New Results on a General Class of Minimum Norm Optimization Problems

Kuowen Chen* Jian Li† Yuval Rabani‡ Yiran Zhang§

Abstract

We study the general norm optimization for combinatorial problems, initiated by Chakrabarty and Swamy (STOC 2019). We propose a general formulation that captures a large class of combinatorial structures: we are given a set $\mathcal{U}$ of n weighted elements and a family of *feasible* subsets $\mathcal{F}$. Each subset $S \in \mathcal{F}$ is called a feasible solution/set of the problem. We denote the *value vector* by $\boldsymbol{v} = \{\boldsymbol{v}_i\}_{i\in[n]}$, where $\boldsymbol{v}_i \geq 0$ is the value of element i. For any subset $S \subseteq \mathcal{U}$, we use $\boldsymbol{v}[S]$ to denote the n-dimensional vector $\{v_e \cdot \mathbf{1}[e \in S]\}_{e\in\mathcal{U}}$ (i.e., we zero out all entries that are not in S). Let $f : \mathbb{R}^n \to \mathbb{R}_+$ be a symmetric monotone norm function. Our goal is to minimize the norm objective $f(\boldsymbol{v}[S])$ over feasible subset $S \in \mathcal{F}$. The problem significantly generalizes the corresponding min-sum and min-max problems.

We present a general equivalent reduction of the norm minimization problem to a multi-criteria optimization problem with logarithmic budget constraints, up to a constant approximation factor. Leveraging this reduction, we obtain constant factor approximation algorithms for the norm minimization versions of several covering problems, such as interval cover, multi-dimensional knapsack cover, and logarithmic factor approximation for set cover. We also study the norm minimization versions for perfect matching, s-t path and s-t cut. We show the natural linear programming relaxations for these problems have a large integrality gap. To complement the negative result, we show that, for perfect matching, it is possible to obtain a bi-criteria result: for any constant $\epsilon, \delta > 0$, we can find in polynomial time a nearly perfect matching (i.e., a matching that matches at least $1-\epsilon$ proportion of vertices) and its cost is at most $(8+\delta)$ times of the optimum for perfect matching. Moreover, we establish the existence of a polynomial-time $O(\log\log n)$-approximation algorithm for the norm minimization variant of the s-t path problem. Specifically, our algorithm achieves an α-approximation with a time complexity of $n^{O(\log\log n/\alpha)}$, where $9 \leq \alpha \leq \log\log n$.

Keywords: Approximation Algorithms, Minimum Norm Optimization, Linear Programming

1 Introduction

In many optimization problems, a feasible solution typically induces a multi-dimensional value vector (e.g., by the subset of elements of the solution), and the objective of the optimization problem is to minimize either the total sum (i.e., ℓ_1 norm) or the maximum (i.e., ℓ_∞ norm) of the vector entry. For example, in the minimum perfect matching problem, the solution is a subset of edges and the induced value vector is the weight vector of the matching (i.e., each entry of the vector is the weight of edge if the edge is in the matching and 0 for a non-matching edge) and we would like to minimize the total sum. Many of such problems are fundamental in combinatorial optimization but require different algorithms for their min-sum and min-max variants (and other possible variants). Recently there have been a rise of interests in developing

*Tsinghua University, ckw22@mails.tsinghua.edu.cn
†Tsinghua University, lapordge@gmail.com
‡The Hebrew University of Jerusalem, yrabani@cs.huji.ac.il
§Tsinghua University, zhangyir22@mails.tsinghua.edu.cn

algorithms for more general objectives, such as ℓ_p norms [4, 24], top-ℓ norms [22], ordered norms [10, 13] and more general norms [1, 14, 15, 20, 31, 33], as interpolation or generalization of min-sum and min-max objectives. The algorithmic study of such generalizations helps unify, interpolate and generalize classic objectives and algorithmic techniques.

The study of approximation algorithm for general norm minimization problems is initiated by Chakrabarty and Swamy [14]. They studied two fundamental problems, load balancing and k-clustering, and provided constant factor approximation algorithm for these problems. For load balancing, the induced value vector is the vector of machine loads and for k-clustering the vector is the vector of service costs. Subsequently, the norm minimization has been studied for a variety of other combinatorial problem such as general machine scheduling problem [20], stochastic optimization problems [31], online algorithms [39], parameterized algorithms [1] etc. In this paper, we study the norm optimization problem for a general set of combinatorial problems. In our problem, a feasible set is a subset of elements and the multi-dimensional value vector is induced by the subset of elements of the solution. Our problem is defined formally as follows:

Definition 1.1. (The Norm Minimization Problem ($\mathsf{MinNorm}$)) We are given a set $\mathcal{U} = [n]$ of n weighted elements and a family of *feasible* subsets $\mathcal{F}$. Each subset $S \in \mathcal{F}$ is called a feasible solution/set of the problem. We denote the *value vector* by $\boldsymbol{v} = \{\boldsymbol{v}_i\}_{i\in[n]}$, where $\boldsymbol{v}_i \geq 0$ is the value of element i. We say a subset $S \subseteq \mathcal{U}$ *feasible* if $S \in \mathcal{F}$. For any subset $S \subseteq \mathcal{U}$, we use $\boldsymbol{v}[S]$ to denote the n-dimensional vector $\{v_e \cdot \mathbf{1}[e \in S]\}_{e\in\mathcal{U}}$ (i.e., we zero out all entries that are not in S), and we call $\boldsymbol{v}[S]$ *the value vector induced by* S. Let $f : \mathbb{R}^n \to \mathbb{R}_+$ be a symmetric monotone norm function. Given the norm function $f(\cdot)$, our goal is to find a feasible solution in $\mathcal{F}$ such that the norm of the value vector induced by the solution is minimized, i.e., we aim to solve the following optimization problem

$$\mathsf{MinNorm:} \qquad \text{minimize} \quad f(\boldsymbol{v}[S]) \qquad \text{subject to} \qquad S \in \mathcal{F}.$$

Note that the case $f(\boldsymbol{v}[S]) = \sum_{e\in S} v_e$ is the most studied min-sum objective and we call the corresponding problem the *original optimization problem*. Other interesting norms include ℓ_p norms, Top-ℓ norms (the sum of top-ℓ entries), ordered norms (see its definition in Section 3). Note that our general framework covers the k-clustering studied in [14]: in the k-clustering problem, the universe $\mathcal{U}$ is the set of edges and each feasible solution in $\mathcal{F}$ is a subset of edges that corresponds to a k-clustering. The load balancing problem does not directly fit into our framework, since one needs to first aggregate the processing times to machine loads, then apply the norm.

Before stating our results, we briefly mention some results that are either known or very simple to derive.

1. (Matroid) Suppose the feasible set $\mathcal{F}$ is a matroid and a feasible solution is a basis of this matroid. In fact, the greedy solution (i.e., the optimal min-sum solution) is the optimal solution for any monotone symmetric norm. This is a folklore result and can be easily seen as follows: First, it is easy to establish the following observation, using the exchange property of matroid: We use $\textsc{Top}_\ell(S)$ to denote the sum of largest ℓ elements of S. For any $\ell \in \mathbb{Z}_{\geq 1}$ and any basis $S \in \mathcal{F}$, $\textsc{Top}_\ell(S_{\text{greedy}}) \leq \textsc{Top}_\ell(S)$ where S_{greedy} is the basis obtained by the greedy algorithm. Then using the majorization lemma by Hardy, Littlewood and Pòlya (Lemma 3.2), we can conclude S_{greedy} is optimal for any monotone symmetric norm.

2. (Vertex Cover) We first relax the problem to the following convex program:

$$\text{min.} \quad f(v_1x_1, \ldots, v_nx_n) \quad \text{s.t.} \quad x_i + x_j \geq 1 \text{ for any } (i,j) \in E.$$

The objective is convex since f is norm (in particular the triangle inequality of norm). Then, we solve the convex program and round all $x_i \geq 1/2$ to 1 and others to 0. It is easy to see this gives a 2-approximation (using the property $f(\alpha x) = \alpha f(x)$ for $\alpha \geq 0$).

3. (Set Cover) The norm-minimization set cover problem is a special case of the generalized load balancing problem introduced in [20]. Here is the reduction: each element corresponds to a job and each subset to a machine; if element i is in set S_j, the processing time $p_{ij} = 1$, otherwise $p_{ij} = \infty$; the inner norm of each machine is the max norm (i.e., ℓ_∞) and the outer norm is $f(\cdot)$. Hence, this implies an $O(\log n)$-approximation for norm-minimization set cover problem using the general result in [20]. The algorithm in [20] is based on a fairly involved configuration LP. In Appendix F, we provide a much simpler randomized rounding algorithm that is also an $O(\log n)$-approximation. Note this is optimal up to a constant factor given the approximation hardness of set cover [21, 23].

4. ($\textsc{Top}_\ell$ and Ordered Norms) If the min-sum problem can be solved or approximated efficiently, one can also solve or approximate the corresponding $\textsc{Top}_\ell$ and ordered norm optimization problems. This mostly follows from known techniques in [10, 14, 22]. For completeness, the general and formal statements are provided in Appendix C.

Our Contributions

Our technical contribution can be summarized as follows:

1. (Theorem 4.1) We present a general reduction of the norm minimization problem to a multi-criteria optimization problem with logarithmic budget constraints, up to a constant approximation factor. This immediately implies an $O(\alpha \log n)$-approximation for the MinNorm problem if there is a poly-time α-approximation for the corresponding weight minimization problem (See Theorem 4.5).

2. Leveraging the reduction in Theorem 4.1, we obtain constant factor approximation algorithms for the norm minimization versions of several covering problems, such as interval covering (Theorem 6.5), multi-dimensional knapsack cover (Theorem 5.1 and Theorem 5.4). These algorithms are based on rounding the natural linear programming relaxation of the multi-criteria optimization problem, possibly with a careful enumeration of partial solutions. For set cover, we obtain a simple randomized approximation algorithm with approximation factor $O(\log n)$ (Theorem F.1), which is much simpler than the general algorithm in [20].

3. We also study the norm minimization versions for perfect matching, s-t path and s-t cut. We show the natural linear programming relaxations for these problems have a large integrality gap (Theorem 7.2 and Theorem 7.3). This indicates that it may be difficult to achieve constant approximation factors for these problems.

4. To complement the above negative result, we show that, for perfect matching, it is possible to obtain a bi-criteria approximation: for any constant $\epsilon > 0$, we can find a nearly perfect matching that matches at least $1 - \epsilon$ proportion of vertices and the norm of this solution is at most $(8 + \delta)$ times of the optimum for perfect matching where δ is any positive real constant (Theorem 9.5).

5. We present an approximate dynamic programming approach that yields a α-approximation $n^{O(\log \log n/\alpha)}$-time algorithm for the min-norm s-t path problem for $9 \leq \alpha \leq \log \log n$ (Theorem 8.1), demonstrating an alternative technique for solving norm minimization problems beyond LP rounding.

2 Related Work

Top-ℓ and Ordered Optimization: As a special case of general norm optimization, ordered optimization for combinatorial optimization problems have received significant attention in the recent years. In fact, an ordered norm can be written as a conical combination of top-ℓ norms (see Claim 3.1). The ordered k-median problem was first studied by Byrka et al. [10] and Aouad and Segev [2]. Byrka et al. [10] obtained the first constant factor approximation algorithm (the factor is $38 + \epsilon$). Independently, Chakrabarty and Swamy [13] obtained an algorithm with approximation factor 18 for the top-ℓ norm), which can be combined with the enumeration procedure of Aouad and Segev [2] to get the same factor for the general ordered k-median. The current best known approximation is 5, by Chakrabarty and Swamy [14]. Deng and Zhang [19] studied ordered k-median with outliers and obtained a constant factor approximation algorithm. Maalouly and Wulf [22] studied the top-ℓ norm optimization for the matching problem and obtained an polynomial time exact algorithm (see also Theorem B.1 in Appendix B). Braverman et al. studied coreset construction for ordered clustering problems [8] which was motivated by applications in machine learning. Batra et al. [5] studied the ordered min-sum vertex cover problem and obtained the first poly-time approximation approximation with approximation factor $2 + \epsilon$.

General Symmetric Norm Optimization: Chakrabarty and Swamy [14] first studied general monotone symmetric norm objectives for clustering and unrelated machine load balancing and obtained constant factor approximation algorithms, substantially generalizing the results for k-Median and k-Center and makespan minimization for unrelated machine scheduling. In a subsequent paper [15], they obtained a simpler algorithm for load balancing that achieves an approximation factor of $2 + \epsilon$. Abbasi et al. [1] studied the parametrized algorithms for the general norm clustering problems and provided the first EPAS (efficient parameterized approximation scheme). Deng et al. [20] introduced the generalized load balancing problem, which further generalizes the problem studied by [15]. In the generalized load balancing problem, the load of a machine i is a symmetric, monotone (inner) norm of the vector of processing times of jobs assigned to i. The generalized makespan is another (outer) norm aggregating the loads. The goal is to find an assignment of jobs to minimize the generalized makespan. They obtained a logarithmic factor approximation, which is optimal up to constant factor since the problem generalizes the set cover problem. For the special case where the inner norms are top-k norms, Ayyadevara et al. [3] showed the natural configuration LP has a $\Omega(\log^{1/2} n)$ integrality gap.

Submodular/Supermodular Optimization: Optimizing submodular/supermodular function under various combinatorial constraints is another important class of optimization problems with general objectives and has been studied extensively in the literature. See e.g., [9, 11, 16, 35] and the survey [34]. However, note that results for submodular functions does not imply results for general symmetric monotone norms, since a general symmetric monotone norm is not necessarily a submodular function (see e.g., [20]).

Patton et al. [39] studied submodular norm objectives (i.e., norms that also satisfies continuous submodular property). They showed that it can approximate well-known classes of norms, such as ℓ_p norms, ordered norms, and symmetric norms and applied it to a variety of problems such as online facility location, stochastic probing, and generalized load balancing. Recently, Kesselheim et al. [33] introduced the notion of p-supermodular norm and showed that every symmetric norm can be approximated by a p-supermodular norm. Leveraging the result, they obtain new algorithms online load-balancing and bandits with knapsacks, stochastic probing and so on.

Multi-budgeted Optimization: There is a body of literature in the problem of optimizing linear or submodular objectives over a combinatorial structure with additional budget constraints (see e.g., [6, 12, 17, 25, 26, 27, 40]). For a single budget constraint, randomized or deterministic

PTASes have been developed for various combinatorial optimization problems (e.g. spanning trees with a linear budget [40]). Assuming that a pseudopolynomial time algorithm for the exact version of the problems exists, Grandoni and Zenklusen showed that one can obtain a PTAS for the corresponding problem with any fixed number of linear budgets [25]. More powerful techniques such as randomized dependent rounding and iterative rounding have been developed to handle more general submodular objectives and/or other combinatorial structures such as matroid or intersection of matroid (e.g., [17, 25, 26, 27]). Iterative rounding technique [26, 37] has been used in general norm minimization problems [14, 15]. Our algorithms for matching (Section 9) and knapsack cover (Section 5) also adopt the technique.

3 Preliminaries

Throughout this paper, for vector $\boldsymbol{v} \in \mathbb{R}_+^n$, define $\boldsymbol{v}^{\downarrow}$ as the non-increasingly sorted version of $\boldsymbol{v}$, and $\boldsymbol{v}[S] = \{\boldsymbol{v}_j \cdot \mathbf{1}[j \in S]\}_{j\in[n]}$ for any $S \subseteq [n]$. Let $\textsc{Top}_k : \mathbb{R}^n \to \mathbb{R}_+$ be the top-k norm that returns the sum of the k largest *absolute values* of entries in any vector, $k \le |n|$. Denote $[n]$ as the set of positive integers no larger than $n \in \mathbb{Z}$, and $a^+ = \max\{a, 0\}$, $a \in \mathbb{R}$.

We say function $f : \mathbb{R}^n \to \mathbb{R}_+$ is a *norm* if: (i) $f(\boldsymbol{v}) = 0$ if and only if $\boldsymbol{v} = 0$, (ii) $f(\boldsymbol{u} + \boldsymbol{v}) \le f(\boldsymbol{u}) + f(\boldsymbol{v})$ for all $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$, (iii) $f(\theta\boldsymbol{v}) = |\theta| f(\boldsymbol{v})$ for all $\boldsymbol{v} \in \mathbb{R}^n, \theta \in \mathbb{R}$. A norm f is *monotone* if $f(\boldsymbol{v}) \le f(\boldsymbol{u})$ for all $0 \le \boldsymbol{v} \le \boldsymbol{u}$, and *symmetric* if $f(\boldsymbol{v}) = f(\boldsymbol{v}')$ for any permutation $\boldsymbol{v}'$ of $\boldsymbol{v}$. We are also interested in the following special monotone symmetric norms.

Top-ℓ norms. Let $\ell \in [n]$. A function is a Top-ℓ norm, denoted by $\textsc{Top}_\ell : \mathbb{R}^n \to \mathbb{R}_+$, if for each input vector $\boldsymbol{v} \in \mathbb{R}^n$ it returns the sum of the largest ℓ *absolute values* of entries in $\boldsymbol{v}$. For non-negative vectors, it simply returns the sum of the largest ℓ entries. We notice that by letting $\ell \in \{1, n\}$, $\textsc{Top}_\ell$ recovers the $\mathcal{L}_\infty$ and $\mathcal{L}_1$ norms, respectively, thus it generalizes the latter two.

Ordered norms. Let $\boldsymbol{v} \in \mathbb{R}_+^n$ be a non-increasing non-negative vector. For each vector $\boldsymbol{v} \in \mathbb{R}^{\mathcal{X}}$, let $\boldsymbol{v}^{\downarrow} \in \mathbb{R}^{|\mathcal{X}|}$ denote its non-increasingly sorted version and define $|\boldsymbol{v}| = \{|\boldsymbol{v}_i| : i \in \mathcal{X}\} \in \mathbb{R}_+^{\mathcal{X}}$. A function is a $\boldsymbol{w}$-ordered norm (or simply an ordered norm), denoted by $\textsc{Ord}_{\boldsymbol{w}} : \mathbb{R}^{\mathcal{X}} \to \mathbb{R}_+$, if for each input vector $\boldsymbol{v} \in \mathbb{R}^{\mathcal{X}}$ it returns the inner product of $\boldsymbol{w}$ and $|\boldsymbol{v}|^{\downarrow}$; we obtain $\textsc{Ord}_{\boldsymbol{w}}(\boldsymbol{v}) = \boldsymbol{w}^\top \boldsymbol{v}^{\downarrow}$ whenever $\boldsymbol{v} \in \mathbb{R}_+^{\mathcal{X}}$. It is easy to see that, by having $\boldsymbol{v}$ as a vector of ℓ 1s followed by $(|\mathcal{X}| - \ell)$ 0s, $\textsc{Ord}_{\boldsymbol{w}}$ recovers $\textsc{Top}_\ell$. On the other hand, it is known that each ordered norm can be written as a conical combination of Top-ℓ norms, as in the following claim.

Claim 3.1. *(See, e.g., [14]). For each $\boldsymbol{v} \in \mathbb{R}_+^{\mathcal{X}}$ and another non-increasing vector $\boldsymbol{w} \in \mathbb{R}_+^{|\mathcal{X}|}$, one has*

$$\textsc{Ord}_{\boldsymbol{w}}(\boldsymbol{v}) = \sum_{\ell=1}^{|\mathcal{X}|} (\boldsymbol{w}_\ell - \boldsymbol{w}_{\ell+1}) \textsc{Top}_\ell(\boldsymbol{v}),$$

where we define $\boldsymbol{v}_{|\mathcal{X}|+1} = 0$.

The following lemma is due to Hardy, Littlewood and Pòlya. [29].

Lemma 3.2. *([29]). If $\boldsymbol{v}, \boldsymbol{u} \in \mathbb{R}_+^{\mathcal{X}}$ and $\alpha \ge 0$ satisfy $\textsc{Top}_\ell(\boldsymbol{v}) \le \alpha \cdot \textsc{Top}_\ell(\boldsymbol{u})$ for each $\ell \in [|\mathcal{X}|]$, one has $f(\boldsymbol{v}) \le \alpha \cdot f(\boldsymbol{u})$ for any symmetric monotone norm $f : \mathbb{R}^{\mathcal{X}} \to \mathbb{R}_+$.*

The following results can be found in standard combinatorial optimization textbooks, e.g., Schrijver's book [41].

Lemma 3.3. *Let $\mathcal{N}_1, \mathcal{N}_2$ be two laminar families on a common ground set $\mathcal{X}$ and $A \in \{0,1\}^{(\mathcal{N}_1 \cup \mathcal{N}_2)\times\mathcal{X}}$ be the incidence matrix of $\mathcal{N}_1 \cup \mathcal{N}_2$. Then A is totally unimodular.*

Lemma 3.4. *Let A be a totally unimodular $m \times n$ matrix and $b \in \mathbb{Z}^m$, then the linear program $P = \{Ax \le b\}$ has integral vertex solutions.*

Here are some well-known results from linear programming.

Lemma 3.5. *(See, e.g., Chapter 2 of [7]). Consider a linear program with the polyhedron*

$$P = \{x \in \mathbb{R}^n : A_1 x = b_1,\ A_2 x \le b_2,\ A_3 x \ge b_3\},$$

where $A_1 \in \mathbb{R}^{m_1 \times n}, A_2 \in \mathbb{R}^{m_2 \times n}, A_3 \in \mathbb{R}^{m_3 \times n}$ are matrices and $b_1 \in \mathbb{R}^{m_1}, b_2 \in \mathbb{R}^{m_2}, b_3 \in \mathbb{R}^{m_3}$ are vectors. A point $x^* \in \mathbb{R}^n$ is an extreme point (or a basic feasible solution) of the linear program if and only if: (1) $x^* \in P$, and (2) there exists a set of n linearly independent rows of the matrix $A = \begin{pmatrix} A_1 \\ A_2 \\ A_3 \end{pmatrix}$, which can be assembled into a submatrix A_0 such that $A_0 x^* = b_0$, where b_0 is the subvector of b corresponding to the rows in A_0.

Lemma 3.6. *For a linear program with the polyhedron*

$$P = \{x \in \mathbb{R}^n : A_1 x = b_1,\ A_2 x \le b_2,\ A_3 x \ge b_3\},$$

where $A_1 \in \mathbb{R}^{m_1 \times n}, A_2 \in \mathbb{R}^{m_2 \times n}, A_3 \in \mathbb{R}^{m_3 \times n}$ are matrices and $b_1 \in \mathbb{R}^{m_1}, b_2 \in \mathbb{R}^{m_2}, b_3 \in \mathbb{R}^{m_3}$ are vectors. Let x^* be an extreme point (or basic feasible solution) of the linear program. For an index set $Q \subseteq [n]$ (recall $x^*[Q]$ is the vector that retains only the elements indexed by Q), $x^*[Q]$ is still an extreme point of the linear program with polyhedron

$$P' = \{x[Q] : x \in P,\ x_i = x_i^* \text{ for all } i \in [n] \setminus Q\}.$$

Proof. TOPROVE 0 ☐

4 A General Reduction to Multi-Budgeted Optimization Problem

In this section, we provide an equivalent formulation for the general symmetric norm minimization problem MinNorm (up to constant approximation factor). Recall that as defined in Section 1, we are given a set $\mathcal{U}$ of n elements, and $\mathcal{F}$ represents a family of feasible subsets of $\mathcal{U}$. The goal of MinNorm is to find a feasible subset $S \in \mathcal{F}$ to minimize $f(\boldsymbol{v}[S])$, where f is a symmetric monotone norm function. We say that we find a c-approximation for the problem for some $c \ge 1$, if we can find an S such that $f(\boldsymbol{v}[S]) \le c \cdot f(\boldsymbol{v}[S^*])$, where S^* is the optimal solution. Since a general norm function is quite abstract and hard to deal with, we formulate the following (equivalent, up to constant approximation factor) optimization problem which is more combinatorial in nature.

Definition 4.1 (Logarithmic Budgeted Optimization (LogBgt))**.** The input of a Logarithmic Budgeted Optimization Problem is a tuple $\eta = (\mathcal{U}; S_1, S_2, \ldots, S_T; \mathcal{F})$, where:

- $\mathcal{U}$ is a finite set with n elements.
- $S_1, S_2, \cdots, S_T$ are disjoint subsets of $\mathcal{U}$, where $T = \lceil \log n \rceil$ is the number of sets. For $1 \le i \le T$, We refer to S_i as the i-th group, and for any $u \in S_i$, we call i the group index of u.
- $\mathcal{F}$ is a family of feasible subsets of $\mathcal{U}$. The size of $|\mathcal{F}|$ may be exponentially large in n, but we ensure that there exists a polynomial-time algorithm to decide whether $D \in \mathcal{F}$ for a subset $D \subseteq \mathcal{U}$.

For any $c' > 0$, we say a subset $D \subseteq \mathcal{U}$ is a c'**-valid** solution if and only if:

1. D satisfies the feasibility constraint, i.e., $D \in \mathcal{F}$, and
2. $|D \cap S_i| \le c' \times 2^i$ for all $1 \le i \le T$.

For any $c \ge c_0 \ge 1$, we define (c, c_0)-LogBgt problems as follows: Given an input η, the goal is to find a c-valid solution or certify that there is no c_0-valid solution. In particular, we denote $(c, 1)$-LogBgt as c-LogBgt.

Notice that the structure of a problem is defined by $\mathcal{U}$ and $\mathcal{F}$ (for example, the vertex cover problem is given by vertex set $\mathcal{U}$ and $\mathcal{F}$ contains all subsets of $\mathcal{U}$ corresponding to a vertex cover), so each problem corresponds to a MinNorm version and an LogBgt version. We show that solving LogBgt is equivalent to approximating MinNorm, up to constant approximation factors. In fact, the reduction from norm approximation to optimization problem with multiple budgets has been implicitly developed in prior work [14, 32]. For generality and ease of usage, we encapsulate the reduction in the following general theorem.

Theorem 4.1. *For any $c \ge 1$ (c can depend on n) and $\epsilon > 0$, if we can solve c-LogBgt in polynomial time, we can approximate the MinNorm problem within a factor of $(4c + \epsilon)$ in polynomial time. On the other hand, if we can find a c-approximation for MinNorm in polynomial time, we can solve the $47c^2$-LogBgt in polynomial time.*

In the following, we give a formal proof for Theorem 4.1. We also give another form of it in Theorem D.1 that will be used in the following sections.

To prove Theorem 4.1, we first define some notations. Let the optimal solution for the MinNorm problem be S^*. Define $\boldsymbol{u} = \boldsymbol{v}[S^*]$ and let $o = \boldsymbol{u}^{\downarrow}$ (recall that $\boldsymbol{u}^{\downarrow}$ represents the vector obtained by sorting the elements of $\boldsymbol{u}$ in non-increasing order).

Let δ, ε be (small) positive constants. Define the set of positions

$$\mathrm{POS} = \{\min\{2^s, n\} : s \ge 0\}.$$

We need the following lemma, which is proved in [14]. The lemma says that we can in polynomial time construct a poly-sized set of threshold vectors such that there is one threshold vector which is close to the optimal vector for all positions in POS.

Lemma 4.2. *[14] Let ε be a fixed small positive constant. Suppose we can obtain in polynomial time a set Q of polynomial size that is guaranteed to contain a real number in $[o_1^{\downarrow}, (1+\varepsilon)o_1^{\downarrow}]$. Then, in time $O\left(|Q| \max\{(n/\varepsilon)^{O(1/\varepsilon)}, poly(n)\}\right)$, we can obtain a set T of polynomial many vectors in $\mathbb{R}^{POS}$, which contains a threshold vector (denoted by $\boldsymbol{t}^*$) satisfying: $o_\ell^{\downarrow} \le t_\ell^* \le (1+\varepsilon)o_\ell^{\downarrow}$ if $o_\ell^{\downarrow} \ge \varepsilon o_1^{\downarrow}/n$ and $t_\ell^* = 0$ otherwise. Moreover, for all $\ell \in POS$, t_ℓ^* is either 0 or a power of $1+\varepsilon$.*

Let advance(i) be the smallest element in POS that is at least i, for $1 \le i \le n$. For each $\boldsymbol{t} \in \mathbb{R}^{\mathrm{POS}}$, define $g(\boldsymbol{t})$ to be the n-dimensional vector such that $g(\boldsymbol{t})_i = \boldsymbol{t}_{\mathrm{advance}(i)}$.

Lemma 4.3. *Given a norm function f, if $\boldsymbol{t}$ is a valid threshold (i.e., it satisfies the statement in Lemma 4.2), we have*

$$f(g(\boldsymbol{t})) \le (1+\varepsilon) f(\boldsymbol{o}).$$

Proof. TOPROVE 1 □

Moreover, we have the following lemma:

Lemma 4.4. *We are given an n-dimensional vector $\boldsymbol{u}$, a threshold vector $\boldsymbol{t}$, and a positive integer $c \ge 1$. If for each $\ell \in POS$ $(1 \le i \le T)$, there are at most $c\ell$ entries in $\boldsymbol{u}$ that is larger than $\boldsymbol{t}_\ell$ and the largest entry of $\boldsymbol{u}$ is at most $\boldsymbol{t}_1$, then $f(\boldsymbol{u}) \le 2cf(g(\boldsymbol{t}))$.*

Proof. TOPROVE 2 □

Now we prove Theorem 4.1 which establishes the equivalence between norm approximation and the LogBgt problem.

Proof of Theorem 4.1. We first reduce the MinNorm problem to the LogBgt problem. We enumerate all possible threshold vectors $\boldsymbol{t} \in \mathbb{R}^{\mathrm{POS}}$ as defined in Lemma 4.2 (note that there are only n elements, so we just need to let Q (in Lemma 4.2) be the set of the values of all elements). Suppose $\boldsymbol{t}$ is a valid guess. We can also assume that we have guessed the exact value of $o_1^{\downarrow}$ (because it only has n choices). We construct sets $S_1, \cdots, S_T$ for LogBgt in the following way. For each element $e \in \mathcal{U}$, we do the following:

- if its value v_e is larger than $\boldsymbol{t}_1$, we do not add it to any set;
- if its value v_e is at most $\max\{\boldsymbol{t}_n, \varepsilon o_1^{\downarrow}/n\} = \max\{\boldsymbol{t}_n, \varepsilon \boldsymbol{t}_1^{\downarrow}/n\}$, we add it to S_T;
- otherwise, if its value v_e is at most $\boldsymbol{t}_\ell$ and larger than $\boldsymbol{t}_{\mathrm{next}(\ell)}$, where $\ell = 2^i$, we add it to S_{i+1}, for $0 \le i \le T-1$. Recall next(ℓ) means the next element in POS.

Now, consider the optimal solution $\boldsymbol{o} = \boldsymbol{u}^{\downarrow}[S^*]$ for MinNorm corresponding to the valid guess $\boldsymbol{t}$. For $1 \le i \le T-1$, consider how many elements in S^* that are added in S_i. By definition, an element e in S_i have value $\boldsymbol{v}_e$ larger than $\boldsymbol{t}_{2^i}$. Also, the value v_e is larger than $\varepsilon o_1^{\downarrow}/n$. For $\ell \in \mathrm{POS}$, $o_\ell \le \boldsymbol{t}_\ell$ by definition of valid guess. Thus, since $\boldsymbol{t}$ is a valid guess, there are at most 2^i elements of S^* from S_i (note that this also holds for $i = T = \lceil \log n \rceil$ as there are n elements).

If we can solve the c-LogBgt problem, we can get a solution $S \in \mathcal{F}$ such that there are at most $c \cdot 2^i$ elements in each S_i. As each element e with value $v_e > t_\ell$ for $\ell = 2^i$ cannot be in S_j for $j > i$, for each ℓ, the number of elements with value larger than t_ℓ is at most $2c\ell$. We partition S into A, B such that the elements in A have values less than $\varepsilon o_1/n$ and the elements in B have values at least $\varepsilon o_1/n$. We need this partition because of the condition $o_\ell^{\downarrow} \ge \varepsilon o_1^{\downarrow}/n$ in Lemma 4.2. Note that A, B are not needed in the algorithm, but only useful in the analysis. So we have

$$f(\boldsymbol{v}[S]) \le f(\boldsymbol{v}[A]) + f(\boldsymbol{v}[B]) \le n \cdot f(\varepsilon o_1/n) + 4cf(g(\boldsymbol{t})) \le \varepsilon f(\boldsymbol{o}) + 4c(1+\varepsilon)f(\boldsymbol{o}) \le 4c(1+2\varepsilon)f(\boldsymbol{o})$$

where the first inequality following from the triangle inequality of the norm, the second from the definition of A and Lemma 4.4 and the third from Lemma 4.3. Therefore, the first part of Theorem 4.1 follows.

Now we prove the other direction of Theorem 4.1. Suppose we can c-approximate the norm optimization problem MinNorm for any monotone symmetric norm. Consider the norm

$$f(\mathbf{v}) = \max_{\ell=2^i, 1 \le i \le \log(n+2)} \left\{ \frac{\textsc{Top}_{\ell-2}(\mathbf{v})}{2^{i/2}} \right\}.$$

We construct $\mathcal{U}$ to be the union of $S_1, S_2, \cdots, S_T$ and the elements in S_i have value $1/2^{i/2}$ for any $1 \le i \le T-1$. The elements in S_T have value 0. So for the 1-valid optimal solution S^* for the LogBgt problem, consider the value of $f(\boldsymbol{v}[S^*])$. For any $1 \le i \le T-1$ and the corresponding $\ell = 2^i$, we can see that

$$\textsc{Top}_{\ell-2}(\mathbf{v}) = \sum_{j=1}^{i-1} \frac{2^j}{2^{j/2}} \le \frac{2^{i/2}}{\sqrt{2}-1}.$$

So the norm $f(\boldsymbol{v}[S^*])$ is at most $1/(\sqrt{2}-1)$.

Let $a = \left\lceil \log \frac{4c^2}{(\sqrt{2}-1)^2} \right\rceil$ and $c' = 2^a$. For a feasible solution S for the LogBgt, suppose there are at least $2^a \cdot 2^j$ elements in S_j (i.e. $|S \cap S_j| \ge 2^{a+j}$). We consider the $\textsc{Top}_{\ell-2}(\mathbf{v})$ for

$i = a + j, \ell = 2^i$ (As there are at most n elements, $i \le \log n$). It shows that the norm $f(\boldsymbol{v}[S])$ is at least

$$\frac{2^{a+j} - 2}{2^{(a+j)/2+j/2}} > \frac{1}{2}\sqrt{2^a}$$

as $a + j \ge 3$. So consider the solution $D \in \mathcal{F}$ of LogBgt we get from solving the MinNorm problem by reduction above. If the norm $f(\boldsymbol{v}[D])$ is larger than $c/(\sqrt{2} - 1)$, there is no 1-valid solution for LogBgt. Otherwise, the norm $f(\boldsymbol{v}[D])$ is at most $c/(\sqrt{2} - 1) \le \sqrt{2^a}/2$. Since the norm is at most $\frac{\sqrt{2^a}}{2}$, we know that the solution for the MinNorm problem is a solution for the c'-LogBgt problem. As

$$c' \le 2 \times \frac{4c^2}{(\sqrt{2} - 1)^2} < 47c^2,$$

the second part of Theorem 4.1 follows. □

A Logarithmic Approximation: Based on Theorem 4.1, we can easily deduce the following general theorem. We use $\mathfrak{A}$ to denote a general combinatorial optimization problem with the min-sum objective function $\min_{S \in \mathcal{F}} \boldsymbol{v}(S)$, where we write $\boldsymbol{v}(S) = \sum_{e \in S} v_e$ and $\mathcal{F}$ is the set of feasible solutions.

Theorem 4.5. *If there is a poly-time approximation algorithm for the min-sum problem $\mathfrak{A}$ (with approximation factor $\alpha \ge 1$), there is a poly-time factor $(4\alpha\lceil \log n \rceil + \epsilon)$ approximation algorithm for the corresponding MinNorm problem for any fixed constant $\epsilon > 0$.*

Proof. TOPROVE 3 □

5 Multi-dimensional Knapsack Cover Problem

In this section, we consider the multi-dimensional knapsack cover problem defined as follows.

Definition 5.1 (Min-norm d-dimensional Knapsack Cover Problem (MinNorm-KnapCov))**.** Let d be a positive integer. We are given a set of items $\mathcal{U} = \{1, 2, \ldots, n\}$, where each item $i \in \mathcal{U}$ has a weight vector $w_i \in \mathbb{R}^d$. The feasible set $\mathcal{F}$ is defined as:

$$\mathcal{F} = \left\{ D \subseteq \mathcal{U} : \sum_{v \in D} w_{v,i} \ge 1 \quad \forall i \in \{1, 2, \ldots, d\} \right\}.$$

Now, given a symmetric monotone norm f and a value vector $\boldsymbol{v} \in \mathbb{R}^{\mathcal{U}}_{\ge 0}$, we can define the norm minimization problem for d-dimensional Knapsack Cover and denote it as MinNorm-KnapCov.

In light of Theorem 4.1, we introduce $T = \lceil \log n \rceil$ disjoint sets $S_1, S_2, \ldots, S_T$ and consider the LogBgt problem with $(\mathcal{U}; S_1, \ldots, S_T; \mathcal{F})$, which We denote as LogBgt-KnapCov. We consider LogBgt-KnapCov for two cases: (1) $d = O(1)$ and (2) $d = O(\sqrt{\log n / \log\log W})$ (W will be defined in Section 5.2). For both cases, we use the following natural linear programming formulation for LogBgt-KnapCov :

$$\begin{aligned} \min \quad & 0 \\ s.t. \quad & \sum_{v \in \mathcal{U}} x_v w_{v,i} \ge 1 \quad \forall 1 \le i \le d \\ & \sum_{v \in S_j} x_v \le 2^j \quad \forall 1 \le j \le T \\ & 0 \le x_v \le 1 \quad \forall v \in S_j, 1 \le j \le T \end{aligned} \tag{LP-KnapCover-1}$$

For both cases, we develop a method called **partial enumeration**. Partial enumeration lists a subset of possible partial solutions for the first several groups. Here is the complete definition:

Definition 5.2 (Partial Enumeration)**.** For a (c, c_0)-LogBgt problem with set $S_1, S_2, \cdots S_T$, the **partial enumeration** algorithm first determine a quantity T_0 (depending on the problem at hand). The partial enumeration algorithm returns a subset $X \subseteq 2^{S_1} \times 2^{S_2} \times \cdots \times 2^{S_{T_0}}$. Each element of X is a **partial solution** $(D_1, D_2, \cdots, D_{T_0})$ (Recall the definition of partial solution: $D_i \subseteq S_i$), and this algorithm ensures:

1. If there exists a c_0-valid solution, then at least one partial solution $(D_1, D_2, \cdots, D_{T_0}) \in X$ satisfies that there exists an c-valid **extended solution** (A solution D is called an extended solution of a partial solution $(D_1, \cdots, D_{T_0})$ if $D \cap S_i = D_i$ for all $i = 1, 2, \cdots, T_0$).

2. The size of X is polynomial, and this partial enumeration algorithm runs in polynomial time.

5.1 An Algorithm for $d = O(1)$

In this subsection, we design a polynomial-time constant-factor approximation algorithm for $\mathsf{MinNorm\text{-}KnapCov}$ with $d = O(1)$. [1]

Theorem 5.1. *If d is a constant, then for any constant $\varepsilon > 0$, there exists an polynomial-time algorithm which can solve $(1 + \varepsilon)$-$\mathsf{LogBgt\text{-}KnapCov}$. Thus we have a polynomial-time $(4 + \varepsilon)$-approximation algorithm for $\mathsf{MinNorm\text{-}KnapCov}$ when $d = O(1)$.*

In this subsection, the partial enumeration algorithm needs an integer $T_0 > \log d$ (we discuss how to determine the exact value of T_0 later). Recall that a partial solution can be described as a vector $(D_1, D_2, \cdots, D_{T_0})$, where $D_i \subseteq S_i$ for $i = 1, 2, \cdots, T_0$. Our partial enumeration algorithm in this section simply outputs the set of all possible **partial solution**s (i.e., $X = X_1^{all} \times X_2^{all} \times \ldots \times X_{T_0}^{all}$ where $X_i^{all} = \{X' \subseteq S_i : |X'| \leq 2^i\}$). The number of of such partial solutions can be bounded as in Lemma 5.2.

Then we need to extend a partial solution to a complete solution, i.e., construct the rest of the sets $D_{T_0+1}, D_{T_0+2}, \cdots, D_T$. For a partial solution $(D_1, D_2, \ldots, D_{T_0})$, a solution $D \subseteq \mathcal{U}$ is called an **Extended Solution** of $(D_1, D_2, \ldots, D_{T_0})$ if and only if $D \cap S_j = D_j$ for each $1 \leq j \leq T_0$. Without loss of generality, we sometimes describe an extended solution in the form of $(D_1, D_2, D_3, \ldots, D_T)$.

After the enumeration, we sequentially check these partial solutions. For each partial solution $(D_1, D_2, \ldots, D_{T_0})$, we consider the following linear program:

$$
\begin{aligned}
\min \quad & 0 \\
\text{s.t.} \quad & \sum_{T_0 < j \leq T} \sum_{u \in S_j} x_u w_{u,i} \geq \max\left(0, 1 - \sum_{1 \leq j \leq T_0} \sum_{u \in D_j} w_{u,i}\right) && \forall 1 \leq i \leq d \\
& \sum_{u \in S_j} x_u \leq 2^j && \forall T_0 < j \leq T \\
& 0 \leq x_u \leq 1 && \forall u \in S_j,\ T_0 < j \leq T
\end{aligned}
\tag{LP-KnapCover}
$$

We solve the LP (LP-KnapCover) and perform the following steps:

1. Check the feasibility of Linear Programming (LP-KnapCover). If there is no feasible solution, then return "No Solution" directly. Otherwise, obtain an extreme point x^* for (LP-KnapCover).

[1] We are grateful to an anonymous reviewer for her/his insightful suggestions, which significantly simplifies the algorithm in this subsection (in particular the rounding algorithm Algorithm 1). Here we only present the simplified algorithm.

2. For any $T_0 < j \le T$, round all nonzero x_u^* to 1 for all $u \in S_j$. More specifically, we add them to our extended solution: $D_j \leftarrow \{u \in S_j : x_u^* > 0\}$ for all j such that $T_0 < j \le T$.

Algorithm 1: Rounding Algorithm for d-dimensional Knapsack Cover Problem

Result: Solution D or "No Solution"

$D \leftarrow D_1 \cup D_2 \cup \cdots \cup D_{T_0}$;
if (LP-KnapCover) *has no solution* **then**
  Return "No Solution";
Solve Linear Program (LP-KnapCover) and obtain an extreme point x^*;
$D \leftarrow D \cup \{u \in S_j \mid T_0 < j \le T, x_u^* > 0\}$;
Return D;

Lemma 5.2. *There exists a* $\exp(O(2^{T_0}\log n))$*-time algorithm to enumerate all partial solutions* $(D_1, \cdots, D_{T_0})$ *satisfying* $|D_i| \le 2^i$ *for all* $1 \le i \le T_0$.

Proof. TOPROVE 4 ☐

Lemma 5.3. *If Algorithm 1 returns a solution* D*, then* $|D \cap S_j| \le \left(\frac{d}{2^{T_0}} + 1\right) \cdot 2^j$ *for all* $T_0 < j \le T$.

Proof. TOPROVE 5 ☐

Proof of Theorem 5.1. According to Lemma 5.2 and 5.3, our algorithm runs in $\exp(O(2^{T_0}\log n))$ time, and it can return a $\left(\frac{d}{2^{T_0}} + 1\right)$-valid solution or confirm there is no 1-valid solution. Hence, for any constant $\epsilon > 0$, we can choose $T_0 = \log d + \log(1/\epsilon) + O(1)$. Then, the algorithm runs $\exp(O(d\log n/\epsilon))$-time for $(1+\epsilon)-$LogBgt-KnapCov.

Finally, applying Theorem 4.1, we obtain a polynomial-time $(4+\varepsilon)$-approximation algorithm for the MinNorm-KnapCov problem with $d = O(1)$. ☐

5.2 An Algorithm for Larger d

In this subsection, we provide a polynomial-time constant-factor approximation algorithm for $d = O\left(\sqrt{\frac{\log n}{\log\log W}}\right)$, where W is defined as

$$W = \max_{1\le i\le d} \frac{\max_{u\in\mathcal{U}} w_{u,i}}{\min_{u\in\mathcal{U}, w_{u,i}>0} w_{u,i}}.$$

To ensure there exist valid solutions, $\{u \in \mathcal{U} : w_{u,i} > 0\}$ must be a non-empty set. Here is the main result in this subsection:

Theorem 5.4. *There exists a poly*$(n, \log(W))$ *algorithm that can solve* 2*-LogBgt-KnapCov when* $d = O\left(\sqrt{\frac{\log n}{\log\log W}}\right)$*. Thus we have a* $(4+\varepsilon)$*-approximation algorithm for MinNorm-KnapCov with* $d = O\left(\sqrt{\frac{\log n}{\log\log W}}\right)$.

This algorithm employs the same rounding procedure but modifies the partial enumeration method. The new algorithm choose $T_0 = \lceil \log d \rceil$. For $1 \le j \le T_0$, it partitions S_j into multiple subsets based on vectors of size d, which represent the logarithms of weights. Instead of enumerating all subsets, we only enumerate the number of elements within each subset and then take double the number of any elements in this subset.

Now, we provide an overview of the algorithm for the LogBgt-KnapCov. We first describe the partial enumeration algorithm. We first obtain a set $X_j \subseteq 2^{S_j}$ for each $1 \le j \le T_0$, and then combine them by direct product: $X = X_1 \times X_2 \times \cdots \times X_{T_0}$. For each j, we classify $w_{u,i}$ into $O(\log W)$ groups for each coordinate $i = 1, 2, \cdots, d$, and partition $u \in S_j$ into a total of $O((\log W)^d)$ groups. We then enumerate the number of elements in each group and select arbitrary elements in the groups. We can show the number of such partial solutions is bounded by a polynomial (Lemma 5.5). For each partial solution $(D_1, D_2, \cdots D_{T_0}) \in X$, we use it as part of the input the rounding algorithm (Algorithm 1).

Since the rounding algorithm is the same as in the previous section, we only describe the partial enumeration procedure in details. The details are as follows:

1. For each dimension i, find the minimum positive weight $\gamma_{j,i}$ among all items in S_j (if all of them are 0, $\gamma_{j,i}$ can be anything and we do not care about it).

2. Construct a modified vector w'_u for each u in S_j:
   - If $w_{u,i} > 0$, set $w'_{u,i} = \lfloor \log(w_{u,i}/\gamma_{j,i}) \rfloor + 1$.
   - If $w_{u,i} = 0$, set $w'_{u,i} = 0$.

3. We say that two vectors w'_u and w'_v are equal if $w'_{u,i} = w'_{v,i}$ for all $i = 1, 2, \cdots, d$. Let r_j as the number of different vector w'_us for $u \in S_j$.

4. Partition S_j into r_j groups: $S_{j,1}, S_{j,2}, \cdots, S_{j,r_j}$ based on different w'_u. Formally, $u \in S_j$ and $v \in S_j$ belong to the same group if $w'_u = w'_v$.

5. For each vector $c \in \mathbb{Z}^{r_j}$ such that $c_k \le |S_{j,k}|$ for all $1 \le k \le r_j$ and $c_1 + c_2 + \cdots + c_{r_j} \le 2^j$, pick a subset from S_j with exactly $\min\{2c_k, |S_{j,k}|\}$ elements from each partition $S_{j,k}$, and add this subset to X_j. (Recall that $X_j \subseteq 2^{S_j}$ is a set of subsets of S_j.)

6. Finally, the algorithm returns $X = X_1 \times X_2 \times \ldots \times X_{T_0}$ as the set of partial solutions.

The pseudo-code can be found in Algorithm 2.

Lemma 5.5. *If $d = O\left(\sqrt{\frac{\log n}{\log\log W}}\right)$, then Algorithm 2 runs in poly$(n, \log(W))$-time.*

Proof. TOPROVE 6 □

Lemma 5.6. *For any integer j, x and $u_0, u_1, u_2 \in S_{j,x}$, $w_{u_0,i} + w_{u_1,i} \ge w_{u_2,i}$ for all $1 \le i \le d$.*

Proof. TOPROVE 7 □

Lemma 5.7. *If the original problem has a 1-valid solution, then there exists a partial solution $(D_1, D_2, \cdots D_{T_0})$ returned by Algorithm 2 such that there exists an extended solution $(D_1, \cdots D_T)$ satisfying:*

1. *$|D_j| \le 2 \times 2^j$ for $1 \le j \le T_0$, and*

2. *$|D_j| \le 2^j$ for $T_0 < j \le T$.*

Proof. TOPROVE 8 □

Lemma 5.8. *Algorithm 2 is a **Partial Enumeration Algorithm** for 2-LogBgt-KnapCov when $d = O(\sqrt{\frac{\log n}{\log\log W}})$. More specifically, it satisfies the following properties:*

1. *If this problem has a 1-valid solution, at least one of its output partial solutions has a 2-valid extended solution.*

2. *It always runs in polynomial time.*

Algorithm 2: Partial Enumeration Algorithm for d-dimensional Knapsack Cover Problem

Result: a set X that $X \subseteq 2^{S_1} \times \cdots \times 2^{S_{T_0}}$

$X_j \leftarrow \emptyset \forall j = 1, \cdots T_0$;
forall $j = 1$ *to* T_0 **do**
  $\gamma_{j,i} \leftarrow \min_{u \in S_j, w_{u,i} > 0}\{w_{u,i}\}$ for all $1 \leq i \leq T_0$;
  Define vector $w'_u \in \mathbb{R}^d$ for all $u \in S_j$;
  forall $u \in S_j$, $i \in \{1, 2, \cdots d\}$ **do**
    if $w_{u,i} = 0$ **then**
      $w'_{u,i} \leftarrow 0$;
    else
      $w'_{u,i} \leftarrow 1 + \lfloor \log(w_{u,i}/\gamma_{j,i}) \rfloor$;
  Partition S_j into several sets $S_{j,1}, S_{j,2}, \cdots S_{j,r_j}$ based on different w'(For $u \in S_{j,x}, v \in S_{j,y}$, $w'_u = w'_v$ if and only if $x = y$);
  forall *Vector* $c \in \mathbb{Z}^{r_j}_{\geq 0}$ *that* $c_k \leq |S_{j,k}|$ *for all* $1 \leq k \leq r_j$ *and* $c_1 + c_2 + \cdots + c_{r_j} \leq 2^j$ **do**
    Pick a subset $D_j^{(c)} \subseteq S_j$ such that contains exactly $\min\{2c_k, |S_{j,k}|\}$ elements in $S_{j,k}$ for all $1 \leq k \leq r_j$;
    $X_j \leftarrow X_j \cup \{D_j^{(c)}\}$
Return $X = X_1 \times \cdots \times X_{T_0}$;

Proof. TOPROVE 9 ☐

Proof of Theorem 5.4. We combine the new enumeration method and the previous rounding algorithm (Algorithm 1) with $T_0 = \lceil \log d \rceil$.

Recall the output of Algorithm 2 is a set of partial solutions: $X \subseteq 2^{S_1} \times 2^{S_2} \times \cdots \times 2^{S_{T_0}}$. For any $(D_1, D_2, \cdots D_{T_0}) \in X$, we run Algorithm 1 with input $(D_1, \ldots, D_{T_0})$.

- According to Lemma 5.7, if there exists a 1-valid solution, then Algorithm 2 can find a partial solution such that there exists an extended solution D' satisfying
  - $|D' \cap S_j| \leq 2 \cdot 2^j$ for all $1 \leq j \leq T_0$.
  - $|D' \cap S_j| \leq 2^j$ for all $T_0 < j \leq T$.
- If there exists an extended solution D' such that $|D' \cap S_j| \leq 2^j$ for all $T_0 < j \leq T$, then the solution set of (LP-KnapCover) is non-empty, so Algorithm 1 can return a solution.
- According to Lemma 5.3, if Algorithm 1 returns a solution D, then $|D \cap S_j| \leq 2 \cdot 2^j$ for all $T_0 < j \leq T$.
- Based on Lemma 5.5, we know this algorithm is polynomial-time.

Combining the above conclusions, it is clear that there exists a $poly(n, \log W)$-time algorithm that can solve 2-LogBgt-KnapCov for $d = O\left(\left(\sqrt{\frac{\log n}{\log\log W}}\right)\right)$.

☐

6 Interval Cover Problem

In this section, we study the norm minimization for the interval cover problem, which is defined as follows:

Definition 6.1 (Min-norm Interval Cover Problem (MinNorm-IntCov)). Given a set $\mathcal{U}^{\text{int}}$ of intervals and a target interval Γ on the real axis, a feasible solution of this problem is a subset $D \subseteq \mathcal{U}^{\text{int}}$ such that D fully covers the target interval Γ (i.e., $\Gamma \subseteq \bigcup_{I\in D} I$). Suppose $\boldsymbol{v} \in \mathbb{R}_{\geq 0}^{\mathcal{U}^{\text{int}}}$ is a value vector and $f(\cdot)$ is a monotone symmetric norm function. Our goal is to find a feasible subset D such that $f(\boldsymbol{v}[D])$ is minimized. We denote the problem as MinNorm-IntCov.

As the algorithms and proofs of this section are complicated, we just provide our main ideas in this section, and defer the full proofs to Appendix E. In light of Theorem 4.1, we can focus on obtaining a constant-factor approximation algorithm for (c, c_0)-LogBgt-IntCov. The input of a LogBgt-IntCov problem is a tuple $\eta^{\text{int}} = (\mathcal{U}^{\text{int}}; S_1^{\text{int}}, \ldots, S_T^{\text{int}}; \Gamma)$, which is the LogBgt problem with input $(\mathcal{U}^{\text{int}}; S_1^{\text{int}}, \ldots, S_T^{\text{int}}; \mathcal{F}^{\text{int}})$ where the set of feasible solutions is $\mathcal{F}^{\text{int}} = \{D \subseteq \mathcal{U}^{\text{int}} : \Gamma \subseteq \bigcup_{I\in D} I\}$.

In this section, we begin by transforming the interval cover problem into a new problem called the *tree cover* problem. The definitions of both problems are provided in Section 6.1. These transformation results in only a constant-factor loss in the approximation factor (i.e., if a polynomial-time algorithm can solve c-LogBgt-TreeCov for some constant c, then there exists a polynomial-time constant-factor approximation algorithm for LogBgt-IntCov).

Next, we focus on LogBgt-TreeCov. We first employ the **partial enumeration** algorithm, as defined in Section 5, to list partial solutions for the first $T_0 = \lfloor \log\log\log n \rfloor$ sets. The details of this process are provided in Section 6.2. Following partial enumeration, we apply a rounding algorithm to evaluate each partial solution. The entire rounding process is detailed in Section 6.3.

6.1 From Interval Cover to Tree Cover

We first introduce some notations for the tree cover problem. Denote a rooted tree as $G = (V, E, r)$, where (V, E) forms an undirected tree and r is the root. For each node $u \in V$, let $\text{Ch}(u)$ be the set of children of u, and $\text{Des}(u)$ be the set of all descendants of u (including u). It is easy to see that $\text{Des}(u) = \{u\} \cup \text{Des}(\text{Ch}(u))$. For a subset of vertices $P \subseteq V$, we define $\text{Des}(P) = \bigcup_{u\in P} \text{Des}(u)$. We also define $\text{Par}(u)$ as the parent node of u and define $\text{Anc}(u)$ as the set of ancestors of u ($\text{Anc}(r) = \{r\}$, and for any $u \in V \setminus \{r\}$, $\text{Anc}(u) = \{u\} \cup \text{Anc}(\text{Par}(u))$). In addition, we define the set of leaves $\text{Leaf}(G) = \{u \in V : \text{Ch}(u) = \emptyset\}$.

Definition 6.2 (LogBgt Tree Cover Problem (LogBgt-TreeCov)). We are given a tuple $\eta^{\text{tr}} = (\mathcal{U}^{\text{tr}}; S_1^{\text{tr}}, \ldots, S_T^{\text{tr}}; G)$, where $G = (V, E, r)$ is a rooted tree and $\mathcal{U}^{\text{tr}} = V \setminus \{r\}$. $T = \lceil \log n \rceil$, and $S_1^{\text{tr}}, S_2^{\text{tr}}, \cdots S_T^{\text{tr}}$ is a partition of $V \setminus \{r\}$ (and S_i^{tr} is called the ith group), and r is not in any group. The partition $S_1^{\text{tr}}, S_2^{\text{tr}}, \cdots S_T^{\text{tr}}$ satisfies the following property: For any node $u \in S_i^{\text{tr}}$ and an arbitrary child v of u, v belongs to group S_j^{tr} with $j > i$. For each $u \in V \setminus \{r\}$, we define $\text{Id}(u) = j$ if $u \in S_j^{\text{tr}}$. In particular, we denote $\text{Id}(r) = 0$. So $\text{Id}(u) > \text{Id}(\text{Par}(u))$ for all $u \in \mathcal{U}^{\text{tr}}$.

A feasible solution for the tree cover problem is a subset $D \subseteq \mathcal{U}^{\text{tr}}$ such that the descendants of D covers all leaves. Formally, the feasible set is defined as

$$\mathcal{F}^{\text{tr}} = \{D \subseteq \mathcal{U}^{\text{tr}} : \text{Leaf}(G) \subseteq \text{Des}(D) = \bigcup\nolimits_{u\in D} \text{Des}(u)\}.$$

We prove the following theorem to reduce the interval cover problem to the tree cover problem. The proof of the theorem can be found in Appendix E.1.

Theorem 6.1. *If there exists a polynomial-time algorithm for the $(c, 8c_0)$-LogBgt-TreeCov problem, then there exists a polynomial-time algorithm for the $(3c, c_0)$-LogBgt-IntCov problem.*

Based on this theorem, we mainly need to deal with the tree cover problem in the following subsections.

6.2 Partial Enumeration Method for Tree Cover Problem

In this subsection, we present a partial enumeration algorithm for the LogBgt-TreeCov problem. Recall that we introduced the concept of partial enumeration in Section 5. For a LogBgt-TreeCov problem with input $(\mathcal{U}^{\mathrm{tr}}; S_1^{\mathrm{tr}}, S_2^{\mathrm{tr}}, \ldots, S_T^{\mathrm{tr}}; G)$, where $G = (V, E, r)$ is a rooted tree, and $n = |\mathcal{U}^{\mathrm{tr}}|$. we set $T_0 = \lfloor \log\log\log n \rfloor$ and perform partial enumeration for the first T_0 sets. The goal is to find a set $X \subseteq 2^{S_1^{\mathrm{tr}}} \times 2^{S_2^{\mathrm{tr}}} \times \cdots \times 2^{S_{T_0}^{\mathrm{tr}}}$ such that there exists a partial solution $(D_1, D_2, \ldots, D_{T_0}) \in X$ satisfying: at least one of the partial solution can be extended to a c-valid solution for some constant c.

In this subsection, we define $\mathrm{Id}(u)$ as the group index of u for $u \in \mathcal{U}^{\mathrm{tr}}$. Now we focus on the LogBgt-TreeCov problem. For each $u \in \mathcal{U}^{\mathrm{tr}}$, define the first type of cost $C_1(u) = \frac{1}{2^{\mathrm{Id}(u)}}$. We then define the second type of cost:

$$C_2(u) = \begin{cases} C_1(u) & \text{if } u \in \mathrm{Leaf}(G) \\ \min\{C_1(u), \sum\limits_{v \in \mathrm{Ch}(u)} C_2(v)\} & \text{if } u \notin \mathrm{Leaf}(G) \end{cases}$$

Intuitively, the cost $C_1(u)$ represents the "cost" of selecting u, as it indicates the proportion of the group that u occupies. Meanwhile, $C_2(u)$ denotes the minimum cost required to cover u using its descendants.

We now present the partial enumeration algorithm. The pseudo-code can be found in Algorithm 4 in Appendix E.2. Here, we briefly describe the main idea of the partial enumeration algorithm.

We employ a depth-first search (DFS) strategy to explore most of the states in the search space. During the search process, we maintain two sets:

- $P \subseteq \mathcal{U}^{\mathrm{tr}}$, representing the set of candidate elements that can still be explored, i.e., $\mathrm{Des}(P)$ contains all uncovered leaves.
- $D \subseteq \mathcal{U}^{\mathrm{tr}}$, storing the elements that have already been selected as part of the partial solution.

Initially, $P = \mathrm{Ch}(r)$ is the child set of the root, and $D = \emptyset$. At each recursive step, we select $u \in P$ with the smallest group index. The recursion proceeds by exploring two possibilities:

1. Adding u to the partial solution, i.e., including u in D and continuing the search.
2. Excluding u from the partial solution, i.e., replacing u with its child nodes while keeping D unchanged. (If u is a leaf, this option is not applicable.)

The search terminates when (P, D) fails to satisfy at least one of the following conditions:

1. $\exists u \in P$, $\mathrm{Id}(u) \le T_0$
2. $\forall u \in D$, $C_2(u) > \frac{1}{\log n}$
3. $(\sum_{v \in D} C_2(v)) + (\sum_{u \in P} C_2(u)) \le 2c_0 T$
4. $\forall 1 \le i \le T_0, \quad |D \cap S_i^{\mathrm{tr}}| \le 2c_0 \cdot 2^i$

The first and fourth conditions are derived from the objective of the Partial Enumeration Method. Regarding the second condition, we observe that for all $u \in \mathcal{U}^{\mathrm{tr}}$, it holds that $\mathrm{Id}(u) \le T_0$ and $C_1(u) > \frac{1}{\log n}$. Furthermore, if $C_2(u) \le \frac{1}{\log n}$, the impact of ignoring u is negligible.

The third condition is based on the property that for any $2c_0$-valid solution $D^* \subseteq \mathcal{U}^{\mathrm{tr}}$, it satisfies:

$$\sum_{u \in D^*} C_2(u) \le 2c_0 T.$$

Due to these conditions, for each group S_j where $1 \le j \le T_0$, we only need to determine at most $2c_0T^2$ items. Consequently, we can establish that our partial enumeration algorithm runs in polynomial time.

We then present the following theorem:

Theorem 6.2. *If the* LogBgt-TreeCov *problem with input η^{tr} has a c_0-valid solution, Algorithm 4 runs in polynomial time and at least one of the output partial solutions has a $2c_0$-valid extended solution.*

The complete proof can be found in Appendix E.2.

6.3 A Rounding Algorithm for Tree Cover Problem

We now focus on the (c, c_0)-LogBgt-TreeCov problem with input $\eta^{\mathrm{tr}} = (\mathcal{U}^{\mathrm{tr}}; S_1^{\mathrm{tr}}, \dots, S_T^{\mathrm{tr}}; G)$, where $G = (V, E, r)$ is a rooted tree. Let $L = \mathrm{Leaf}(G)$ be the set of leaves. Recall that $\mathrm{Anc}(u)$ represents the set of ancestors of node u. For sets $\mathcal{V}, \mathcal{L} \subseteq \mathcal{U}^{\mathrm{tr}}$ and $c \ge 1$, we express the formulation of the linear program as follows:

$$\begin{aligned} \min \quad & 0 \\ s.t. \quad & \sum_{v \in \mathrm{Anc}(u) \cap \mathcal{V}} x_v = 1 \quad \forall u \in \mathcal{L} \\ & \sum_{v \in S_i^{\mathrm{tr}} \cap \mathcal{V}} x_v \le c \cdot 2^i \quad \forall T_0 + 1 \le i \le T \\ & x_v \ge 0 \quad \forall v \in \mathcal{V} \end{aligned} \tag{LP-Tree-Cover(c,$\mathcal{V}$,$\mathcal{L}$)}$$

We call $\sum_{v \in S_i^{\mathrm{tr}} \cap \mathcal{V}} x_v \le c \cdot 2^i$ *cardinality constraints*, and call $\sum_{v \in \mathrm{Anc}(u) \cap \mathcal{V}} x_v = 1$ *feasibility constraints*. Recall that $T_0 = \lfloor \log\log\log n \rfloor$. Also, define $T_1 = \lfloor \log\log n \rfloor$.

The algorithm is as follows, and the pseudocode is Algorithm 5 in Appendix E:

1. Check if LP-Tree-Cover($2c_0$, V_0, L_0) has a feasible solution. If so, obtain an extreme point x^*. Otherwise, confirm that there is no such integral solution.

2. Remove the leaves u with $x_u^* = 0$, and delete all the descendants of their parents. Then $\mathrm{Par}(u)$ becomes a leaf. Repeat this process until $x_u^* \ne 0$ for each leaf u. Let the modified node set and leaf set be V_1 and L_1, respectively.

3. For $u \in V_1$, attempt to round x_u^*. If $x_u^* \ge 1/2$, round it to 1. If $x_u^* > 0$, and u is not a leaf in $S_{T_1+1}^{\mathrm{tr}} \cup \dots \cup S_T^{\mathrm{tr}}$, also round it to 1. In all other cases, round x_u^* to 0. Let D' be the set of nodes u for which x_u^* was rounded to 1. Note that D' may not cover L_1.

4. Remove all descendants in D', and attempt to choose another set from $S_{T_0+1}^{\mathrm{tr}} \cup \dots \cup S_{T_1}^{\mathrm{tr}}$ to cover all leaves. Formalize this objective as LP-Tree-Cover. Specifically,

   - $V_2 = (V_1 \setminus \mathrm{Des}(D')) \cap \{u \in \mathcal{U}^{\mathrm{tr}} : T_0 + 1 \le \mathrm{Id}(u) \le T_1\}$, and
   - $L_2 = (V_2 \cap L_1) \cup \{u \in V_2 : \exists v \in \mathrm{Ch}(u) \cap (V_1 \setminus V_2), (V_2 \setminus \mathrm{Des}(D')) \cap \mathrm{Des}(v) \cap L_1 \ne \emptyset\}$.

   To understand this, observe that V_2 consists of the nodes in the (T_0+1)th to Tth groups that remain uncovered. The set L_2 includes nodes in V_2 that are either leaves or have at least one uncovered child with a group index greater than T_0 (i.e., at least one descendant leaf remains uncovered).

   Then solve LP-Tree-Cover($2c_0$, V_2, L_2). The fact that this problem must have feasible solutions is proved later, so we do not need to consider the case of no solution.

5. Let x^{**} be an extreme point of LP-Tree-Cover($2c_0$, V_2, L_2). For each $u \in V_2$, round it to 1 if and only if $x_u^{**} > 0$. Let $D'' = \{u \in V_2 : x_u^{**} > 0\}$, then D'' covers L_2.

6. Combine the three parts of the solution. That is, return $\left(\bigcup_{i=1}^{T_0} D_i\right) \cup D' \cup D''$.

We prove the following lemma for Algorithm 5:

Lemma 6.3. *Let $T_0 = \lfloor \log\log\log n \rfloor$. If a partial solution $(D_1, D_2, \ldots, D_{T_0})$ has a $2c_0$-valid extended solution, then Algorithm 5 finds a $(4c_0 + 1)$-valid solution.*

By combining Theorem 6.2 and Lemma 6.3, we establish the following theorem:

Theorem 6.4. *For any $c_0 \geq 1$, there exists a polynomial-time algorithm for $(4c_0 + 1, c_0)$-LogBgt-TreeCov.*

Furthermore, applying Theorem 6.1 and Theorem 6.4, we obtain the following result:

Theorem 6.5. *There exists a polynomial-time algorithm that solves the $(3(32c_0+1), c_0)$-LogBgt-IntCov. Consequently, we obtain a polynomial-time constant-factor approximation algorithm for MinNorm-IntCov.*

The complete proofs are provided in Appendix E.

7 Integrality Gap for Perfect Matching, s-t Path, and s-t Cut

In this section, we argue that it may be challenging to achieve constant approximations for the norm optimization problems for perfect matching, s-t path, and s-t cut just by LP rounding. We show that the natural linear programs have large integrality gaps.

Definition 7.1 (Min-Norm s-t Path Problem (MinNorm-Path))**.** Given a directed graph $G^{\text{path}} = (V^{\text{path}}, \mathcal{U}^{\text{path}})$ (here V^{path} is the set of vertices and $\mathcal{U}^{\text{path}}$ is the set of edges) and nodes $s, t \in V^{\text{path}}$, define the feasible set:

$$\mathcal{F}^{\text{path}} = \{D \subseteq \mathcal{U}^{\text{path}} : D \text{ forms a path from } s \text{ to } t\}.$$

For the MinNorm version, we are also given a monotone symmetric norm f and a value vector $\boldsymbol{v} \in \mathbb{R}_{\geq 0}^{\mathcal{U}^{\text{path}}}$. The goal is to select an s-t path $D \in \mathcal{F}^{\text{path}}$ that minimizes $f(\boldsymbol{v}[D])$.

In light of Theorem 4.1, we can define the LogBgt-Path problem with input tuple $\eta^{\text{path}} = (\mathcal{U}^{\text{path}}; S_1^{\text{path}}, \ldots, S_T^{\text{path}}; G^{\text{path}}; s; t)$, which is the LogBgt problem defined in Definition 4.1 with input $(\mathcal{U}^{\text{path}}; S_1^{\text{path}}, \ldots, S_T^{\text{path}}; \mathcal{F}^{\text{path}})$.

Definition 7.2 (Min-Norm Perfect Matching Problem (MinNorm-PerMat))**.** Given a bipartite graph $G^{\text{pm}} = (L, R, \mathcal{U}^{\text{pm}})$ with $|L| = |R|$, define the feasible set:

$$\mathcal{F}^{\text{pm}} = \{D \subseteq \mathcal{U}^{\text{pm}} : D \text{ forms a perfect matching in } G^{\text{pm}}\}.$$

For the MinNorm version, we are also given a monotone symmetric norm f and a value vector $\boldsymbol{v} \in \mathbb{R}_{\geq 0}^{\mathcal{U}^{\text{pm}}}$. The goal is to select $D \in \mathcal{F}^{\text{pm}}$ that minimizes $f(\boldsymbol{v}[D])$.

We define the LogBgt-PerMat problem with $\eta^{\text{pm}} = (\mathcal{U}^{\text{pm}}; S_1^{\text{pm}}, \ldots, S_T^{\text{pm}}; G^{\text{cut}}; s; t)$ as the LogBgt problem with $(\mathcal{U}^{\text{pm}}; S_1^{\text{pm}}, \ldots, S_T^{\text{pm}}; \mathcal{F}^{\text{pm}})$.

Due to Theorem 4.1, we establish the equivalence between approximating the MinNorm problem and the LogBgt problem. Hence, if we can show that the LogBgt version is hard to approximate, then the same hardness (up to constant factor) also applies to the MinNorm version.

Now, we consider using the linear programming rounding approach to approximate the LogBgt problem with $\eta = (\mathcal{U}; S_1, \ldots, S_T; \mathcal{F})$. Such an algorithm proceeds according to the following pipeline:

- First, we formulate the natural linear program (for $c \geq 1$) as the following:

$$\begin{aligned} \min \quad & 0 & & \\ s.t. \quad & x \text{ satisfies} & & \text{relaxed constraints of } \mathcal{F}, \\ & \sum_{u \in S_i} x_u \leq c \cdot 2^i & & \forall 1 \leq i \leq T, \\ & x_u \geq 0 & & \forall u \in \mathcal{U}. \end{aligned} \tag{LP-LBO-Original(η, c)}$$

- Then, to solve the (c, c_0)-LogBgt problem, we first check whether LP-LBO-Original(η, c_0) has feasible solutions. If feasible, we use a rounding algorithm to find an integral solution for (LP-LBO-Original(η, c)).

Since the factor c in LP-LBO-Original(η, c) determines the approximation factor, we study the following linear program.

$$\begin{aligned} \min \quad & z & & \\ s.t. \quad & x \text{ satisfies} & & \text{relaxed constraints of } \mathcal{F}, \\ & \sum_{u \in S_i} x_u \leq z \cdot 2^i & & \forall 1 \leq i \leq T, \\ & x_u \geq 0 & & \forall u \in \mathcal{U}. \end{aligned} \tag{LP-LBO(η)}$$

Clearly, if LP-LBO-Original(η, c) has a feasible solution, the optimal value of LP-LBO(η) is at most c. Suppose we can round a fractional solution LP-LBO-Original(η, c) to an integral feasible c'-valid solution $\bar{x}$ for some constant c' (i.e., $\bar{x}$ satisfies $\mathcal{F}$ and $\sum_{u \in S_i} \bar{x}_u \leq c' \cdot 2^i \ \forall 1 \leq i \leq T$). Then, we get an integral solution for LP-LBO(η) with objective value c', contradicting the fact that the integrality gap of LP-LBO(η) is $\omega(1)$. Hence, we can conclude that if the integrality gap of LP-LBO(η) is $\omega(1)$, it would be difficult to derive a constant factor approximation algorithm for both LogBgt and MinNorm-version of the problem using the LP LP-LBO-Original(η, c).

7.1 Reduction from Perfect Matching to s-t Path

For an LogBgt perfect matching problem with $\eta^{\text{pm}} = (\mathcal{U}^{\text{pm}}; S_1^{\text{pm}}, \dots, S_T^{\text{pm}}; G^{\text{pm}})$, where $G^{\text{pm}} = (L, R, E)$ is a bipartite graph, we consider the following LP (on the left). The LP on the right is for LogBgt-Path with $\eta^{\text{path}} = (\mathcal{U}^{\text{path}}; S_1^{\text{path}}, \dots, S_T^{\text{path}}; G^{\text{path}}; s; t)$.

$$\begin{aligned} \min \quad & z & & \\ \text{s.t.} \quad & \sum_{i \in L, (i,j) \in E} x_{i,j} = 1 & & \forall j \in R, \\ & \sum_{j \in R, (i,j) \in E} x_{i,j} = 1 & & \forall i \in L, \\ & \sum_{e \in S_i^{\text{pm}}} x_e \leq z \cdot 2^i & & \forall 1 \leq i \leq T, \\ & 0 \leq x_{i,j} \leq 1 & & \forall (i,j) \in \mathcal{U}^{\text{pm}}. \end{aligned}$$

(LP-LBO-PM(η^{pm}))

$$\begin{aligned} \min \quad & z & & \\ \text{s.t.} \quad & \sum_{j \in V^{\text{path}}, (i,j) \in \mathcal{U}^{\text{path}}} x_{i,j} = 0 & & \forall i \in V^{\text{path}} \setminus \{s, t\}, \\ & \sum_{j \in V^{\text{path}}, (s,j) \in \mathcal{U}^{\text{path}}} x_{s,j} = 1 & & \\ & \sum_{i \in V^{\text{path}}, (i,t) \in \mathcal{U}^{\text{path}}} x_{i,t} = 1 & & \\ & \sum_{e \in S_i} x_e \leq z \cdot 2^i & & \forall 1 \leq i \leq T, \\ & 0 \leq x_e \leq 1 & & \forall e \in \mathcal{U}^{\text{path}}. \end{aligned}$$

(LP-LBO-Path(η^{path}))

We have the following theorem showing that LogBgt-Path problem is not harder than LogBgt-PerMat problem.

Theorem 7.1. *For any arbitrary function $\alpha(n) \geq 1$, We have the following conclusions:*

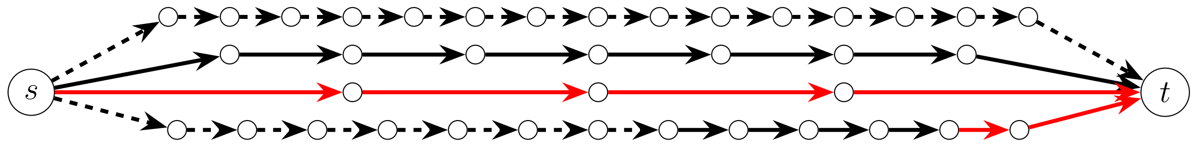


Figure 1: An example for $c = 2, k = 3$. The red solid edges are in S_1^{path}. The black solid edges are in S_2^{path}. The dashed edges are in S_3^{path}.

(a) *If the integrality gap of* (LP-LBO-PM(η^{pm})) *is no more than $\alpha(n)$ for all instances η^{pm} of the* LogBgt-PerMat *problem, then the integrality gap of* (LP-LBO-Path(η^{path})) *is also $O(\alpha(n))$ for all instances of the* LogBgt-Path *problem.*

(b) *If we have a polynomial-time $\alpha(n)$-approximation algorithm for the* MinNorm-PerMat *problem, then we have a polynomial-time $O(\alpha(n))$-approximation algorithm for the* MinNorm-Path *problem.*

Proof. TOPROVE 10 □

7.2 Integrality Gaps for Min-Norm s-t Path and Min-Norm Perfect Matching

Theorem 7.2. *For infinitely many n, there exists an instance η^{cut} of size n such that the integrality gap of* (LP-LBO-Path(η^{path})) *can be $\Omega(\log n)$. Thus, the relaxations of the natural linear programming of both* LogBgt-Path *and* LogBgt-PerMat *have integrality gaps of $\Omega(\log n)$.*

Proof. TOPROVE 11 □

7.3 Integrality Gaps for Min-Norm s-t Cut

Also, we have the result about the integrality gap for s-t cut.

Definition 7.3 (Min-Norm s-t Cut Problem (MinNorm-Cut))**.** Given a directed graph $G^{\text{cut}} = (V^{\text{cut}}, \mathcal{U}^{\text{cut}})$ and nodes $s, t \in V^{\text{cut}}$, define the feasible set:

$$\mathcal{F}^{\text{cut}} = \{D \subseteq \mathcal{U}^{\text{cut}} : \text{There is no path from } s \text{ to } t \text{ in the graph } G' = (V^{\text{cut}}, \mathcal{U}^{\text{cut}} \setminus D)\}.$$

For the MinNorm version, we are also given a monotone symmetric norm f and a value vector $\boldsymbol{v} \in \mathbb{R}_{\geq 0}^{\mathcal{U}^{\text{cut}}}$. The goal is to select an s-t cut $D \in \mathcal{F}^{\text{cut}}$ that minimizes $f(\boldsymbol{v}[D])$.

Similarly, we define the LogBgt-Cut problem with $\eta^{\text{cut}} = (\mathcal{U}^{\text{cut}}; S_1^{\text{cut}}, \ldots, S_T^{\text{cut}}; G^{\text{cut}}; s; t)$ as the LogBgt problem with $(\mathcal{U}^{\text{cut}}; S_1^{\text{cut}}, \ldots, S_T^{\text{cut}}; \mathcal{F}^{\text{cut}})$, where $G^{\text{cut}} = (V^{\text{cut}}, \mathcal{U}^{\text{cut}})$ is a directed graph, we can write the following LP:

$$\begin{aligned}
\min \quad & & z & \\
\text{s.t.} \quad & & 0 \leq p_v \leq 1 & \quad \forall v \in V^{\text{cut}}, \\
& & x_{u,v} \geq p_u - p_v & \quad \forall (u,v) \in \mathcal{U}^{\text{cut}}, \\
& \sum_{(u,v) \in S_i^{\text{cut}}} & x_{u,v} \leq z \cdot 2^i & \quad \forall 1 \leq i \leq T, \\
& & p_s = 0 & \\
& & p_t = 1 & \\
& & 0 \leq x_{u,v} \leq 1 & \quad \forall (u,v) \in \mathcal{U}^{\text{cut}}.
\end{aligned} \tag{LP-LBO-Cut(η^{cut})}$$

Theorem 7.3. *For infinitely many n, there exists an instance η^{cut} of size n such that the integrality gap of* (LP-LBO-Cut(η^{cut})) *can be* $\Omega(\log n)$.

Proof. TOPROVE 12 □

Remark. In the example in the proof, the gap between any feasible subset of $\mathcal{U}' = \{e \in \mathcal{U} : x_e > 0\}$ ($\{x_e\}_{e\in\mathcal{U}}$ is the fractional solution) and the fractional solution is larger than any given constant. Thus any rounding algorithm that deletes zero-value variables (including the rounding algorithm we developed in this paper and the iterative rounding method in [14]) cannot successfully yield a constant-factor approximation.

8 An Algorithm for Min-Norm s-t Path Problem

Recall that we define the MinNorm-Path problem and prove that the natural linear program has a large integrality gap in Section 7. In this section, we provide a factor α approximation algorithm that runs in $n^{O(\log\log n/\alpha)}$ time, for any $9 \le \alpha \le \log\log n$. In particular, this implies an $O(\log\log n)$-factor polynomial-time approximation algorithm and a constant-factor quasi-polynomial $n^{O(\log\log n)}$-time algorithm for MinNorm-Path. Note that this does not contradict Theorem 7.2, since we do not use the LP rounding approach in this section.

In light of Theorem 4.1 with $\varepsilon = 1$, we consider $\frac{\alpha-1}{4}$-LogBgt-Path problem, with input tuple $\eta = (\mathcal{U}; S_1, S_2, \dots, S_T; G; s; t)$, where $n = |\mathcal{U}|$ and $T = \lceil \log n \rceil$, where α is the approximation factor we aim to achieve.

We first provide an overview of our main ideas. A natural approach to solve LogBgt-Path is to employ dynamic programming, in which the states keep track of the number of edges used from each group. However, since we have $T = \lceil \log n \rceil$ groups, the number of states may be as large as $n^{O(T)}$. To resolve this issue, we perform an *approximation dynamic programming*, in which we only approximate the number of edges in each group. In particular, the numbers are rounded to the nearest power of p after each step, for carefully chosen value $p > 1$ (to ensure that we do not lose too much in the rounding). This rounding technique is inspired by a classic approximation algorithm for the subset sum problem [30] Now, the dynamic programming state is a vector that approximates the number of edges used from each group in a path from x to y with at most 2^i edges, where $x, y \in V$ and $0 \le i \le \lceil \log n \rceil$. The dynamic programming process involves storing the number of selected items in each group and rounding them to the nearest power of p at each iteration. However, this method results in a state space of size $(\log_p n)^T = n^{\Omega(\log\log n)}$, which is better than $n^{O(T)}$, but still super-polynomial. To resolve the above issue, we need the second idea, which is to trade off the approximation factor and the running time. In particular, we introduce an integer parameter β, defined as $\beta = \frac{\alpha-1}{4(1+\delta)}$ for some $\delta \in \left[\frac{1}{2}, 2\right]$. We then partition the original T groups into T/β *supergroups*, each containing β groups. This reduces the number of states to $(\log_p n)^{O(T/\beta)}$, but incurs a loss of $O(\beta)$ in the approximation factor. Our main result in this section is the following theorem:

Theorem 8.1. *For any $9 \le \alpha \le \log\log n$, there exists a $n^{O(\log\log n/\alpha)}$-time algorithm (Algorithm 3) for $\frac{\alpha-1}{4}$-LogBgt-Path. Thus we have an approximation approximation which runs in $n^{O(\log\log n/\alpha)}$-time and achieves an approximation factor of α for MinNorm-Path.*

Now, we present the details of our algorithm. Let $K = \lceil T/\beta \rceil$, and $B_i = \min(T, i \cdot \beta)$ for all $0 \le i \le K$.

For $1 \le i \le K$ and $D \subseteq \mathcal{U}$, define $C_i(D) = \sum_{j=B_{i-1}+1}^{B_i} \frac{1}{2^j} |D \cap S_j|$ (specifically, $C_i(\emptyset) = 0$). Furthermore, we define the vector $C(D) = (C_1(D), \dots C_K(D)) \in \mathbb{R}^K$. It is important to notice that:

- If $D \subseteq \mathcal{U}$ is a c-valid solution ($c > 0$), then $C_i(D) \le c\beta$ for all $1 \le i \le K$.

- If $C_i(D) \le c\beta$ for all $1 \le i \le K$, then $D \subseteq \mathcal{U}$ is a $c\beta$-valid solution.

In iteration i ($1 \le i \le \lceil \log n \rceil$), for each pair of vertices $x, y \in V$, we define $Q_{i,x,y}$ as a set of vectors in $\mathbb{R}^K$ that encodes information about paths from x to y containing at most 2^i edges. Specifically, for any path $D \subseteq \mathcal{U}$ from x to y with at most 2^i edges, the set $Q_{i,x,y}$ includes a corresponding vector that approximates $C(D)$.

- Initially, $Q_{0,x,x} = \{C(\emptyset)\}$. For $Q_{0,x,y}$, it is set to $\{C(\{(x,y)\})\}$ if (x,y) is an edge, and $\emptyset$ if (x,y) is not an edge.
- In the i-th iteration ($1 \le i \le \lceil \log n \rceil$), we begin by initializing $Q_{i,x,y} = \emptyset$ for all x, y. Then, for each pair x, y, we enumerate all vertices z and add the sum of $Q_{i-1,x,z}$ and $Q_{i-1,z,y}$ to $Q_{i,x,y}$. Here, the sum of two sets is defined as the set of all pairwise sums of elements from the two sets.
- To reduce the size of $Q_{i,x,y}$, we round the components of these vectors in $Q_{i,x,y}$ to 0 or powers of $p = 1 + \frac{\delta/2}{\log n}$ (recall that $\delta \in [1/2, 2]$).

Algorithm 3: An Algorithm for LogBgt-Path

Output: A $(\alpha - 1)/4$-valid Solution D or "No Solution"

Initially, set all $Q_{i,x,y}$ to $\emptyset$.;
$\forall x \in V$, $Q_{0,x,x} \leftarrow Q_{0,x,x} \cup \{(C(\emptyset)\}$. ;
$\forall$ $(x,y) \in \mathcal{U}$, $Q_{0,x,y} = Q_{0,x,y} \cup \{C(\{(x,y)\})\}$.;
for $i \leftarrow 1$ **to** $\lceil \log n \rceil$ **do**
    forall $x, y, z \in V, u \in Q_{i-1,x,z}, u' \in Q_{i-1,z,y}$ **do**
        Add u and u' up, and round it to the power of p. (Recall that $p = (1 + \frac{\delta/2}{\log n})$).
        Formally, define $w \in \mathbb{R}^K$ by $w_j = p^{\lceil \log_p (u_j + u'_j) \rceil}$ for $1 \le j \le K$ (Specially, if $u_j + u'_j = 0$, then $w_j = 0$.);
        if $\forall 1 \le j \le K, w_j \le p^i \beta$ **then**
            $Q_{i,x,y} \leftarrow Q_{i,x,y} \cup \{w\}$;
if $Q_{\lceil \log n \rceil, s, t} = \emptyset$ **then**
    return "No Solution";
Select $u \in Q_{\lceil \log n \rceil, s, t}$;
return D

Lemma 8.2. *If there exists a 1-valid solution, then Algorithm 3 finds an $(\alpha-1)/4$-valid solution.*

Proof. TOPROVE 13 □

Proof of Theorem 8.1. The values of vectors in any $Q_{i,x,y}$ are rounded to 0 or a power of $p = (1 + \frac{1}{\log n})$. Also, at step i, any value w_j is at most

$$2^i C_j(D) \le 2^i \beta \le 2^i \log \log n,$$

and if it is larger than 0, it is at least $1/2^T$. Therefore, for any $1 \le i \le \lceil \log n \rceil, x, y \in V, 1 \le j \le K$, the number of different values of w_j in $Q_{i,x,y}$ is no more than $2 + \log_p \left(\frac{p^i \log \log n}{1/2^T} \right) = 2 + \frac{T}{\log(1 + (\delta/2)/\log n)} + i + \frac{\log \log \log n}{\log(1+(\delta/2)/\log n)} \le \frac{1}{\delta} \log^2 n$.

Therefore,

$$|Q_{i,x,y}| \le \left(\frac{1}{\delta}\log^2 n\right)^K = \left(\frac{1}{\delta}\log^2 n\right)^{O\left(\frac{\log n}{\beta}\right)}$$
$$= \exp\left(O\left(\frac{\log(1/\delta)}{\beta}\cdot\log\log n\cdot\log n\right)\right) = n^{O\left(\frac{\log\log n}{\alpha}\right)}$$

Therefore, this algorithm is $n^{O(\log\log n/\alpha)}$-time. According to Lemma 8.2 and Theorem 4.1, we find a α-approximation $n^{O(\log\log n/\alpha)}$-time algorithm for MinNorm-Path. $\square$

9 A Bi-criterion Approximation for Matching

In this section, we consider the matching problem. While we do not know how to design a constant factor approximation algorithm for the MinNorm version of perfect matching problem yet, we demonstrate that it is possible to find a nearly perfect matching within a constant approximation factor – a bi-criterion approximation algorithm. In particular, for any given norm, we can find a matching that matches $1-\epsilon$ fraction of nodes and its corresponding norm is at most c (a constant) times the norm of the optimal integral perfect matching. Note that a constant factor approximation algorithm (without relaxing the perfect matching requirement) is impossible using the natural linear program, due to Theorem 7.2.

First, we introduce some notations. We are given a bipartite graph $G=(L,R,E)$, where L is the set of nodes of the first color class, R is the set of node of the other color class, and E is the set of edges. Let $m=|L|=|R|$. We define $\mathcal{U}=E, n=|\mathcal{U}|$. We also study the LogBgt version and we let $S_1, S_2\cdots, S_T$ be the disjoint subsets of $\mathcal{U}$.

We consider two relaxations of the perfect matching requirement.

Definition 9.1 (ϵ-relaxed Matching)**.** Let $0<\epsilon<1$. We define the following problem as ϵ-relaxed matching. Given a bipartite graph $G=(L,R,E)$, a set $S\subseteq E$ is a relaxed matching if in the subgraph $G'=(L,R,S)$, the degree of each vertex is 1 or 2 and the number of vertices with degree 2 is at most $2\epsilon m$ (recall $m=|L|=|R|$).

Definition 9.2 (ϵ-nearly Matching)**.** Let $0<\epsilon<1$. We define the following problem as ϵ-nearly matching. Given a bipartite graph $G=(L,R,E)$, a set $S\subseteq E$ is a nearly matching if it is a matching with at least $(1-\epsilon)m$ edges.

Let $T=\lceil\log n\rceil$. Let $\boldsymbol{d}\in\mathbb{R}^T_{\ge 0}$ be a vector. We define (LP-Perfect-Matching($\boldsymbol{d}$)) as follows:

$$\begin{aligned}
\min \quad & 0 & \\
s.t. \quad & \sum_{i\in L,(i,j)\in E} x_{i,j} = 1 & \forall j\in R \\
& \sum_{j\in R,(i,j)\in E} x_{i,j} = 1 & \forall i\in L \\
& \sum_{e\in S_i} x_e \le d_i & \forall 1\le i\le T \\
& 0\le x_e\le 1 & \forall e\in\mathcal{U}
\end{aligned} \qquad \text{(LP-Perfect-Matching}(\boldsymbol{d}))$$

We call the first and second line of constraints *degree constraints* and the third line *budget constraints*. For any of the problems above, a solution $D\subseteq E$ is called c-valid for a constant c if $|D\cap S_i|\le c\times 2^i$ for $1\le i\le T$.

Let $\boldsymbol{d}^*=(2^1,2^2,\cdots,2^T)$. A 1-valid solution for perfect matching problem is an integral solution for LP-Perfect-Matching($\boldsymbol{d}^*$).

Here is a rounding process for ϵ-relaxed matching.

Lemma 9.1. *For any $\boldsymbol{d} \in \mathbb{R}^T_{\geq 0}$, if LP-Perfect-Matching($\boldsymbol{d}$) has feasible fractional solutions, then there exists an polynomial-time rounding algorithm to find an integral solution of ϵ-relaxed matching, and the integral solution satisfies the following conditions for each i:*

- *If $d_i > 0$, the number of elements chosen from S_i is at most $2d_i + 9/\epsilon$.*
- *If $d_i = 0$, the number of elements chosen from S_i is 0.*

Proof. TOPROVE 14 □

Similarly, given ϵ, if we partially enumerate solutions from S_i for constant number of i (from 1 to some constant k), we can let the right hand side be not larger than $(2+\delta)d_i$. This gives the following theorem:

Theorem 9.2. *If there exists a 1-valid solution for a* LogBgt-PerMat *problem, then there exists an $O(n^{\frac{18}{\delta\epsilon}})$-time algorithm to obtain a $(2+\delta)$-valid solution for the corresponding ϵ-relaxed matching problem for any $\delta, \epsilon < 1$.*

Proof. TOPROVE 15 □

Notice that for the solution we got in Theorem 9.2, as only $2\epsilon m$ number of nodes are of degree 2, we just delete one edge from each of them. We can get a matching with at least $(1-\epsilon)m$ edges (for an vertex in L with no edge left, it corresponds to at least one vertex in R with 2 original edges, and the number of such vertices in R is at most ϵm). By deleting edges, the number of edges in S_i for each i does not increase. So this gives the following theorem from Theorem 9.2:

Theorem 9.3. *If there exists a 1-valid solution for a* LogBgt-PerMat *problem, then there exists an $O(n^{\frac{18}{\delta\epsilon}})$-time algorithm to obtain a $(2+\delta)$-valid solution for the corresponding* LogBgt *ϵ-nearly matching problem for any $\delta, \epsilon < 1$.*

Now we consider the MinNorm version of these problems. By Theorem D.1, we can get:

Theorem 9.4. *Given a* MinNorm-PerMat *problem with a monotone symmetric norm $f(\cdot)$ and a value vector $\boldsymbol{v}$, let D^* be an optimal solution for this problem. For any constants $\epsilon, \delta < 1$, there exists a polynomial-time algorithm to obtain a ϵ-relaxed matching D such that $\frac{f(\boldsymbol{v}[D])}{f(\boldsymbol{v}[D^*])} \leq (8+\delta)$.*

Theorem 9.5. *Given a* MinNorm-PerMat *problem with a monotone symmetric norm $f(\cdot)$ and a value vector $\boldsymbol{v}$, let D^* be an optimal solution for this problem. For any constants $\epsilon, \delta < 1$, there exists a polynomial-time algorithm to obtain a ϵ-nearly matching D such that $\frac{f(\boldsymbol{v}[D])}{f(\boldsymbol{v}[D^*])} \leq (8+\delta)$.*

10 Concluding Remarks

In this paper, we propose a general formulation for general norm minimization in combinatorial optimization. Our formulation captures a broad class of combinatorial structures, encompassing various fundamental problems in discrete optimization. Via a reduction of the norm minimization problem to a multi-criteria optimization problem with logarithmic budget constraints, we develop constant-factor approximation algorithms for multiple important covering problems, such as interval cover, multi-dimensional knapsack cover, and set cover (with logarithmic approximation factors). We also provide a bi-criteria approximation algorithm for min-norm perfect matching, and an $O(\log\log n)$-approximation algorithm for the min-norm s-t path problem, via a nontrivial approximate dynamic programming approach.

Our results open several intriguing directions for future research. First, one can explore other combinatorial optimization problems, such as Steiner trees and other network design problems, within our general framework. Additionally, our formulation could be extended to encompass the min-norm load balancing problem studied in [14] (where job processing times are first summed into machine loads before applying a norm), and even the generalized load balancing [20] and cascaded norm clustering problems [1, 18] (which allow for two levels of cost aggregation via norms). Second, obtaining a nontrivial true approximation algorithm for perfect matching – rather than a bi-criterion approximation – remains an important open problem. Third, it is an important open problem whether a polynomial-time constant-factor approximation exists for the min-norm s-t path problem. Lastly, it would be interesting to study other general objective functions beyond symmetric monotone norms and submodular functions (such as general subadditive functions [28] and those studied in [36]).

References

[1] Fateme Abbasi, Sandip Banerjee, Jarosław Byrka, Parinya Chalermsook, Ameet Gadekar, Kamyar Khodamoradi, Dániel Marx, Roohani Sharma, and Joachim Spoerhase. Parameterized approximation schemes for clustering with general norm objectives. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1377–1399. IEEE, 2023.

[2] Ali Aouad and Danny Segev. The ordered k-median problem: surrogate models and approximation algorithms. *Math. Program.*, 177(1-2):55–83, 2019.

[3] Nikhil Ayyadevara, Nikhil Bansal, and Milind Prabhu. On minimizing generalized makespan on unrelated machines. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, page 21, 2023.

[4] Yossi Azar and Amir Epstein. Convex programming for scheduling unrelated parallel machines. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 331–337, 2005.

[5] Jatin Batra, Syamantak Das, and Agastya Vibhuti Jha. Tight approximation algorithms for ordered covering. In *Algorithms and Data Structures Symposium*, pages 120–135. Springer, 2023.

[6] André Berger, Vincenzo Bonifaci, Fabrizio Grandoni, and Guido Schäfer. Budgeted matching and budgeted matroid intersection via the gasoline puzzle. *Mathematical Programming*, 128:355–372, 2011.

[7] Dimitris Bertsimas and John Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, 1st edition, 1997. ISBN 1886529191.

[8] Vladimir Braverman, Shaofeng H-C Jiang, Robert Krauthgamer, and Xuan Wu. Coresets for ordered weighted clustering. In *International Conference on Machine Learning*, pages 744–753. PMLR, 2019.

[9] Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1433–1452. SIAM, 2014.

[10] Jaroslaw Byrka, Krzysztof Sornat, and Joachim Spoerhase. Constant-factor approximation for ordered k-median. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 620–631, 2018.

[11] Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a submodular set function subject to a matroid constraint. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 182–196. Springer, 2007.

[12] Paolo M. Camerini, Giulia Galbiati, and Francesco Maffioli. Random pseudo-polynomial algorithms for exact matroid problems. *Journal of Algorithms*, 13(2):258–273, 1992.

[13] Deeparnab Chakrabarty and Chaitanya Swamy. Interpolating between k-median and k-center: Approximation algorithms for ordered k-median. In *45th International Colloquium on Automata, Languages, and Programming*, volume 107 of *LIPIcs*, pages 29:1–29:14, 2018.

[14] Deeparnab Chakrabarty and Chaitanya Swamy. Approximation algorithms for minimum norm and ordered optimization problems. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 126–137, 2019.

[15] Deeparnab Chakrabarty and Chaitanya Swamy. Simpler and better algorithms for minimum-norm load balancing. In *27th Annual European Symposium on Algorithms*, volume 144 of *LIPIcs*, pages 27:1–27:12, 2019.

[16] Chandra Chekuri and Alina Ene. Submodular cost allocation problem and applications. In *Automata, Languages and Programming - 38th International Colloquium*, volume 6755 of *Lecture Notes in Computer Science*, pages 354–366, 2011.

[17] Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Multi-budgeted matchings and matroid intersection via dependent rounding. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1080–1097. SIAM, 2011.

[18] Eden Chlamtáč, Yury Makarychev, and Ali Vakilian. Approximating fair clustering with cascaded norm objectives. In *Proceedings of the 2022 annual ACM-SIAM symposium on discrete algorithms (SODA)*, pages 2664–2683. SIAM, 2022.

[19] Shichuan Deng and Qianfan Zhang. Ordered k-median with outliers and fault-tolerance. *CoRR*, abs/2011.04289, 2020. URL https://arxiv.org/abs/2011.04289.

[20] Shichuan Deng, Jian Li, and Yuval Rabani. Generalized unrelated machine scheduling problem. *CoRR*, abs/2202.06292, 2022. URL https://arxiv.org/abs/2202.06292.

[21] Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Symposium on Theory of Computing*, pages 624–633, 2014.

[22] Nicolas El Maalouly. Exact matching: Algorithms and related problems. In *40th International Symposium on Theoretical Aspects of Computer Science*, 2023.

[23] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[24] Daniel Golovin, Anupam Gupta, Amit Kumar, and Kanat Tangwongsan. All-norms and all-l_p-norms approximation algorithms. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (2008)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2008.

[25] Fabrizio Grandoni and Rico Zenklusen. Approximation schemes for multi-budgeted independence systems. In *European Symposium on Algorithms*, pages 536–548. Springer, 2010.

[26] Fabrizio Grandoni, Ramamoorthi Ravi, and Mohit Singh. Iterative rounding for multi-objective optimization problems. In *European Symposium on Algorithms*, pages 95–106. Springer, 2009.

[27] Fabrizio Grandoni, R. Ravi, Mohit Singh, and Rico Zenklusen. New approaches to multi-objective optimization. *Math. Program.*, 146(1-2):525–554, 2014.

[28] Anupam Gupta, Viswanath Nagarajan, and Sahil Singla. Adaptivity gaps for stochastic probing: submodular and xos functions. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, page 1688–1702, USA, 2017. Society for Industrial and Applied Mathematics.

[29] G. H. Hardy, J. E. Littlewood, and G. Pólya. *Inequalities.* Cambridge University Press, 1934. ISBN 978-0-521-35880-4.

[30] Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, October 1975. ISSN 0004-5411. doi: 10.1145/321906.321909. URL https://doi.org/10.1145/321906.321909.

[31] Sharat Ibrahimpur and Chaitanya Swamy. Approximation algorithms for stochastic minimum-norm combinatorial optimization. In *61st IEEE Annual Symposium on Foundations of Computer Science*, pages 966–977, 2020.

[32] Sharat Ibrahimpur and Chaitanya Swamy. Minimum-norm load balancing is (almost) as easy as minimizing makespan. In *48th International Colloquium on Automata, Languages, and Programming*, volume 198 of *LIPIcs*, pages 81:1–81:20, 2021.

[33] Thomas Kesselheim, Marco Molinaro, and Sahil Singla. Supermodular approximation of norms and applications. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 1841–1852, 2024.

[34] Andreas Krause and Daniel Golovin. Submodular function maximization. *Tractability*, 3 (71-104):3, 2014.

[35] Jon Lee, Maxim Sviridenko, and Jan Vondrák. Submodular maximization over multiple matroids via generalized exchange properties. *Mathematics of Operations Research*, 35(4): 795–806, 2010.

[36] Jian Li and Samir Khuller. Generalized machine activation problems. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 80–94, 2011.

[37] André Linhares, Neil Olver, Chaitanya Swamy, and Rico Zenklusen. Approximate multi-matroid intersection via iterative refinement. *Mathematical Programming*, 183:397–418, 2020.

[38] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2005.

[39] Kalen Patton, Matteo Russo, and Sahil Singla. Submodular norms with applications to online facility location and stochastic probing. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2023).* Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.

[40] Ram Ravi and Michel X Goemans. The constrained minimum spanning tree problem. In *Algorithm Theory—SWAT'96: 5th Scandinavian Workshop on Algorithm Theory Reykjavík, Iceland, July 3–5, 1996 Proceedings 5*, pages 66–75. Springer, 1996.

[41] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

A Additional Preliminaries

We need the following version of Chernoff bounds.

Lemma A.1. (Chernoff bounds (see, e.g., [38])). *Let $X_1, \dots, X_n$ be independent Bernoulli variables with $\mathbb{E}[X_i] = p_i$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathbb{E}[X] = \sum_{i=1}^{n} p_i$. For $\nu \geq 6\mu$, one has $\Pr[X \geq \nu] \leq 2^{-\nu}$.*

B Top-ℓ Norm Optimization

Minimization Problems. We first consider combinatorial optimization problems for which minimizing a linear objective is poly-time solvable. We use $\mathfrak{A}$ to denote the ordinary combinatorial optimization problem under consideration. In particular, we assume that there is a poly-time algorithm Alg that solves the original min-sum optimization problem $\mathfrak{A}$: $\min_{S \in \mathcal{F}} \boldsymbol{v}(S)$, where we write $\boldsymbol{v}(S) = \sum_{e \in S} v_e$. The idea of the following theorem is not new and has been used in previous papers for specific combinatorial problems (e.g., [10, 22]). However, to the best of our knowledge, the exact formulation of the following theorem in this general form has not appeared before and we think it is worth recording.

Theorem B.1. *If the minimization problem $\mathfrak{A}$ can be solved in poly-time, there is a poly-time algorithm for solving the top-ℓ* minimization *problem optimally.*

Proof. TOPROVE 16 □

We can generalize the above theorem to approximation algorithms.

Theorem B.2. *If there is a poly-time approximation algorithm for the minimization problem $\mathfrak{A}$ (with approximation factor $\alpha \geq 1$), there is a poly-time factor α approximation algorithm for the top-ℓ* minimization *problem optimally.*

Proof. TOPROVE 17 □

Remark: The above theorem does not imply a constant approximation for the top-ℓ minimization of k-median. The reason is that the modified edge weights do not satisfy triangle inequality any more, which is required for known constant approximation algorithms of k-median. In fact, achieving a constant approximation factor for the top-ℓ minimization of k-median requires significant effort, as done in [10, 13, 14].

Maximization Problems. The above approach does not directly work for the maximization problem. We adopt the same idea in [22]. The exact version of problem $\mathfrak{A}$ asks the question whether there is a feasible solution of with weight exactly equal to a given integer K. We say an algorithm runs in pseudopolynomial time for the exact version of $\mathfrak{A}$ if the running time is polynomial in n and K (the algorithm can detect if there is no feasible solution with weight exactly K and return "no solution"). For many combinatorial problems, a pseudopolynomial algorithm for the exact version is known. Examples include shortest path, spanning tree, matching and knapsack.

Theorem B.3. *If the exact version of problem $\mathfrak{A}$ admits a pseudopolynomial time algorithm, we can solve the top-ℓ* maximization *problem for $\mathfrak{A}$ if all weights are polynomially bounded nonnegative integers.*

Proof. TOPROVE 18 □

C Ordered Norm Optimization

In this section, we consider the ordered norm minimization problem. Recall that $\textsc{Ord}_{\boldsymbol{w}}(\boldsymbol{v}) = \boldsymbol{w}^\top \boldsymbol{v}^\downarrow$. We use $\mathfrak{A}$ to denote the original combinatorial optimization problem under consideration. In particular, we assume that there is a poly-time algorithm Alg that solves the minimization problem $\mathfrak{A}$: $\min_{S\in\mathcal{F}} \boldsymbol{v}(S)$, where we write $\boldsymbol{v}(S) = \sum_{e\in S} v_e$.

In this section, we fix $\delta, \varepsilon > 0$. Define the set $\mathrm{POS} = \{\min\{\lceil (1+\delta)^s \rceil, n\} : s \geq 0\}$. For each $\ell \in \mathrm{POS}$, we define $\mathrm{next}(\ell)$ to be the smallest element in POS that is larger than ℓ for $\ell < n$. And we define $\mathrm{next}(n) = n+1$. We sparsify the weight vector $\boldsymbol{v}$ so that $\widetilde{w}_i = w_\ell$ where ℓ is the smallest element in POS such that $\ell \geq i$. Also, let $\widetilde{w}_{n+1} = 0$. For any $\boldsymbol{t} \in \mathbb{R}^{\mathrm{POS}}$, define

$$h_{\boldsymbol{t}}(\widetilde{\boldsymbol{w}}; a) := \sum_{\ell\in\mathrm{POS}} (\widetilde{w}_\ell - \widetilde{w}_{\mathrm{next}(\ell)})(a - t_\ell)^+.$$

And

$$\mathrm{prox}_{\boldsymbol{t}}(\widetilde{\boldsymbol{w}}; \boldsymbol{v}) = \sum_{\ell\in\mathrm{POS}} (\widetilde{w}_\ell - \widetilde{w}_{\mathrm{next}(\ell)})\ell \cdot t_\ell + \sum_{i=1}^{n} h_{\boldsymbol{t}}(\widetilde{\boldsymbol{w}}; v_i).$$

Let $\boldsymbol{o} = \boldsymbol{v}[S^\star]$ be the vector for the optimal solution.

We need some lemmas from prior work [14]. In particular, they are Lemma 4.2, 6.9, 6.5 and 6.8 in [14].

Lemma C.1. *For any $\boldsymbol{v} \in \mathbb{R}^n_+$,*

$$\textsc{Ord}_{\widetilde{\boldsymbol{w}}}(\boldsymbol{v}) \leq \textsc{Ord}_{\boldsymbol{w}}(\boldsymbol{v}) \leq (1+\delta)\textsc{Ord}_{\widetilde{\boldsymbol{w}}}(\boldsymbol{v}).$$

Lemma C.2. *Suppose we can obtain in polynomial time a set Q of polynomial size that contains a real number in $[o^\downarrow_1, (1+\varepsilon)o^\downarrow_1]$. Then, in time $O\left(|Q| \max\{(n/\varepsilon)^{O(1/\varepsilon)}, n^{O(1/\delta)}\}\right)$, we can obtain a set T of polynomial number of vectors in $\mathbb{R}^{POS}$ which contains a threshold vector (denoted by $\boldsymbol{t}^*$) satisfying: $o^\downarrow_\ell \leq t^*_\ell \leq (1+\varepsilon)o^\downarrow_\ell$ if $o^\downarrow_\ell \geq \varepsilon o^\downarrow_1/n$ and $t^*_\ell = 0$ otherwise. Moreover, for all $\ell \in POS$, t^*_ℓ is either 0 or a power of $1+\varepsilon$.*

Lemma C.3. *For any $\boldsymbol{v} \in \mathbb{R}^n_+, \boldsymbol{t} \in \mathbb{R}^{POS}$, the following inequality holds:*

$$\textsc{Ord}_{\widetilde{\boldsymbol{w}}}(\boldsymbol{v}) \leq \mathrm{prox}_{\boldsymbol{t}}(\widetilde{\boldsymbol{w}}; \boldsymbol{v}).$$

Lemma C.4. *Let $\boldsymbol{t} \in \mathbb{R}^{POS}$ be a valid threshold vector (i.e., $t_\ell \geq t_{next(\ell)}$ for all $\ell \in POS$) satisfying $v^\downarrow_\ell \leq t_\ell \leq (1+\varepsilon)v^\downarrow_\ell$ for $v^\downarrow_\ell \geq \varepsilon v^\downarrow_1/n$, and $t_\ell = 0$ otherwise. Then, for any value vector $\boldsymbol{v} \in \mathbb{R}^n_+$, we have that*

$$\mathrm{prox}_{\boldsymbol{t}}(\widetilde{\boldsymbol{w}}; \boldsymbol{v}) \leq (1+2\varepsilon)\textsc{Ord}_{\widetilde{\boldsymbol{w}}}(\boldsymbol{v}).$$

Theorem C.5. *If the original minimization problem $\mathfrak{A}$ can be solved in poly-time, there is a poly-time factor $(1+\epsilon)$ approximation algorithm for the ordered* minimization *problem, for any positive constant ϵ.*

Proof. TOPROVE 19 □

We can generalize the above theorem to approximation algorithms.

Theorem C.6. *If there is a poly-time approximation algorithm for the minimization problem $\mathfrak{A}$ (with approximation factor $\alpha \geq 1$), there is a poly-time factor $\alpha(1+\epsilon)$ approximation algorithm for the ordered* minimization *problem optimally for any positive constant ϵ.*

Proof. TOPROVE 20 □

D Another form of Theorem 4.1

In Section 9, we need to use the following theorem, whose proof is very similar to Theorem 4.1.

Theorem D.1. *Assume that we have two* MinNorm *instances with the same $\mathcal{U}$, the same value vector $\boldsymbol{v}$ and same norm f, but differ in their feasible sets, denoted by $\mathcal{F}_1$ and $\mathcal{F}_2$, where $\mathcal{F}_1, \mathcal{F}_2 \neq \emptyset$. Let c be a positive integer and $\epsilon > 0$. Consider the* LogBgt *versions of these two problems. Suppose that for any disjoint subsets $S_1, S_2, \cdots, S_T$ of $\mathcal{U}$, if there exists a 1-valid solution in $\mathcal{F}_1$, then we can compute a c-valid solution in $\mathcal{F}_2$ (with the same $S_1, S_2, \cdots, S_T$) in polynomial time. Under this assumption, we can find a set $S \in \mathcal{F}_2$ such that $f(\boldsymbol{v}[S]) \leq (4c + \epsilon) \cdot f(\boldsymbol{v}[S^*])$, where S^* is the optimal solution in $\mathcal{F}_1$ that minimizes the norm f in polynomial time.*

Proof of Theorem D.1. We enumerate all possible threshold vectors $\boldsymbol{t} \in \mathbb{R}^{\mathrm{POS}}$ as defined in Lemma 4.2 (for $\mathcal{F}_1$). Suppose $\boldsymbol{t}$ is a valid guess for the optimal solution of $\mathcal{F}_1$. We can also assume that we have an exactly correct guess of $o_1^{\downarrow}$ (o_1 is also for $\mathcal{F}_1$). We construct sets $S_1, \cdots, S_T$ in the following way. For each element $e \in \mathcal{U}$, if its value v_e is larger than $\boldsymbol{t}_1$, we do not add it to any set. If it is at most $\max\{\boldsymbol{t}_n, \varepsilon o_1^{\downarrow}/n\}$, we add it to S_T. For any other element, if it is at most $\boldsymbol{t}_\ell$ and larger than $\boldsymbol{t}_{\mathrm{next}(\ell)}$, where $\ell = 2^i$, we add it to S_{i+1}, for $0 \leq i \leq T-1$. Now, consider the optimal solution $\boldsymbol{o}$ for MinNormfor $\mathcal{F}_1$ (if there are multiple optimal solutions, consider the one corresponding to the valid guess $\boldsymbol{t}$). For $1 \leq i \leq T-1$, consider how many elements in $\boldsymbol{o}$ that are added in S_i. By definition, elements in S_i are with value larger than $\boldsymbol{t}_{2^i}$. Also, the values are larger than $\varepsilon o_1^{\downarrow}/n$. So for $\ell = 2^i$, $o_\ell \leq \boldsymbol{t}_\ell$ by definition of valid guess. Thus, there are at most 2^i elements in $\boldsymbol{o}$ that are added in S_i. This also holds for $i = T$ as there are only n total number of elements.

Assume that we can get a solution $S \in \mathcal{F}_2$ such that there are at most $c \cdot 2^i$ elements in each S_i. We partition S into A, B such that the elements in A have value less than $\varepsilon o_1/n$ and elements in B have value at least $\varepsilon o_1/n$. We need this partition because of the condition $o_\ell^{\downarrow} \geq \varepsilon o_1^{\downarrow}/n$ in Lemma 4.2. Note that A, B are not needed in the algorithm. They are only useful in the analysis. So we have

$$f(\boldsymbol{v}[S]) \leq f(A) + f(B) \leq n \cdot f(\varepsilon o_1/n) + 4cf(g(\boldsymbol{t})) \leq \varepsilon f(\boldsymbol{o}) + 4c(1+\varepsilon)f(\boldsymbol{o}) \leq 4c(1+2\varepsilon)f(\boldsymbol{o})$$

where the first inequality following from the triangle inequality of the norm, the second from the definition of A and Lemma 4.4 and the third from Lemma 4.3. Therefore, the theorem follows. $\square$

E Algorithms and Proofs Omitted in Section 6

E.1 Proofs Omitted in Section 6.1

We call an input tuple $\eta^{\mathrm{int}} = (\mathcal{U}^{\mathrm{int}}; S_1^{\mathrm{int}}, \ldots, S_T^{\mathrm{int}}; \Gamma)$ *locally disjoint* if $\forall 1 \leq j \leq T$, $I, J \in S_j^{\mathrm{int}}, I \neq J$, $I \cap J = \emptyset$. We say that η^{int} satisfies the *laminar family* constraint if:

- $\mathcal{U}^{\mathrm{int}}$ forms a laminar family (i.e. each two intervals either have no common interior points or one contain the other), and
- $\forall I, J \in \mathcal{U}^{\mathrm{int}}$, if $I \subseteq J$, then the group index of I is not less than that of J.

Now we focus on the LogBgt-IntCov problem with input $\eta^{\mathrm{int}} = (\mathcal{U}^{\mathrm{int}}; S_1^{\mathrm{int}}, S_2^{\mathrm{int}}, \ldots, S_T^{\mathrm{int}}; \Gamma)$. We want to prove that it is equivalent to LogBgt-TreeCov up to a constant approximation factor (see Theorem 6.1). We first change the input into η^{ld}, which is a locally disjoint input. Then we change it into η^{lam}, which is a laminar family input. At last, we change it to the input of LogBgt-TreeCov.

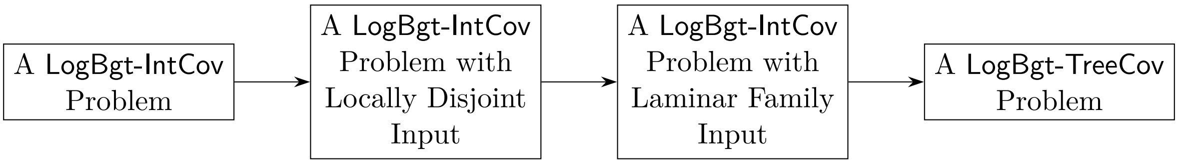


Figure 2: Outline of Appendix E.1.

First, We use the following process to construct a new LogBgt-IntCov problem with input $\eta^{\mathrm{ld}} = (\mathcal{U}^{\mathrm{ld}}; S_1^{\mathrm{ld}}, S_2^{\mathrm{ld}}, \ldots, S_T^{\mathrm{ld}}; \Gamma)$, where $\mathcal{U}^{\mathrm{ld}} = \bigcup_{1\le i\le T} S_i^{\mathrm{ld}}$, and for all $1 \le j \le T$, S_j^{ld} is constructed by the following process (the process for different j are independent):

We use two interval sets, S_{tmp}^{int} and S_{tmp}^{ld}, as temporary variables for this conversion process. Initially, we set $S_{tmp}^{\mathrm{int}} \leftarrow S_j^{\mathrm{int}}$ and $S_{tmp}^{\mathrm{ld}} \leftarrow \emptyset$. Then we start a loop. At each iteration, we do the following steps:

1. If $S_{tmp}^{\mathrm{int}} = \emptyset$, we set $S_j^{\mathrm{ld}} = S_{tmp}^{\mathrm{ld}}$ and finish the loop.

2. Choose the interval I in S_{tmp}^{int} whose left endpoint is the leftmost, and among those, select the one whose right endpoint is the rightmost.

3. Add the chosen interval I to S_{tmp}^{ld}. i.e., $S_{tmp}^{\mathrm{ld}} \leftarrow S_{tmp}^{\mathrm{ld}} \cup \{I\}$.

4. Remove the intersection part from S_{tmp}^{int}. i.e., $S_{tmp}^{\mathrm{int}} \leftarrow \{J \setminus I : J \in S_{tmp}^{\mathrm{int}}, J \not\subseteq I\}$.

Lemma E.1. *The LogBgt-IntCov problem with input $\eta^{\mathrm{ld}} = (\mathcal{U}^{\mathrm{ld}}; S_1^{\mathrm{ld}}, \ldots, S_T^{\mathrm{ld}}; \Gamma)$, constructed from η^{int} by the above process, satisfies the following properties:*

(a) *η^{ld} is locally disjoint. i.e., $\forall 1 \le j \le T$, $I, J \in S_j^{\mathrm{ld}}$, $I \cap J = \emptyset$.*

(b) *For any $c \ge 1$, if there exists a c-valid solution for the problem with input η^{ld}, then there exists a c-valid solution for the problem with input η^{int}.*

(c) *For any $c_0 \ge 1$, if there exists a c_0-valid solution for the problem with input η^{int}, then there exists a $2c_0$-valid solution for the problem with input η^{ld}.*

Proof. TOPROVE 21 □

Next, we use the new input tuple η^{ld} to construct another LogBgt-IntCov problem with input tuple $\eta^{\mathrm{lam}} = (\mathcal{U}^{\mathrm{lam}}; S_1^{\mathrm{lam}}, S_2^{\mathrm{lam}}, \ldots, S_T^{\mathrm{lam}}; \Gamma)$, where $\mathcal{U}^{\mathrm{lam}} = \bigcup_{1\le j\le T} S_j^{\mathrm{lam}}$, and $S_1^{\mathrm{lam}}, S_2^{\mathrm{lam}}, \ldots, S_T^{\mathrm{lam}}$ are constructed by the following process:

We use a series of interval sets $S_1', S_2', \ldots, S_T'$ as temporary variables for this conversion process. Initially, we set $S_j' \leftarrow \emptyset$ for all $1 \le j \le T$. Then, we begin a loop to sequentially enumerate elements from $S_1^{\mathrm{ld}}, S_2^{\mathrm{ld}}, \ldots, S_T^{\mathrm{ld}}$. In the ith loop for $1 \le i \le T$, we sequentially consider all intervals in S_i^{ld}. This means that when we enumerate $I \in S_{k+1}^{\mathrm{ld}}$, all intervals $J \in S_j^{\mathrm{ld}}$ for $j \le k$ have already been enumerated. During this loop, we ensure that $\mathcal{U}' = \bigcup_{1\le j\le T} S_j'$ always forms a laminar family. When we reach $I \in S_{k+1}^{\mathrm{ld}}$, we perform the following steps:

1. For all $t \le k$ and $J \in S_t'$, if $J \subseteq I$, we delete J from S_t' (i.e., $S_t' \leftarrow S_t' \setminus \{J\}$). We say that interval I *removes* interval J at this iteration.

2. We select intervals from $S_1', S_2', \ldots, S_k'$ such that the selected interval J has exactly one endpoint in I (i.e., $J \cap I \neq \emptyset$, $I \setminus J \neq \emptyset$, and $J \setminus I \neq \emptyset$, also each of them doesn't only consist of one point). Denote this set as M, and split it into two parts: M_L is the part where the left endpoint is in I, and M_R is the part where the right endpoint is in I.

3. Choose the leftmost left endpoint in M_L. Denote A as the interval from this endpoint to the right endpoint of I (if $M_L = \emptyset$, then $A = \emptyset$). For each $j = 1, 2, \ldots, k$ and $J \in M \cap S'_j$, if $J \in M_L$, we add the interval A (i.e., $S'_j \leftarrow (S'_j \setminus \{J\}) \cup \{J \cup A\}$). If $J \in M_R$, we add the interval $I \setminus A$. (i.e., $S'_j \leftarrow (S'_j \setminus \{J\}) \cup \{J \cup (I \setminus A)\}$). We say that interval I *extends* interval J at this iteration.

4. By the fact that before this iteration $\bigcup_{1 \le j \le T} S'_j$ forms a laminar family, A, B don't have common interior points. So we split I into three parts: A, B, and $(I \setminus A) \setminus B$. Add the non-empty parts to S'_{k+1}.

It is easy to prove that during the process, $\bigcup_{1 \le j \le T} S'_j$ is always a laminar family. So the process can finish. After the loop finishes, we set $S^{\text{lam}}_j \leftarrow S'_j$ for all $1 \le j \le T$.

Lemma E.2. *The* LogBgt-IntCov *problem with input* $\eta^{\text{lam}} = (\mathcal{U}^{\text{lam}}; S^{\text{lam}}_1, \ldots, S^{\text{lam}}_T; \Gamma)$*, constructed from* η^{ld} *by the above process, satisfies the following properties:*

(a) η^{lam} *satisfies the laminar family constraint. i.e., (i)* $\mathcal{U}^{\text{lam}}$ *forms a laminar family, and (ii)* $\forall I, J \in \mathcal{U}^{\text{lam}}$*, if* $I \subseteq J, I \ne J$*, then* $\mathrm{Id}(\eta^{\text{lam}}; I) < \mathrm{Id}(\eta^{\text{lam}}; J)$*.*

(b) *For any* $c \ge 1$*, if there exists a* c*-valid solution for the problem with input* η^{int}*, then there exists a* $3c$*-valid solution for* η^{ld}*.*

(c) *For any* $c_0 \ge 1$*, if there exists a* c_0*-valid solution for the problem with input* η^{ld}*, then there exists a* $4c_0$*-valid solution for* η^{int}*.*

Proof. TOPROVE 22 □

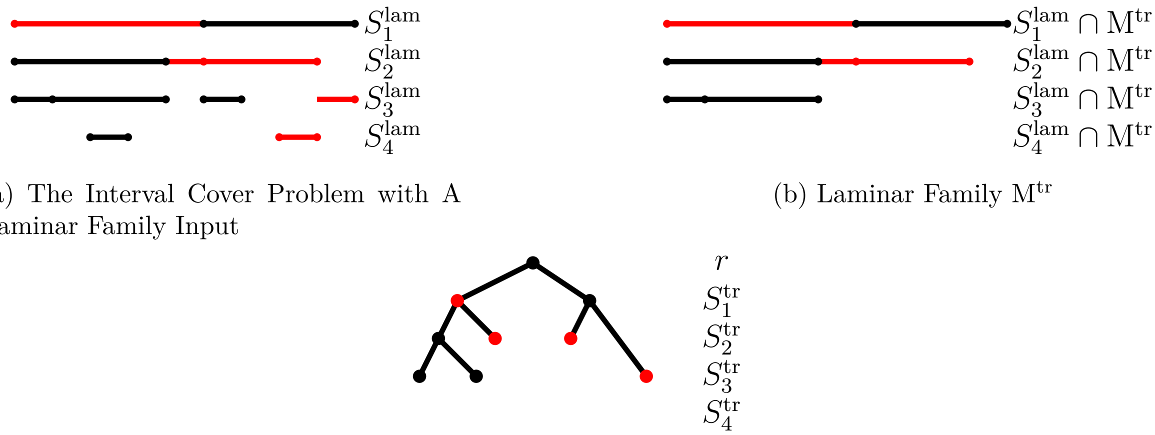
Now we change the LogBgt-IntCov with laminar family input into a LogBgt-TreeCov problem. For an interval $I \in \mathcal{U}^{\text{lam}}$, $I \ne \bigcup_{J \in \mathcal{U}^{\text{lam}}, J \subset I, J \ne I} J$. Since $\mathcal{U}^{\text{lam}}$ is a laminar family, we must select at least one interval that covers I. So we can ignore other intervals contained in I. We define $\mathrm{M}^{\text{tr}} = \mathcal{U}^{\text{lam}} \setminus \{I \in \mathcal{U}^{\text{lam}} : \exists I', I \subseteq I', I' \ne \bigcup_{J \in \mathcal{U}^{\text{lam}}, J \subset I', J \ne I'} J\}$. Then we construct a LogBgt-TreeCov with input $\eta^{\text{tr}} = (\mathcal{U}^{\text{tr}}; S^{\text{tr}}_1, S^{\text{tr}}_2, \ldots, S^{\text{tr}}_T; G)$, where $G = (V, E, r)$ is a rooted tree $G = (V, E, r)$ constructed from the laminar family M^{tr}:

1. Define the Vertex Set The vertex set V consists of $|\mathrm{M}^{\text{tr}}| + 1$ nodes. This includes
   (a) A root node r
   (b) A node for each set in M^{tr}.

2. Establish the Parent-Child Relationships. The tree structure is determined by the hierarchical containment in M^{tr}, since M^{tr} is laminar (i.e., sets are either disjoint or nested).
   (a) Choose the Root r: Define r as a virtual node representing the universal set that contains all elements considered in M^{tr}.
   (b) Determine the Parent of Each Node: For each set interval $I \in \mathrm{M}^{\text{tr}}$, find the smallest interval $I' \in \mathrm{M}^{\text{tr}} \cup \{\Gamma\}$ (Recall that Γ is the universal set) that strictly contains I. The node corresponding to I is a child of the node corresponding to I'.
   (c) Construct the Edge Set E: Add an edge from the parent node I' to the child node I.

In addition, we set S^{tr}_j be the set of corresponding items in $S^{\text{lam}}_j \cap \mathrm{M}^{\text{tr}}$ for $j = 1, 2, \ldots, T$, and $\mathcal{U}^{\text{tr}} = \bigcup_{1 \le j \le T} S^{\text{tr}}_j$.

Lemma E.3. *The tuple* $\eta^{\text{tr}} = (\mathcal{U}^{\text{tr}}; S^{\text{tr}}_1, S^{\text{tr}}_2, \ldots, S^{\text{tr}}_T; G)$*, constructed by the above process, satisfies the following properties:*



(a) The Interval Cover Problem with A Laminar Family Input

(b) Laminar Family M^{tr}

(c) The Corresponding Tree Cover Problem

Figure 3: An example of converting η^{lam} to a LogBgt-TreeCov problem. The red intervals or nodes represent a solution.

(a) $\text{Id}(u) > \text{Id}(\text{Par}(u))$ *for all* $u \in \mathcal{U}^{\text{tr}}$.

(b) *For any* $c \geq 1$*, there exists a* c*-valid solution for the LogBgt-IntCov problem with input* η^{lam} *if and only if there exists a* c*-valid solution for the LogBgt-TreeCov with input* η^{tr}.

Proof. TOPROVE 23 □

Proof of Theorem 6.1. This theorem follows directly from Lemma E.1, Lemma E.2, and Lemma E.3. □

E.2 Proofs Omitted in Section 6.2

For each $u \in \mathcal{U}^{\text{tr}}$, recall that the first type of cost $C_1(u) = \frac{1}{2^{\text{Id}(u)}}$ and the second type of cost

$$C_2(u) = \begin{cases} C_1(u) & \text{if } u \in \text{Leaf}(G) \\ \min\{C_1(u), \sum_{v \in \text{Ch}(u)} C_2(v)\} & \text{if } u \notin \text{Leaf}(G) \end{cases}$$

If $C_2(u) \neq C_1(u)$, we can find a series of descendants of u such that the sum of their C_1 values equals $C_2(u)$. We define the **replacement set** $R(u)$ of a node u as follows:

$$R(u) = \begin{cases} \{u\} & C_2(u) = C_1(u) \\ \cup_{v \in \text{Ch}(u)} R(v) & C_2(u) \neq C_1(u) \end{cases}$$

Lemma E.4. *We consider a LogBgt-TreeCov instance with input* $\eta^{\text{tr}} = (\mathcal{U}^{\text{tr}}; S_1^{\text{tr}}, \ldots, S_T^{\text{tr}}; G)$, $n = |\mathcal{U}^{\text{tr}}|$, $T = \lceil \log n \rceil$, *and* $T_0 = \lfloor \log \log \log n \rfloor$. *Without loss of generality, we assume* $n > 10$. *Then, for any constant* $c \geq 1$*, we have the following properties:*

(a) *For any* $D \subseteq \mathcal{U}^{\text{tr}}$*, if* $\sum_{u \in D} C_1(u) \leq c$*, then* $|D \cap S_i^{\text{tr}}| \leq c \times 2^i$ *for all* $1 \leq i \leq T$.

(b) *For any* c*-valid solution* D^*,

$$\sum_{u \in D^*} C_2(u) \leq \sum_{u \in D^*} C_1(u) \leq c \cdot T.$$

(c) *For any c-valid solution D^*,*

$$\sum_{u \in D^*, \mathrm{Id}(u) \leq T_0, C_2(u) \leq \frac{1}{\log n}} C_2(u) \leq c.$$

(d) *For any $u \in \mathcal{U}^{\mathrm{tr}}$ with $\mathrm{Id}(u) \leq T_0$, if $C_2(u) \leq \frac{1}{\log n}$, then $C_2(u) \neq C_1(u)$.*

(e) *For any $u \in \mathcal{U}^{\mathrm{tr}}$, if a set $R' \subseteq \mathcal{U}^{\mathrm{tr}}$ covers all leaves in $\mathrm{Des}(u)$, then*

$$\sum_{v \in R'} C_2(v) \geq C_2(u) = \sum_{v \in R(u)} C_2(v).$$

Proof. TOPROVE 24 □

Algorithm 4: Partial Enumeration Algorithm for Tree cover Problem

Data: A LogBgt-TreeCov problem with input $\eta^{\mathrm{tr}} = (\mathcal{U}^{\mathrm{tr}}; S_1^{\mathrm{tr}}, S_2^{\mathrm{tr}}, \ldots, S_T^{\mathrm{tr}}; G)$, where $G = (V, E, r)$ is a rooted tree

Result: A partial solution set $X \subseteq 2^{S_1^{\mathrm{tr}}} \times \cdots 2^{S_{T_0}^{\mathrm{tr}}}$

Function `PartEnumTree(`P,D`)`**:**
  if $(\sum_{v \in D} C_2(v)) + (\sum_{u \in P} C_2(u)) > 2c_0 T$ ***or*** $\exists i \leq T_0, |D \cap S_i^{\mathrm{tr}}| > 2c_0 \cdot 2^i$ **then**
    return $\emptyset$
  if $P = \emptyset$ ***or*** $\forall u \in P, \mathrm{Id}(u) > T_0$ **then**
    return $\{D\}$;
  Select u from P with minimum $\mathrm{Id}(u)$;
  $X' \leftarrow \emptyset$;
  if $u \notin \mathit{Leaf}(G)$ **then**
    $X' \leftarrow X' \cup$ `PartEnumTree(`$(P \setminus \{u\}) \cup \mathrm{Ch}(u), D$`)`;
  if $C_2(u) > \frac{1}{\log n}$ **then**
    $X' \leftarrow X' \cup$`PartEnumTree(`$P \setminus \{u\}, D \cup \{u\}$`)`;
  return X';
$X' \leftarrow$ `PartEnumTree(`$Ch(r)$,$\emptyset$`)`;
return $X = \{(D \cap S_1^{\mathrm{tr}}, D \cap S_2^{\mathrm{tr}}, \cdots, D \cap S_{T_0}^{\mathrm{tr}}) : D \in X'\}$

We employ a depth-first search (DFS) strategy to explore most of the states in the search space. During the search process, we maintain two sets:

- $P \subseteq \mathcal{U}^{\mathrm{tr}}$, representing the set of candidate elements that can still be explored, i.e., $\mathrm{Des}(P)$ contains all uncovered leaves.
- $D \subseteq \mathcal{U}^{\mathrm{tr}}$, storing the elements that have already been selected as part of the partial solution.

Initially, $P = \mathrm{Ch}(r)$ is the child set of the root, and $D = \emptyset$. At each recursive step, we select $u \in P$ with the smallest group index. The recursion proceeds by exploring two possibilities:

1. Adding u to the partial solution, i.e., including u in D and continuing the search.
2. Excluding u from the partial solution, i.e., replacing u with its child nodes while keeping D unchanged. (If u is a leaf, this option is not applicable.)

The search terminates when (P, D) fails to satisfy at least one of the following conditions:

1. $\exists u \in P$, $\mathrm{Id}(u) \leq T_0$

2. $\forall u \in D, \quad C_2(u) > \frac{1}{\log n}$

3. $(\sum_{v\in D} C_2(v)) + (\sum_{u\in P} C_2(u)) \leq 2c_0T$

4. $\forall 1 \leq i \leq T_0, \quad |D \cap S_i^{\mathrm{tr}}| \leq 2c_0 \cdot 2^i$

Lemma E.5. *For any sets $D, P \subseteq \mathcal{U}^{\mathrm{tr}}$ satisfying the following conditions:*

- $D \cap \mathrm{Des}(P) = \emptyset$,
- $\forall u, v \in P$, $\mathrm{Des}(u) \cap \mathrm{Des}(v) = \emptyset$,
- $\forall u, v \in D$, $\mathrm{Des}(u) \cap \mathrm{Des}(v) = \emptyset$,

If $(\sum_{v\in D} C_2(v)) + (\sum_{u\in P} C_2(u)) > c_0T$ for some constant $c_0 \geq 1$, then there is no c_0-valid solution D' satisfying $D' \subseteq D \cup \mathrm{Des}(P)$.

Proof. TOPROVE 25 □

Proof of Theorem 6.2.

We ignore all nodes $u \in S_j^{\mathrm{tr}}$ $(j \leq T_0)$ with $C_2(u) \leq \frac{1}{\log n}$. By E.4 (d) and the definition of $R(u)$, $\forall v \in R(u)$, v cannot be ignored, and v isn't in the first T_0 groups. Assume D^* is a c_0-valid solution that $\forall u, v \in D^*, u \neq v$, $\mathrm{Des}(u) \cap \mathrm{Des}(v) = \emptyset$. If D^* contains an ignored node u, we replace D^* by $(D^* - \{u\}) \cup R(u)$. Repeating the process, we can get another set D' from D^*, and D' covers all leaves.

Formally, denote $I = \{u \in \mathcal{U}^{\mathrm{tr}} : \mathrm{Id}(u) \leq T_0, C_2(u) \leq \frac{1}{\log n}\}$, then $D' = (D^* - I) \cup R(I \cap D^*)$ for $R(I \cap D^*) = \cup_{u\in(I\cap D^*)} R(u)$. Also,

$$\sum_{v\in R(I\cap D^*)} C_2(v) \leq \sum_{u\in(I\cap D^*)} \sum_{v\in R(u)} C_2(v) = \sum_{u\in(I\cap D^*)} C_2(u) \leq c$$

The last inequality is due to Lemma E.4 (c). Then for all $1 \leq j \leq T$,

$$|D' \cap S_j^{\mathrm{tr}}| \leq |(D^* - I) \cap S_j^{\mathrm{tr}}| + |R(I \cap D^*) \cap S_j^{\mathrm{tr}}| \leq (c_0 \times 2^j) + (c_0 \times 2^j) = 2c_0 \times 2^j$$

$|R(I \cap D^*) \cap S_j^{\mathrm{tr}}| \leq c_0 \cdot 2^j$ follows from Lemma E.4 (a). Therefore, D' is a $2c_0$-valid solution. Without loss of generality, we assume there is a $2c_0$-valid solution D'' without ignored nodes, satisfying $\forall u, v \in D''$, $\mathrm{Des}(u) \cap \mathrm{Des}(v) = \emptyset$. Because of Lemma E.5, the boundary cases won't remove the branches with $2c_0$-valid solutions. If we simulate the process of `PartEnumTree`, whenever we try to decide whether to choose u, there must exist a branch in which we choose u if and only if $u \in D''$. This means D'' is an extended solution for a partial solution in the output set X. Therefore, Algorithm 4 must output a partial solution with a $2c_0$-valid solution when there exists a c_0-valid solution. Now we only need to discuss its complexity.

For each $1 \leq i \leq T_0$, we focus on the number of branches when we decide on the vertices in S_i^{tr}. By ignoring the nodes u with $C_2(u) \leq \frac{1}{\log n}$, due to Lemma E.4 (b), the number of nodes we need to consider is not greater than:

$$(2c_0 \times T) \,/\, \left(\frac{1}{\log n}\right) \leq 2c_0 \times T^2$$

We only need to select at most $2c_0 \times 2^i$ elements, so the total number of branches is $\leq (1 + 2c_0 \cdot T^2)^{2c_0 \cdot 2^i}$. Therefore, the total number of branches is no more than:

$$\prod_{i=1}^{T_0} \left(2c_0T^2 + 1\right)^{2c_0 \times 2^i} = \exp\left(\sum_{i=1}^{T_0} (2c_0 \cdot 2^i) \ln(1 + 2c_0T^2)\right)$$

$$= \exp\left(O(2^{T_0}\log\log n)\right) = \exp\left(O(\log\log^2 n)\right) = O(n)$$

Thus the algorithm can be done in polynomial time.

□

E.3 Proofs Omitted in Section 6.3

For a given partial solution $(D_1, D_2, \ldots, D_{T_0})$, we remove the vertices that have already been covered. Let $V_0 = \left(\bigcup_{i=T_0+1}^{T} S_i^{\mathrm{tr}}\right) \setminus \left(\bigcup_{i=1}^{T_0} \mathrm{Des}(D_i)\right)$ be the set of remaining nodes under this partial solution, and let $L_0 = V_0 \cap \mathrm{Leaf}(G)$ be the leaf set of V_0. In this subsection, we denote $T_1 = \lfloor \log\log n \rfloor$. We expect the rounding algorithm (Algorithm 5) to either find an integral solution in LP-Tree-Cover($4c_0 + 1$, V_0, L_0) or confirm that there is no integral solution in LP-Tree-Cover($2c_0$, V_0, L_0). Recall that we design the complete algorithm as follows:

1. Check if LP-Tree-Cover($2c_0$, V_0, L_0) has a feasible solution. If so, obtain an extreme point x^*. Otherwise, confirm that there is no such integral solution.

2. Remove the leaves u with $x_u^* = 0$, and delete all the descendants of their parents. Then $\mathrm{Par}(u)$ becomes a leaf. Repeat this process until $x_u^* \neq 0$ for each leaf u. Let the modified node set and leaf set be V_1 and L_1, respectively.

3. For $u \in V_1$, attempt to round x_u^*. If $x_u^* \geq 1/2$, round it to 1. If $x_u^* > 0$, and u is not a leaf in $S_{T_1+1}^{\mathrm{tr}} \cup \cdots \cup S_T^{\mathrm{tr}}$, also round it to 1. In all other cases, round x_u^* to 0. Let D' be the set of nodes u for which x_u^* was rounded to 1. Note that D' may not cover L_1.

4. Remove all descendants in D', and attempt to choose another set from $S_{T_0+1}^{\mathrm{tr}} \cup \cdots \cup S_{T_1}^{\mathrm{tr}}$ to cover all leaves. Formalize this objective as LP-Tree-Cover. Specifically,

   - $V_2 = (V_1 \setminus \mathrm{Des}(D')) \cap \{u \in \mathcal{U}^{\mathrm{tr}} : T_0 + 1 \leq \mathrm{Id}(u) \leq T_1\}$, and
   - $L_2 = (V_2 \cap L_1) \cup \{u \in V_2 : \exists v \in \mathrm{Ch}(u) \cap (V_1 \setminus V_2), (V_2 \setminus \mathrm{Des}(D')) \cap \mathrm{Des}(v) \cap L_1 \neq \emptyset\}$.

   To understand this, observe that V_2 consists of the nodes in the (T_0+1)th to Tth groups that remain uncovered. The set L_2 includes nodes in V_2 that are either leaves or have at least one uncovered child with a group index greater than T_0 (i.e., at least one descendant leaf remains uncovered).

   Then solve LP-Tree-Cover($2c_0$, V_2, L_2). The fact that this problem must have feasible solutions is proved later, so we do not need to consider the case of no solution.

5. Let x^{**} be an extreme point of LP-Tree-Cover($2c_0$, V_2, L_2). For each $u \in V_2$, round it to 1 if and only if $x_u^{**} > 0$. Let $D'' = \{u \in V_2 : x_u^{**} > 0\}$, then D'' covers L_2.

6. Combine the three parts of the solution. That is, return $\left(\bigcup_{i=1}^{T_0} D_i\right) \cup D' \cup D''$.

Algorithm 5: Rounding Algorithm for Interval Cover Problem

Data: A (c, c_0)-LogBgt-TreeCov problem with input $\eta^{\text{tr}} = (\mathcal{U}^{\text{tr}}; S_1^{\text{tr}}, \ldots, S_T^{\text{tr}}; G)$ and a partial solution $(D_1, D_2, \cdots D_{T_0})$.

Result: A Solution D or "No Solution"

Set $V_0 \leftarrow \left(\bigcup_{i=T_0+1}^{T} S_i^{\text{tr}}\right) \setminus \left(\bigcup_{i=1}^{T_0} \text{Des}(D_i)\right)$, $L_0 \leftarrow V_0 \cap \text{Leaf}(G)$;

if *LP-Tree-Cover($2c_0$,V_0, L_0) has no solution* **then**

  return "No Solution";

Solve LP-Tree-Cover($2c_0$,V_0, L_0) and obtain an extreme point x^*;

$L_1 \leftarrow L_0, V_1 \leftarrow V_0$;

while $\exists u \in L_1, x_u^* = 0$ **do**

  $L_1 \leftarrow (L_1 \setminus \text{Des}(\text{Ch}(\text{Par}(u)))) \cup \{\text{Par}(u)\}$;

  $V_1 \leftarrow V_1 \setminus \text{Des}(\text{Ch}(\text{Par}(u)))$

Set $D' \leftarrow \{v \in V_1 \cap (\bigcup_{i=T_0+1}^{T} S_i^{\text{tr}}) : x_v^* \geq 1/2\} \cup \{v \in (V_1 \setminus L_1) \cap (\bigcup_{i=T_1+1}^{T} S_i^{\text{tr}}) : x_v^* > 0\}$;

$V_2 \leftarrow (V_1 \setminus \text{Des}(D')) \cap \left(\bigcup_{i=T_0+1}^{T_1} S_i^{\text{tr}}\right)$;

$L_2 \leftarrow \{u \in V_2 : u \in L_1 \text{ or } \exists v \in \text{Ch}(u) \cap (V_1 \setminus V_2), (V_2 \setminus \text{Des}(D')) \cap \text{Des}(v) \cap L_1 \neq \emptyset\}$;

Solve LP-Tree-Cover($4c_0$,V_2,L_2), and obtain basic feasible solution x^{**};

$D'' = \{u \in V_2 : x_u^{**} > 0\}$;

return $\left(\bigcup_{i=1}^{T_0} D_i\right) \cup D' \cup D''$

Lemma E.6. *In Algorithm 5, $x^*[V_1]$ is an extreme point of LP-Tree-Cover($2c_0$, V_1, L_1).*

Proof. TOPROVE 26 □

Lemma E.7. *In Algorithm 5, if there exists a solution x^* for LP-Tree-Cover($2c_0$, V_0, L_0), then the fractional solution set of LP-Tree-Cover($4c_0$, V_2, L_2) is non-empty.*

Proof. TOPROVE 27 □

Proof of Lemma 6.3. According to Algorithm 5, it is clear that $D' \cup D''$ covers all leaves in V_0, i.e., $L_0 \subseteq \text{Des}(D') \cup \text{Des}(D'')$. Therefore, $D = \left(\bigcup_{i=1}^{T_0} D_i\right) \cup D' \cup D''$ covers all leaves in the original tree.

If the input partial solution has a $2c_0$-valid extended solution, by simply deleting the nodes with chosen ancestors in this solution, it can be a $2c_0$-valid solution for the LP-Tree-Cover(c,$\mathcal{V}$,$\mathcal{L}$). So LP-Tree-Cover($2c_0$, V_0, L_0) has feasible solutions. In the linear program LP-Tree-Cover($2c_0$,L_1, V_1), $x_u^* > 0$ for all $u \in L_1$. The number of feasibility constraints, $\sum_{v \in \text{Anc}(u)} x_v = 1$, equals the number of leaves. According to Lemma 3.5, the number of non-zero, non-leaf variables is no more than the number of cardinality constraints. The number of cardinality constraints is $T - T_0 \leq \log n$, so we obtain the following inequality:

$$|\{u \in V_1 \setminus L_1 : x_u^* > 0\}| \leq \log n.$$

For $T_1 + 1 \leq i \leq T$, we choose node $u \in V_1$ if and only if $x_u^* \geq 1/2 \vee (x_u^* > 0 \wedge u \in L)$. Then,

$$|D \cap S_i^{\text{tr}}| \leq \left(2 \sum_{u \in V_1 \cap S_i^{\text{tr}}} x_u^*\right) + \log n \leq 2 \cdot 2c_0 \cdot 2^i + 2^{T_1+1} \leq (4c_0 + 1) \cdot 2^i.$$

For $T_0 + 1 \leq i \leq T_1$, by Lemma E.7, we can solve LP-Tree-Cover($4c_0$,V_2, L_2) and obtain an extreme point x^{**}. We perform the same steps as when deriving (V_1, L_1) from (V_0, L_0). We now derive (V_3, L_3) from (V_2, L_2) so that x^{**} corresponds to an extreme point of LP-Tree-Cover($4c_0$, V_3, L_3), and $x_u^{**} > 0$ for all $u \in L_3$. Then the number of non-zero, non-leaf nodes is no more than the number of cardinality constraints, and here the number of effective cardinality constraints is no more than $T_1 - T_0 \leq \log\log n$. Therefore, for $T_0 + 1 \leq i \leq T$,

$$|D \cap S_i^{\text{tr}}| \leq 2 \times 2c_0 \times 2^i + \log\log n \leq 4c_0 \times 2^i + 2^{T_0+1} \leq (4c_0 + 1) \times 2^i.$$

Combining these results, we have $|D \cap S_i^{\text{tr}}| \leq (4c_0 + 1) \times 2^i$ for all $i = 1, 2, \ldots, T$, and D is a $(4c_0 + 1)$-valid solution. ☐

Proof of Theorem 6.4. We first run Algorithm 4 to obtain a partial solution set X, and then run Algorithm 5 for each partial solution in X. If Algorithm 5 returns a $(4c_0 + 1)$-valid solution, then we output it directly. Otherwise, if Algorithm 5 always returns "No Solution," then we confirm that there is no c_0-valid solution.

According to Theorem 6.2, Algorithm 4 runs in polynomial time, and clearly, Algorithm 5 can be done in polynomial time. Therefore, the whole algorithm can be done in polynomial time.

If there exists a c_0-valid solution, then Algorithm 4 outputs a partial solution with a $2c_0$-valid extended solution, and by Lemma 6.3, Algorithm 5 returns a $(4c_0+1)$-valid solution based on this partial solution. Therefore, this algorithm can solve the $(4c_0 + 1, c_0)$-LogBgt-TreeCov. ☐

Proof of Theorem 6.5. Here we combine the conclusions of Theorem 6.1 and Theorem 6.4. We first convert the interval cover problem to a tree cover problem. If the original problem has a c_0-valid solution, then the tree cover problem has a $8c_0$-valid solution. Then, we solve the $(4 \cdot 8c_0 + 1, 12c_0)$-LogBgt-TreeCov to get a $(32c_0 + 1)$-valid solution. Finally, we convert this to an interval cover solution, which is $3(32c_0 + 1)$-valid. ☐

F An $O(\log n)$-Approximation for Min-norm Set Cover

We denote the norm minimization problem for set cover as MinNorm-SetCov. We present a simple algorithm based on randomized rounding and show it is an $O(\log n)$-Approximation. Note this is optimal assuming $P \neq NP$.

By Theorem 4.1, we only need to consider the LogBgt version of set cover (LogBgt-SetCov). We use the following natural LP relaxation:

$$\begin{aligned} \min \quad & 0 \\ s.t. \quad & \sum_{v \in B(u)} x_v \geq 1 \quad \forall u \in A \\ & x_v \geq 0 \quad \forall v \in Q \\ & \sum_{v \in S_i} x_v \leq 2^i \quad \forall 1 \leq i \leq T \end{aligned} \tag{1}$$

Here A is the set of all elements, Q is the set of all sets and $B(u)$ is the sets that contain u for $u \in A$.

Theorem F.1. *Assume that there is a 1-valid solution for LogBgt-SetCov. Then there is a randomized algorithm that can find an $O(\log n)$-valid solution for LogBgt-SetCov with probability larger than 1/2.*

Proof. TOPROVE 28 ☐