q', v', ℓ'')

Now we can define consistent sub-trees of T°:

▶ **Definition 11** (Consistent sub-trees). *Given a sub-tree T of T° that contains r, we say T is* consistent *if the following conditions hold.*

- *Every selector node p in T has exactly one child in T.*
- *If p is a copier node in T, then both of its children in T° are in T.*

The definition explains the names "selector" and "copier": a selector node p in T needs to select one of its children in T° and add it to T, and the children of a copier node p will follow the node p to enter T.

It is easy to see a one-to-one correspondence between consistent labelings $\vec{\ell} = (\ell_u \in L_u)_{u \in \mathbf{V}}$ of $\mathbf{T}$, and consistent sub-trees T of T°. Given the consistent labeling $\vec{\ell}$, the correspondent sub-tree T of T° can be constructed as follows. First, we add r and its child p associated with label $\ell_\mathbf{r}$ to T. Then we grow the tree from p using a recursive procedure. Assume p is associated with node u in $\mathbf{T}$ and label $\ell \in L_u$. If u is a leaf, we stop the procedure. Otherwise let v and v' be the two children of u, then we add the copier child p' of p that corresponds to the tuple $(\ell_\mathbf{r}, \ell_v, \ell_{v'})$ to T. We also add its two children q and q' to T. Then we run the procedure recursively over q and q'. Conversely, given a consistent sub-tree T of T°, we can recover a consistent labeling $\vec{\ell}$ of $\mathbf{T}$.

For convenience, we extend the costs $(c_\ell^i)_{i\in[m],\ell\in L}$ to vertices in V°: For every non-root selector node or leaf node $p \in V^\circ$ associated with a label ℓ, we define $c_p^i = c_\ell^i$ for every $i \in [m]$. For the root or a copier node p, we define $c_p^i = 0$. For a consistent sub-tree $T = (V, E)$ of T°, and $i \in [m]$, we define its type-i cost to be $\mathrm{cost}^i(T) = \sum_{p\in V} c_p^i$. This will be the same as $\mathrm{cost}^i(\vec{\ell})$, for the labeling $\vec{\ell}$ correspondent to T.

We also extend the groups $S_1, S_2, \cdots, S_k$ to node sets in T°: for every $t \in [k]$, S'_t contains the set of nodes $p \in V^\circ$ whose associated label is in S_t. Then, a consistent labeling $\vec{\ell}$ covers a group S_t if and only if the correspondent sub-tree $T = (V, E)$ covers S'_t, namely, $V \cap S'_t \neq \emptyset$.

Therefore, we are guaranteed that there is a consistent sub-tree T^* of T° that covers all groups $S'_1, S'_2, \cdots, S'_k$, and has $\mathrm{cost}^i(T^*) \leq 1$ for every $i \in [m]$. Our goal is to output a random consistent sub-tree T satisfying the conditions correspondent to a and b. This is done using an LP-based algorithm.

3.2 The LP relaxation for finding $T = (V, E)$

Now we describe the LP relaxation that we use to find $T = (V, E)$. For every vertex $p \in V^\circ$, we use x_p to indicate if p is in T, i.e., $p \in V$. For every $t \in [k]$ and $q \in S'_t$, we use y_q^t to indicate if q is the node in T we choose to cover S'_t. There might be multiple nodes in $V \cap S'_t$, and in this case, we only choose one node in the set to cover S'_t; the choice can be arbitrary. The LP is as follows.

$$x_r = 1 \qquad (2)$$

$$\sum_{q\in\Lambda(p)} x_q = x_p, \forall \text{ selector } p \in V^\circ \qquad (3)$$

$$x_q = x_p, \forall \text{ copier } p \in V^\circ, q \in \Lambda(p) \qquad (4)$$

$$x_p \geq 0, \ \forall p \in V^\circ \qquad (5)$$

$$\sum_{q\in\Lambda^*(p)\cap S'_t} y_q^t \leq x_p, \forall p \in V^\circ, t \in [k] \qquad (6)$$

$$\sum_{q\in S'_t} y_q^t = 1, \ \forall t \in [k] \qquad (7)$$

$$\sum_{q\in\Lambda^*(p)} c_q^i x_q \leq x_p, \forall p \in V^\circ, i \in [m] \qquad (8)$$

Constraints (2)-(5) in the LP are for the consistency requirements. (2) says the root is always in T. (3) says if a selector node p is in T, then exactly one of its children is in T. (4) says if a copier node p is in T, and q is a child of p, then q is also in T. (5) is the non-negativity condition. (6) and (7) deal with the covering requirements. (6) says if p is in T, then we choose at most one descendant of p to cover the group S'_t; notice that the constraint implies $y_p^t \leq x_p$ if $p \in S'_t$. (7) says we choose exactly one node in T to cover S'_t. (8) handles the cost requirement: If p is included in T, then the type-i cost of the descendants of p in T is at most 1.

3.3 The rounding algorithm

The rounding algorithm is based on [9]. We solve LP(2) to obtain a solution $x \in [0, 1]^{V^\circ}$. We add r to T and call recursive-rounding(r) to obtain a sub-tree $T = (V, E)$. The procedure is defined in Algorithm 3.

▶ **Observation 12.** *T is always consistent. For every $p \in V^\circ$, we have $\Pr[p \in V] = x_p$.*

Proof. For a selector node p in T, we always choose exactly one child of p and add it to T. For a copier node p added to T and one of its child q, q is added to T with probability 1. By the probabilities we add nodes to T, we can see that $\Pr[p \in V] = x_p$ for every $p \in V^\circ$. ◀

> ■ **Algorithm 3** recursive-rounding(p).

1: **if** p is a selector node **then**
2:      choose one vertex $q \in \Lambda(p)$ randomly, so that q is chosen with probability $\frac{x_q}{x_p}$
3:      add q to T, and call recursive-rounding(q)
4: **else**   ▷ p is a copier or leaf node
5:      **for** every $q \in \Lambda(p)$ **do**
6:          with probability $\frac{x_q}{x_p} = 1$: add q to T, and call recursive-rounding(q)

---

4   Analysis of probabilities of group coverage

We fix $t \in [k]$ and analyze the probability that T coves the group S'_t; or equivalently, the correspondent labeling covers the group S_t. This will prove Property a. For every vertex $p \in V^\circ$, we define $z_p := \sum_{q \in \Lambda^*(p) \cap S'_t} y^t_q$, which indicates whether S'_t is covered by vertices in the sub-tree rooted at p. By (6), we have $z_p \leq x_p$. By (7), we have $z_r = 1 = x_r$.

We define the height of a node $p \in V^\circ$ to be the maximum number of copier nodes in a path from p to one of its descendant leaves. We prove in Appendix A the following lemma:

> ▶ **Lemma 13.** *Assume $p \in V^\circ$ has height h. Then* $\Pr\left[\Lambda^*(p) \cap V \cap S'_t \neq \emptyset \big| p \in V\right] \geq \frac{1}{h+1}\frac{z_p}{x_p}.$

Notice that the height of the root r of T° is $D-1$. Applying the above lemma with $p = r$, we have that T covers group S'_t with probability at least $\frac{1}{D} \cdot \frac{z_r}{x_r} = \frac{1}{D}$. So, the correspondent $\vec{\ell}$ covers S_t with probability at least $\frac{1}{D}$, proving Property a.

4.1   Concentration bound on costs

In this section, we prove Property b. We fix an index $i \in [m]$ and analyze the type-i cost of $T = (V, E)$. For notation convenience, we use c_p to denote c^i_p, and cost for type-i cost.

For every vertex $p \in V^\circ$, let $w_p = \sum_{q \in \Lambda^*(p)} c_q x_q$ be the fractional cost incurred by the sub-tree of T° rooted at p. By (8), we have $w_p \leq x_p$. Let $W_p = \sum_{q \in \Lambda^*(p) \cap V} c_q$ be the cost of T incurred by descendants of p. So, we have $\mathbb{E}[W_p] = w_p$.

As is typical, we shall introduce a parameter $s > 0$ and consider the expectation of the random exponential variables $\mathbf{e}^{sW_p}$. Later we shall set $s = \ln(1 + \frac{1}{2D})$, but the main lemma holds for any $s > 0$. We define an α_h for every integer $h \geq 0$ as $\alpha_0 = \mathbf{e}^s$ and $\alpha_h = \mathbf{e}^{\alpha_{h-1}-1}, \forall h \geq 1$. Notice that $\alpha_0, \alpha_1, \ldots$ is an increasing sequence.

In this section, we count selector nodes in the definition of heights: the height of a node $p \in V^\circ$ is the maximum number of selector nodes in a path from p to its descendant leaf.

> ▶ **Lemma 14.** *For any node p in T° of height h, we have* $\mathbb{E}\left[\mathbf{e}^{sW_p}\big|p \in V\right] \leq \alpha_h^{w_p/x_p}.$

Proof. We prove the lemma for nodes p from bottom to top of the tree T°. Focus on a node p of height h. If p is a copier or leaf node, all children of p has height at most h. We have

$$\mathbb{E}\left[\mathbf{e}^{sW_p}\big|p \in V\right] = \mathbf{e}^{sc_p} \prod_{q \in \Lambda(p)} \mathbb{E}\left[\mathbf{e}^{sW_q}\big|q \in V\right]$$

$$= \alpha_0^{c_p x_p/x_p} \prod_{q \in \Lambda(p)} \alpha_h^{w_q/x_p} \leq \alpha_h^{c_p x_p/x_p} \prod_{q \in \Lambda(p)} \alpha_h^{w_q/x_p} = \alpha_h^{w_p/x_p}.$$

The last inequality used that $\alpha_0 \leq \alpha_h$, and the last equality used that $w_p = c_p x_p + \sum_{q \in \Lambda(p)} w_q$.

Now suppose p is a selector. Then all children of p have height at most $h-1$. Conditioned on $p \in V$, the rounding procedure adds exactly one child q of p to V. Then, we have

$$\mathbb{E}\left[\mathbf{e}^{sW_p}\big|p \in V\right] = \mathbf{e}^{sc_p} \cdot \sum_{q \in \Lambda(p)} \frac{x_q}{x_p} \mathbb{E}\left[\mathbf{e}^{sW_q}\big|q \in V\right] = \mathbf{e}^{sc_p} \cdot \sum_{q \in \Lambda(p)} \frac{x_q}{x_p} \alpha_{h-1}^{w_q/x_q}$$

$$\leq \mathbf{e}^{sc_p}\left(\left(\frac{w_p}{x_p} - c_p\right) \cdot \alpha_{h-1} + \left(1 - \frac{w_p}{x_p} + c_p\right)\right) = \mathbf{e}^{sc_p}\left(1 + \left(\frac{w_q}{x_q} - c_p\right)(\alpha_{h-1} - 1)\right)$$

$$\leq \mathbf{e}^{sc_p} \cdot \exp\left(\left(\frac{w_p}{x_p} - c_p\right)(\alpha_{h-1} - 1)\right) = \mathbf{e}^{sc_p} \cdot \alpha_h^{w_p/x_p - c_p} \leq \alpha_h^{w_p/x_p}.$$

To see the inequality in the second line, we notice the following four facts: (i) α_{h-1}^θ is a convex function of θ, (ii) $w_q/x_q \in [0,1]$ for every $q \in \Lambda(p)$, (iii) $\sum_{q \in \Lambda(p)} \frac{x_q}{x_p} = 1$ and (iv) $\sum_{q \in \Lambda(p)} \frac{x_q}{x_p} \cdot \frac{w_q}{x_q} = \sum_{q \in \Lambda(p)} \frac{w_q}{x_p} = \frac{w_p}{x_p} - c_p$. The equality in the last line is by the definition of α_h. The last inequality used that $\mathbf{e}^s = \alpha_0 \leq \alpha_h$.   ◄

The height of the root r is D.[3] Now, we set $s = \ln(1 + \frac{1}{2D})$.

▶ **Lemma 15.** *For every $h \in [0, D]$, we have $\alpha_h \leq 1 + \frac{1}{2D-h}$.*

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

$$\alpha_h = \mathbf{e}^{\alpha_{h-1}-1} \leq \mathbf{e}^{\frac{1}{2D-h+1}} \leq 1 + \frac{1}{2D-h+1} + \left(\frac{1}{2D-h+1}\right)^2$$

$$= 1 + \frac{2D-h+2}{(2D-h+1)^2} \leq 1 + \frac{1}{2D-h}.$$

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

To wrap up, we apply Lemma 14 on $p = r$. Notice that $r \in V$ always happens, at $W_r = \text{cost}^i(T)$. We have $\mathbb{E}\left[\exp\left(\ln\left(1 + \frac{1}{2D}\right) \cdot \text{cost}^i(T)\right)\right] \leq \alpha_D^{w_r/x_r} \leq 1 + \frac{1}{D}$ by Lemma 15 and that $w_r \leq x_r = 1$. Using the correspondence between sub-trees of T° and labelings of $\mathbf{T}$ proves Property b.

───── **References** ─────

1   B. Awerbuch, Y. Azar, E. Grove, M. Kao, P. Krishnan, and J. Vitter. Load balancing in the l_p norm. In *FOCS*, pages 383–391, 1995.

2   Y. Azar and S. Taub. All-norm approximation for scheduling on identical machines. In T. Hagerup and J. Katajainen, editors, *SWAT*, volume 3111, pages 298–310, 2004.

3   Hans L. Bodlaender. Nc-algorithms for graphs with small treewidth. In Jan van Leeuwen, editor, *Graph-Theoretic Concepts in Computer Science, 14th International Workshop, WG '88, Amsterdam, The Netherlands, June 15-17, 1988, Proceedings*, volume 344 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 1988. `doi:10.1007/3-540-50728-0_32`.

4   P. Chalermsook, S. Das, B. Laekhanukit, and D. Vaz. Beyond metric embedding: Approximating group steiner trees on bounded treewidth graphs. In *SODA*, pages 737–751, 2017.

─────

[3] The height of r is $D + 1$ by definition, but Lemma 14 holds when we define its height to be D, as one can collapse the first two levels of T° into one level.

5   Eden Chlamtác, Michael Dinitz, and Thomas Robinson. Approximating the Norms of Graph Spanners. In Dimitris Achlioptas and László A. Végh, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*, volume 145 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:22, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.APPROX-RANDOM.2019.11`.

6   Eden Chlamtác, Michael Dinitz, and Thomas Robinson. The Norms of Graph Spanners. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.ICALP.2019.40`.

7   Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *STOC*, pages 624–633, 2014.

8   M. Furer and B. Raghavachari. Approximating the minimum-degree steiner tree to within one of optimal. *Journal of Algorithms*, 17(3):409–423, 1994.

9   N. Garg, G. Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group Steiner tree problem. *J. Algorithms*, 37(1):66–84, 2000.

10   D. Golovin, A. Gupta, A. Kumar, and K. Tangwongsan. All-norms and all-l_p-norms approximation algorithms. In *IACS*, volume 2 of *LIPIcs*, pages 199–210, 2008.

11   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. *SIAM Journal on Computing*, 52(2):STOC19–298–STOC19–322, 2023. `doi:10.1137/20M1312988`.

12   X. Guo, G. Kortsarz, B. Laekhanukit, S. Li, D. Vaz, and J. Xian. On approximating degree-bounded network design problems. *Algorithmica*, 84(5):1252–1278, 2022.

13   Mohammad Taghi Hajiaghayi. A list of open problems in bounded degree network design. *The 8'th Workshop on Flexible Network Design*, 2016.

14   E. Halperin and R. Krauthgamer. Polylogarithmic inapproximability. In *STOC*, pages 585–594, 2003.

15   Sungjin Im and Shi Li. Improved approximations for unrelated machine scheduling. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2917–2946, 2023. `doi:10.1137/1.9781611977554.ch111`.

16   K. Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

17   Guy Kortsarz and Zeev Nutov. The minimum degree group steiner problem. *Discret. Appl. Math.*, 309:229–239, 2022.

18   V. Kumar, M. Marathe, S. Parthasarathy, and A. Srinivasan. A unified approach to scheduling on unrelated parallel machines. *J. ACM*, 56(5):28:1–28:31, 2009.

19   C. Lau L, R. Ravi, and M. Singh. *Iterative Methods in Combinatorial Optimization*. Cambridge University Press, 2011.

20   L. C. Lau, J. Naor, M. R. Salavatipour, and M. Singh. Survivable network design with degree or order constraints. *SIAM J. Comput.*, 39(3):1062–1087, 2009.

21   Shi Li, Chenyang Xu, and Ruilong Zhang. Polylogarithmic Approximation for Robust s-t Path. In *51st International Colloquium on Automata, Languages, and Programming (ICALP 2024)*, 2024.

22   Manmeet Kaur Nisha Sharma. A survey of vlsi techniques for power optimization and estimation of optimization. *International Journal of Emerging Technology and Advanced Engineering*, 4, 2014.

23   Mohit Singh and Lap Chi Lau. Approximating minimum bounded degree spanning trees to within one of optimal. *J. ACM*, 62(1), March 2015. `doi:10.1145/2629366`.

24   Y. Wang, X. Hong, T. Jing, Y. Yang, X. Hu, and Guiying Yan. An efficient low-degree RMST algorithm for VLSI/ULSI physical design. In *PATMOS*, pages 442–452, 2004.

A   Proof of Lemma 13

We repeat the lemma below:

▶ **Lemma 13.** *Assume $p \in V^\circ$ has height h. Then* $\Pr\left[\Lambda^*(p) \cap V \cap S'_t \neq \emptyset \middle| p \in V\right] \geq \frac{1}{h+1}\frac{z_p}{x_p}$.

Proof. If $p \in S'_t$ then $\Pr\left[\Lambda^*(p) \cap V \cap S'_t \neq \emptyset \middle| p \in V\right] = 1 \geq \frac{z_p}{x_p}$. The inequality holds trivially. So, we can assume $p \notin S'_t$, and we prove the lemma for nodes p from bottom to top in the tree T°. Suppose p is a leaf; then $h = 0$, and $z_p = 0$ as we assumed $p \notin S'_t$. The inequality trivially holds.

So we can assume p to be a non-leaf node of height h and assume the lemma holds for every $q \in \Lambda(p)$. First, assume p is a selector node. Then all children of p have height at most h.

$$\Pr\left[\Lambda^*(p) \cap V \cap S'_t \neq \emptyset \middle| p \in V\right] \geq \sum_{q \in \Lambda(p)} \frac{x_q}{x_p} \cdot \frac{1}{h+1} \cdot \frac{z_q}{x_q} \geq \sum_{q \in \Lambda(p)} \frac{1}{h+1} \cdot \frac{z_q}{x_p} = \frac{1}{h+1} \cdot \frac{z_p}{x_p}.$$

Then consider the case that p is a copier node. All children of p have height at most $h-1$. Even though p has exactly two children, our analysis works if it has any number of children.

$$\Pr\left[\Lambda^*(p) \cap V \cap S'_t \neq \emptyset \middle| p \in V\right] \geq 1 - \prod_{q \in \Lambda(p)} \left(1 - \frac{1}{h} \cdot \frac{z_q}{x_q}\right) = 1 - \prod_{q \in \Lambda(p)} \exp\left(-\frac{1}{h} \cdot \frac{z_q}{x_p}\right)$$

$$= 1 - \exp\left(-\frac{1}{h} \cdot \frac{z_p}{x_p}\right) \geq \frac{1}{h} \cdot \frac{z_p}{x_p} - \frac{1}{2}\left(\frac{1}{h} \cdot \frac{z_p}{x_p}\right)^2 \geq \frac{1}{h} \cdot \frac{z_p}{x_p} - \frac{1}{2}\left(\frac{1}{h}\right)^2 \frac{z_p}{x_p}$$

$$= \left(\frac{2h-1}{2h^2}\right)\frac{z_p}{x_p} \geq \frac{1}{h+1} \cdot \frac{z_p}{x_p}.$$

The first equality in the first line used that $x_q = x_p$ for every $q \in \Lambda(p)$. The second equality used that $z_p = \sum_{q \in \Lambda(p)} z_q$ as $p \notin S'_t$. The first inequality in the second line used that $e^{-\theta} \leq 1 - \theta + \frac{\theta^2}{2}$ for every $\theta \geq 0$. The second inequality used that $\frac{z_p}{x_p} \leq 1$. ◀

B   Reduction of Degree-Bounded Group Steiner Tree on Bounded-Treewidth Graphs to Tree-Labeling Problem

In this section, we prove Theorem 2, by reducing Group Steiner Tree with degree bounds on bounded treewidth graphs to the tree labeling problem studied in Section 3. Recall the input of the problem contains a graph $G = (V, E)$ with edge costs $c \in R^E_{\geq 0}$, a root r, k groups $S_1, S_2, \cdots, S_k$ of vertices, and a degree bound $\mathrm{db}_v \in \mathbb{Z}_{>0}$ for every $v \in V$. Without loss of generality, we assume $\{r\}, S_1, S_2, \cdots, S_k$ are mutually disjoint. Again, we use opt to denote the minimum cost of a valid subgraph H.

Let $\mathbf{T} = (B, \mathbf{E})$ be the tree decomposition of the graph $G = (V, E)$. Every $b \in B$ is called a bag and let $X_b \subseteq V$ be the set of vertices contained in the bag b. We can add the root r to all the bags, which increases the maximum size of a bag by at most 1. It was shown in [3] that we can assume $\mathbf{T}$ is a rooted binary tree of depth $O(\log n)$, by sacrificing the bag size by an $O(1)$ factor. We summarize the properties as follows:

- $\mathbf{T}$ is a full binary tree rooted at $\mathbf{r}$, with depth $O(\log n)$.
- $|X_b| \leq O(1) \cdot \mathrm{tw}$ for every $b \in B$.
- For every edge $(u, v) \in E$, there is some $b \in B$ with $\{u, v\} \subseteq X_b$.
- For every $v \in V$, the set of bags b with $v \in X_b$ is connected in $\mathbf{T}$.

For every $e \in E$, let b_e be the highest node b such that X_b contains both end vertices of e. This is well-defined due to the last property in the above list. For every $b \in B$, we let $E_b = \{e \in E : b_e = b\}$. So, $(E_b)_{b \in B}$ forms a partition of E.

Notations on Partitions.   Given two partitions Π and Π' of a common set X, we say Π' refines Π if any two elements in X that are in the same set in Π' are also in the same set in Π. We use $\Pi' \leq \Pi$ to denote that Π' refines Π. Given two partitions Π and Π' of X, we use $\Pi \vee \Pi'$ to denote the join of Π and Π' w.r.t the relation $\leq$. That is, we define a graph where there is an edge between u and v if they are in the same set in Π or Π'. Then two vertices u and v are in the same set in the partition $\Pi \vee \Pi'$ if and only if they are in the same connected component in the graph.

Abusing notations slightly, if an element v is not included in a partition Π, we treat $\{v\}$ as a singleton set in Π. This allows us to extend the operators $\leq$ and $\vee$ to two partitions Π and Π' with different ground sets. Given a partition Π and a set X, we let $\Pi[X]$ be the partition Π restricted to the ground set X: two elements $u, v \in X$ are in the same set in $\Pi[X]$ if and only if they are in the same set in Π.

For any set $F \subseteq E$ of edges, we define $\mathrm{CC}(F)$ to be the partition of the vertices incident to F, such that u and v are in the same set in $\mathrm{CC}(F)$ if and only if they are in the same connected component in (V, F).

Construction of Labels and Consistency Triples.   The tree $\mathbf{T}$ for the tree-labeling instance is the same as the decomposition tree $\mathbf{T}$. (This is the reason we use the same notion $\mathbf{T}$.) So we have $\mathbf{V} = B$. Now we fix a bag $b \in B$ and define the set L_b of labels for b. To define the labels, we let $H = (V_H, E_H)$ be any sub-graph of G, which we should think of as the output of the GST problem. Fix a bag $b \in B$, let $\Lambda^*(b)$ be the set of descendants of b in $\mathbf{T}$, including b itself. We then make the following definitions:

- $F_b(H) := E_H \cap E_b$ is the set of edges from E_b that are included in H.
- $\Pi_b^\downarrow(H)$ is the partition of X_b so that two vertices $u, v \in X_b$ is in the set in $\Pi_b^\downarrow(H)$ if and only if they are connected in the graph $(V_H, E_H \cap \bigcup_{b' \in \Lambda^*(b)} E_{b'})$.
- $\Pi_b^\uparrow(H)$ is the partition of X_b so that two vertices $u, v \in X_b$ is in the set in $\Pi_b^\downarrow(H)$ if and only if they are connected in the graph $(V_H, E_H \cap \bigcup_{b' \in B \setminus \Lambda^*(b) \cup \{b\}} E_{b'})$.

In words, $\Pi_b^\downarrow(H)$ and $\Pi_b^\downarrow(H)$ respectively indicate the partition of X_b correspondent to the edges of H in bags below and above b respectively.

Without knowing H, we can define the label set L_b for b to be all tuples $(F_b, \Pi_b^\downarrow, \Pi_b^\uparrow)$ such that $(F_b, \Pi_b^\downarrow, \Pi_b^\uparrow) = (F_b(H), \Pi_b^\downarrow(H), \Pi_b^\uparrow(H))$ for some valid output graph H. We then define the consistency tuples Γ_b's so that a consistent labeling gives a valid outputs sub-graph H.

Formally, let L_b be the set of all tuples $(F_b, \Pi_b^\downarrow, \Pi_b^\uparrow)$ such that
- $F_b \subseteq E_b$ is a forest over X_b, $\mathrm{CC}(F_b) \leq \Pi_b^\downarrow$ and $\mathrm{CC}(F_b) \leq \Pi_b^\uparrow$,
- if $b = \mathbf{r}$, then $\Pi_b^\uparrow = \mathrm{CC}(F_b)$, and
- if b is a leaf, then $\Pi_b^\downarrow = \mathrm{CC}(F_b)$.

Then we define the set Γ_b of triples, for an inner vertex b in $\mathbf{T}$ with two children b' and b''. We have $\big((F_b, \Pi_b^\downarrow, \Pi_b^\uparrow), (F_{b'}, \Pi_{b'}^\downarrow, \Pi_{b'}^\uparrow), (F_{b''}, \Pi_{b''}^\downarrow, \Pi_{b''}^\uparrow)\big) \in \Gamma_b$ if and only if
- $\Pi_b^\downarrow = \big(\Pi_{b'}^\downarrow \vee \Pi_{b''}^\downarrow \vee \mathrm{CC}(F_b)\big)[X_b]$,
- $\Pi_{b'}^\uparrow = \big(\Pi_b^\uparrow \vee \Pi_{b''}^\downarrow \vee \mathrm{CC}(F_{b'})\big)[X_b]$, and
- $\Pi_{b''}^\uparrow = \big(\Pi_b^\uparrow \vee \Pi_{b'}^\downarrow \vee \mathrm{CC}(F_{b''})\big)[X_{b''}]$.

▷ **Claim 16.** Let $\{(F_b, \Pi_b^\downarrow, \Pi_b^\uparrow)\}_{b \in B}$ be a consistent labeling of the tree $\mathbf{T}$. Let $H = (V, \bigcup_{b \in B} F_b)$. Then we have $\Pi_b^\downarrow[H] = \Pi_b^\downarrow$ and $\Pi_b^\uparrow[H] = \Pi_b^\uparrow$ for every $b \in B$.

The claim says that if the labels are consistent, then $\Pi_b^\downarrow$ and $\Pi_b^\uparrow$ represent their true values.

Construction of Covering and Cost Constraints. The requirement that all groups are connected to r can be captured by the covering constraint in the tree-labeling problem. For every $t \in [k]$, a label $(F_b, \Pi_b^\downarrow, \Pi_b^\uparrow) \in L_b$ for some $b \in B$ can satisfy the group S_t if for some $s \in S_t$ we have (s, r) are in the same set in the partition $\Pi_b^\downarrow \vee \Pi_b^\uparrow$.

The edge costs and degree constraints can be captured by the cost constraints in the tree-labeling instance. Consider the costs first. Using binary search, we assume we know the optimum cost C^* for the instance. For every bag $b \in B$ and every label $(F_b, \Pi_b^\downarrow, \Pi_b^\uparrow)$, the cost of the label is $c(F_b) := \sum_{e \in F_b} c_e$. We disallow this label by removing it if $c(F_b) > C^*$. Scaling all costs by C^* so that all costs are in $[0, 1]$. So, the cost being at most C^* in the group Steiner tree instance is equivalent to that the cost of all labels is at most 1.

Finally, we consider the degree constraints $d_H(v) \leq \mathrm{db}_v$ for every $v \in V$. For every $v \in V$, we define a cost constraint in the tree-labeling instance. For every bag $b \in B$ with $v \in X_b$, and every label $(F_b, \Pi_b^\downarrow, \Pi_b^\uparrow)$, the cost of the label is $|\delta(v) \cap F_b|$, where $\delta(v)$ is the incident edges of v in G. Again, we disallow the label if $|\delta(v) \cap F_b| > \mathrm{db}_v$, and we scale the costs by db_v so that all costs are in $[0, 1]$. Then the degree constraint on v is reduced to this cost requirement in the tree labeling instance.

Wrapping Up. We then run the algorithm in Theorem 10 on the constructed tree-labeling instance. Let $(F_b, \Pi_b^\downarrow, \Pi_b^\uparrow)$ be the label of a bag b, and let $H = (V, \bigcup_{b \in B} F_b)$. By Claim 16, the consistency constraints guarantee that the $\Pi_b^\downarrow$ and $\Pi_b^\uparrow$ truthfully represent the connectivity of the graph G. So, if the covering constraint for a group S_t is satisfied, then H indeed connects r and S_t. Recall that $D = O(\log n)$ is the depth of the tree $\mathbf{T}$. By Properties a and b, we have

- For every $t \in [k]$, H connects r and S_t with probability at least $\frac{1}{D}$.
- $\mathbb{E}\left[\exp(\ln(1 + \frac{1}{2D}) \cdot \frac{c(H)}{C^*})\right] \leq 1 + \frac{1}{D}$.
- $\mathbb{E}\left[\exp(\ln(1 + \frac{1}{2D}) \cdot \frac{d_H(v)}{\mathrm{db}_v})\right] \leq 1 + \frac{1}{D}$ for every $v \in V$.

We run the algorithm for $M = \Theta(D \log n) = \Theta(\log^2 n)$ times, with a large hidden constant in the $O(\cdot)$ notation, and output the union H of all sub-graphs constructed by the M times. With high probability, all groups are connected to r in H. $\mathbb{E}[\exp(\ln(1 + \frac{1}{2D}) \cdot \frac{d_H(v)}{\mathrm{db}_v})] \leq (1 + \frac{1}{D})^M = n^{O(1)}$. Using Markov inequality, we have $\exp(\ln(1 + \frac{1}{2D}) \cdot \frac{d_h(v)}{\mathrm{db}_v}) \leq n^{O(1)}$ for every $v \in V$ with high probability. That is, $d_h(v) \leq O(\mathrm{db}_v \log n \cdot D) = O(\log^2 n)\mathrm{db}_v$ with high probability. Similarly, with high probability, we have $c(H) \leq O(\log^2 n)C^*$.

We then analyze the running time of the algorithm. The key parameter deciding the running time is Δ, the maximum size of a label set L_b. As we assumed F_b is a forest over X_b and $|X_b| \leq O(\mathrm{tw})$, there are $\mathrm{tw}^{O(\mathrm{tw})}$ different possibilities for F_b. There are also $\mathrm{tw}^{O(\mathrm{tw})}$ possibilities for each of $\Pi_b^\downarrow$ and $\Pi_b^\uparrow$. So, $|L_b| \leq \mathrm{tw}^{O(\mathrm{tw})}$ for every $b \in B$. Therefore, the running time of the algorithm is $\mathrm{poly}(n) \cdot \Delta^{O(D)} = \mathrm{poly}(n) \cdot (\mathrm{tw}^{O(\mathrm{tw})})^{O(\log n)} = n^{O(\mathrm{tw} \log \mathrm{tw})}$. This finishes the proof of Theorem 2.