Covering a tree with rooted subtrees – parameterized and approximation algorithms[*]

Lin Chen[†] Daniel Marx [‡]

Abstract

We consider the multiple traveling salesman problem on a weighted tree. In this problem there are m salesmen located at the root initially. Each of them will visit a subset of vertices and return to the root. The goal is to assign a tour to every salesman such that every vertex is visited and the longest tour among all salesmen is minimized. The problem is equivalent to the subtree cover problem, in which we cover a tree with rooted subtrees such that the weight of the maximum weighted subtree is minimized. The classical machine scheduling problem can be viewed as a special case of our problem when the given tree is a star. We provide approximation and parameterized algorithms for this problem. We first present a PTAS (Polynomial Time Approximation Scheme). We then observe that, the problem remains NP-hard even if tree height and edge weight are constant, and present an FPT algorithm for this problem parameterized by the largest tour length. To achieve the FPT algorithm, we first formulate the problem as an integer linear program having a certain "tree-fold" structure. Then we show that an ILP with such a structure is FPT, which is a generalization of an earlier FPT result for n-fold integer programming by Hemmecke, Onn and Romanchuk [5]. This extension of n-fold ILP may be of independent interest.

Keywords: Approximation schemes; Fixed Parameter Tractable; Integer Programming; Scheduling

1 Introduction

We consider the multiple traveling salesmen problem on a given tree $T = (V, E)$. In this problem there is a root $r \in V$ where all the m salesmen are initially located. There is a weight $w_e \in \mathbb{Z}_+$ associated with each edge $e \in E$, which is the time consumed by a salesman if he passes this edge. Each salesman starts at r, travels a subset of the vertices and returns to r. The goal is to determine the tours traveled by each salesman such that every vertex is visited by some salesman, and the makespan, i.e., the time when the last salesman returns to r, is minimized.

We observe that the tour of every salesman is actually a subtree rooted at r, and the total traveling time of each salesman is exactly twice the total weight of edges in the subtree. Therefore the problem is equivalent as the minmax subtree cover problem, where we aim to find m subtrees $T_i = (V(T_i), E(T_i))$ for $1 \le i \le m$ such that $r \in V(T_i)$, $V = \cup_i V(T_i)$ and $\max_i w(T_i)$ is minimized, where $w(T_i) = \sum_{e\in E(T_i)} w_e$. We call $w(T_i)$ as the weight of the subtree T_i and $\max_i w(T_i)$ the *makespan*.

The subtree cover problem is a fundamental problem in computer science and has received many studies in the literature. Indeed, when the given graph is a star, the problem is equivalent to the identical machine scheduling problem $P||C_{max}$, where the goal is to assign a set of jobs of processing times $w_1, w_2, \cdots, w_n$ onto m identical parallel machines such the largest load among machines is minimized. We may view each job as an edge of weight w_j in a star graph, whereas $P||C_{max}$ falls exactly into the problem of covering a star with m stars. There exists an FPTAS (fully polynomial time approximation scheme) for the scheduling problem if m is constant [20], and a PTAS (polynomial time approximation scheme) if m is part of the input [7]. Studies on FPT algorithms for the scheduling problem is relatively new. In 2013, Mnich and Wiese [17] provided an FPT (fixed parameter tractable) algorithm parameterized by the largest job processing time $w_{max} = \max\{w_j|1 \le j \le n\}$. Very recently, Knop and Koutecký [13] observes that many scheduling problems can be formulated as an integer program with a special structure called n-fold integer program. Exploiting the FPT algorithm for the n-fold integer program [5] they are able to show an FPT algorithm for various scheduling problems.

The problem becomes much more complicated when the given graph is a tree. Xu et al. [21] showed that there exists an FPTAS when the number of subtrees, m, is a constant. However, it is not known whether there exists a PTAS if m is part of the input. The best-known approximation algorithm so far has an approximation ratio of $(2 + \epsilon)$ by Nagamochi and Okada [18]. If the given graph is a general graph, there exists a 3-approximation algorithm by Nagamochi and Okada [18].

Our contribution. Our main contribution is to show that the subtree cover problem admits a PTAS and a fixed parameter tractable (FPT) algorithm (parameter-

[*]Supported by ERC Starting Grant PARAMTIGHT (No. 280152) and Consolidator Grant SYSTEMATICGRAPH (No. 755978)

[†]Department of Computer Science, University of Houston. Email: `chenlin198662@gmail.com`.

[‡]Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI). Email: `dmarx@cs.bme.hu`.

ized by the makespan). More precisely, we prove the following theorems.

THEOREM 1.1. *There exists a PTAS of running time $m^{O(1/\epsilon^4)}$ for the subtree cover problem.*

THEOREM 1.2. *For some computable function f, there exists an FPT algorithm of running time $f(B)m^4$ for determining whether there exists a feasible solution for the subtree cover problem of makespan B.*

We remark that, despite the fact that the special case of covering a star admits an FPT algorithm parameterized by the largest edge weight, we show in this paper that the subtree cover problem remains NP-hard even if the tree is of height 2 and every edge has a unit weight. Therefore, we restrict our attention to the larger parameter B.

Indeed, our FPT algorithm relies on an FPT algorithm for a more general integer programming problem, which extends the existing FPT algorithm for the n-fold integer programming [5]. We consider the following integer programming:

$$\min\{\mathbf{c}^T\mathbf{x} : A\mathbf{x} = \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \in \mathbb{Z}^{nt}\}, \quad (1.1)$$

In the n-fold integer programming, the matrix A consists of small matrices A_1 and A_2 as follows (Here A_1 is an $s_1 \times t$-matrix and A_2 is an $s_2 \times t$-matrix).

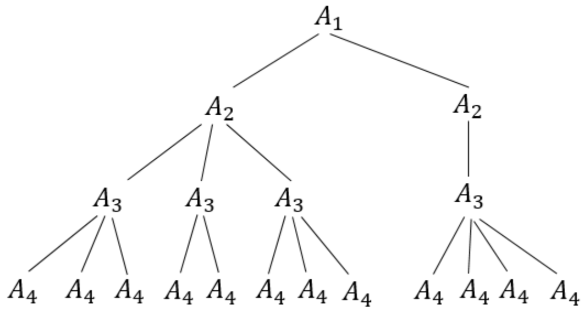
$$A = \begin{bmatrix} A_1 & A_1 & \dots & A_1 \\ A_2 & 0 & \dots & 0 \\ 0 & A_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & A_2 \end{bmatrix}$$

More precisely, the matrix A consists of one row of $(A_1, A_1, \cdots, A_1)$ and a submatrix with A_2 being at the main diagonal. We remark that throughout this paper 0s that appear in a matrix refer to a submatrix consisting of the natural number 0.

The n-fold integer programming has received many studies in the literature. Indeed, the natural ILP formulation of the scheduling and bin packing problem falls into an n-fold integer programming, as is observed by Knop and Koutecký [13]. In 2013, Hemmecke, Onn and Romanchuk presented an FPT algorithm for n-fold integer programming with the running time of $f(s_1, s_2, ||A||_\infty)n^3L$ where f is some computable function, $||A||_\infty$ is the largest absolute value among all entries of A and L is the encoding length of the problem. This algorithm implies an FPT algorithm parameterized by the largest job processing time for $P||C_{max}$ and many other scheduling problems [13]. We further extend their result by considering a broader class of integer programming, namely tree-fold integer programming as we describe as follows.

The structure of an n-fold matrix could be viewed as a star with the root representing the row of $(A_1, A_1, \cdots, A_1)$ and each leaf representing one of the rows $(0, \cdots, 0, A_2, 0, \cdots, 0)$. More precisely, we can view each row i as a vertex i such that vertex i is a parent of vertex j if row i dominates row j, where by saying row i dominates row j, we mean row j is more "sparse" than row i as a vector, i.e., if the k-th coordinate of row j is non-zero, then the k-th coordinate of row i is also non-zero. Using this interpretation, we can generalize an n-fold matrix to a tree-fold matrix. The following is an example (see the matrix at the top of the next page).

A tree-representation of the matrix above is:


A_1
A_2 A_2
A_3 A_3 A_3 A_3
A_4 A_4 A_4 A_4 A_4 A_4 A_4 A_4 A_4 A_4 A_4 A_4

In general, a tree-fold matrix A consists of n copies of small matrices A_1, A_2, $\cdots$, A_τ with A_i being an $s_i \times t$-matrix. Every row consists of 0's and some A_i's in the form of $(0, \cdots, 0, A_i, A_i, \cdots, A_i, 0, \cdots, 0)$ (i.e., A_i appears consecutively). Every column consists of 0's and exactly one copy of each A_i. Furthermore, if we call a row containing A_i as an A_i-row, then any A_i-row is dominated by some A_{i-1}-row, that is, if at a certain row A_i appears consecutively from column ℓ to column k, then there exists some A_{i-1}-row such that A_{i-1} appears consecutively from ℓ' to k' such that $\ell' \leq \ell < k \leq k'$. Representing the matrix as a tree, every row is represented as a vertex and the vertex corresponding to each A_{i-1}-row will be the parent of the vertex corresponding to A_i-row it dominates.

To facilitate the analysis, we further require that the A_1-row contains no 0 and every A_τ-row contains exactly one copy of A_τ, that is, all rows containing A_τ form a sub-matrix with A_τ being at the diagonal. Note that this assumption causes no loss of generality: If it is not the case, we can always add a set of dummy constraints: $0 \cdot \mathbf{x} = 0$, whereas A_1 and A_τ become a $1 \times t$-dummy matrix consisting of 0.

We define ILP (1.1) with A being a tree-fold matrix as a tree-fold integer programming and establish the

$$A = \begin{bmatrix}
A_1 & A_1 & A_1 & A_1 & A_1 & A_1 & A_1 & A_1 & A_1 & A_1 & A_1 & A_1 \\
A_2 & A_2 & A_2 & A_2 & A_2 & A_2 & A_2 & A_2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & A_2 & A_2 & A_2 & A_2 \\
A_3 & A_3 & A_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & A_3 & A_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & A_3 & A_3 & A_3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & A_3 & A_3 & A_3 & A_3 \\
A_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & A_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & A_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & A_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & A_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & A_4 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & A_4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & A_4 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & A_4 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & A_4 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & A_4 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & A_4
\end{bmatrix}$$

following FPT result.

THEOREM 1.3. *For some computable function f, there exists an FPT algorithm of running time $f(t, s_1, s_2, \cdots, s_\tau, ||A||_\infty)n^3 L$ for a tree-fold integer programming, where $||A||_\infty$ is the largest absolute value among all entries of A, and L is the length of the binary encoding of the vector* $(\mathbf{c}, \mathbf{b}, \mathbf{l}, \mathbf{u})$.

Note that $||A||_\infty = \max_j\{||A_j||_\infty\}$, thus the FPT term $f(t, s_1, s_2, \cdots, s_\tau, ||A||_\infty)$ only depends on the small matrices and does not rely on the structure of A. We also remark that, by introducing slack variables for inequalities, our theorem also holds for the integer programming: $\min\{\mathbf{c}^T\mathbf{x} : A\mathbf{x} \le \mathbf{b}, \mathbf{l} \le \mathbf{x} \le \mathbf{u}, \mathbf{x} \in \mathbb{Z}^{nt}\}$.

Related work. As we have mentioned, the problem of covering a star with stars is exactly the identical machine scheduling problem $P||C_{max}$. In terms of approximation algorithms, the first FPTAS (when m is constant) [20] and PTAS (when m is part of the input) [7] for this problem date back to 1976 and 1987, respectively. In recent years, Jansen et al. [10, 11] provided an FPTAS and a PTAS of improved running times, which are shown to be essentially the best possible under exponential time hypothesis by Chen et al. [1]. In terms of FPT algorithms, Mnich and Wiese [17] showed that $P||C_{max}$ is FPT parameterized by the largest job processing time (edge weight). Very recently, Knop and Koutecký [13] observes the relationship between the scheduling problem and n-fold integer programming in terms of FPT algorithms. Indeed, they show that a variety of scheduling problems, including $P||C_{max}$, could be formulated as an n-fold integer programming. Applying the FPT algorithm for n-fold integer programming by Hemmecke, Onn and Romanchuk [5], an FPT algorithm for $P||C_{max}$ follows. It is worth mentioning that parameterized studies for integer programming that has a sparse structure have received much attention in the literature, e.g., [9, 14, 15].

Covering a tree with subtrees is much more complicated. Even et al. [3] gave a 4-approximation algorithm for the subtree cover problem, which was later improved by Nagamochi and Okada [18] into a $(2+\epsilon)$-approximation algorithm. In 2013, Xu et al. [21] showed that if the number of subtrees, m, is a constant, then the problem admits a pseudo-polynomial time exact algorithm and an FPTAS. It remains an important open problem whether there exists a PTAS for the subtree cover problem if m is part of the input. We are not aware of FPT algorithms for this problem.

An even more general problem is to cover the vertices of a general graph with trees where each tree is a subgraph. There exists a 4-approximation algorithm by Even et al. [3], which was improved later by Nagamochi and Okada [18] to a 3-approximation algorithm.

In all the related work we mention above, a feasible solution requires every (sub-)tree to contain a uniform root. There are other variants of the subtree cover problem. One variant that also receives many studies in the literature is the unrooted version of the problem, where we aim to cover the vertices of a given graph with trees and do not necessarily require each tree to be rooted at the same vertex. For this problem, the best-known result is a 3-approximation algorithm by Khani and Salavatipour [12].

2 A PTAS

The goal of this section is to give a PTAS for the subtree cover problem.

2.1 Preliminaries Given two vertices v and v', if v is not a descendant of v', and v' is not a descendant of v either, we say the two vertices are *incomparable*.

We denote by $w(u, v)$ the total weight of the edges on the path between two vertices u and v. We denote by $w(T)$ the total weight of edges in a tree T.

Note that any feasible solution, say, R, specifies m subtrees T_i^R $(1 \le i \le m)$ that cover T. For ease of description, we assume there exist m agents, each owning one subtree T_i^R. Taking the view point of the traveling salesman problem, the subtree T_i^R is essentially the tour of the i-th salesman (divided by 2). For any vertex v, we denote by $M^R(v)$ the set of agents whose subtree contains the vertex v. Let $T(v)$ be the subtree of T rooted at v, and analogously $T_i^R(v)$ the subtree of T_i^R rooted at v. Any subtree of $T(v)$ that contains v is called a v-tree. We say $T_i^R(v)$ is the v-tree owned by agent i at vertex v. We define the weight of agent i at v as the weight of $T_i^R(v)$. We denote by $\Gamma^R(v)$ the set of all the v-trees owned by agents at v. The superscript R may be dropped if it is clear from the context which solution we are referring to. We summarize the notations as Table 1.

Throughout this section, when we say we lift the weight of an agent i by δ at a certain vertex v, we mean $w(T_i(v))$, as well as $w(T_i(v'))$ for any $v' \in V(T_i)$ which is an ancestor of v, are increased by δ simultaneously. Equivalently, we may imagine that we attach a new edge of weight δ to a leaf of T_i. Obviously, after lifting if the weight of each agent at the root r is still bounded by c, then the makespan of the solution is at most c.

Preprocessing. We preprocess the given tree T in the following way. Notice that there exists a 4-approximation algorithm for the subtree cover problem [3]. We can scale the given instance so that $1/4 \le OPT \le 1$. If an edge has a weight larger than ϵ^2, say, $w_{(u,v)} \ge \epsilon$, then we sub-divide the edge into $k \le 1/\epsilon^2$ edges (u, v_1), (v_1, v_2), $\cdots$, (v_{k-2}, v_{k-1}), (v_{k-1}, v) such that the weight of each edge is at most ϵ^2, and their total weight equals $w_{(u,v)}$. After sub-dividing every edge, we get a new instance in which every edge has a weight bounded by ϵ^2. It is easy to see that every feasible instance of the original instance implies a feasible instance of the new intance with the same makespan, and vice versa.

2.2 Structuring the optimum Let σ be an arbitrary v-tree, we say it is *small*, if $w(\sigma) < \epsilon$. Otherwise we say it is *big*. Similarly we say an agent is small at v, if its weight at v is small, and big otherwise. The goal of this subsection is to prove the following theorem, which is crucial for our algorithm.

THEOREM 2.1. *There exists a feasible solution of makespan $OPT + O(\epsilon)$ such that for any $v \in V$, $M(v)$ either consists of only one small agent, or every agent of $M(v)$ is big.*

The theorem looks simple at first glance, as we can always merge small v-trees until it becomes big. However, any merging will cause an increase in the subtree owned by some agent. The question is, can we always select the correct agent so that the subtree owned by each of them is increased by $O(1)$ times? Towards this, we need the following Lemma 2.1.

We first describe the notion of a simple v-tree, which is required by Lemma 2.1. Given a feasible solution, suppose some v-tree σ contains vertices v_1, v_2, $\cdots$, v_k where each v_j is a child of v. Then σ could be split into k edge-disjoint v-trees such that each v-tree contains exactly one child of v. A v-tree that only contains one child of v is called a simple v-tree. We split every v-tree of $\Gamma^R(v)$ into simple v-trees, and denote by $\Gamma_s^R(v)$ the set of all simple v-trees.

LEMMA 2.1. *Let D be any set of vertices that are mutually incomparable. Given a feasible solution R, a new feasible solution R' could be constructed such that the following is true:*

- *The weight of each subtree, and hence the makespan, increases by $O(d_{max})$, where $d_{max} = \max_{v \in D} \max_{\sigma \in \Gamma_s^R(v)} w(\sigma)$, i.e., it is the maximal weight among all the simple v-trees for $v \in D$.*
- *$|M^{R'}(v) \cap M^{R'}(v')| \le 1$ for any $v, v' \in D$.*

Lemma 2.1 is the key to the proof of Theorem 2.1. Briefly speaking, it states that with some bounded loss, we can restrict our attention to a feasible solution such that the subtrees of any two agents can share at most one vertex in D. This guides the selection of subtrees to merge. To prove Lemma 2.1, the basic idea is to first allow an agent to own a fraction of a tree, seeking for a fractional solution with a good structure, and eventually

Table 1: Some Notations

T_i^R	The subtree owned by agent i
$T_i^R(v)$	The subtree of T_i rooted at v Equivalently, the v-tree owned by agent i
v-tree	Any subtree of $T(v)$ rooted at v, e.g., $T_i^R(v)$
$M^R(v)$	The set of agents whose subtree contains vertex v
$w(T_i^R(v))$	The total edge weight of a tree $T_i^R(v)$ Equivalently, the weight of agent i at v
$\Gamma^R(v)$	The set of all v-trees owned by agents

"round" back this fractional solution into an integral solution.

Towards the proof, we need the following lemmas.

We assume that every T_i contains at least one leaf of T since otherwise any vertex $v \in V(T_i)$ is also contained in some $V(T_j)$ which contains any descendant of v, and T_i could be discarded.

For any $v \in V$, we define by $LF(v)$ the set of leaves of the subtree $T(v)$ and LF the set of leaves of T. It is not difficult to see that $M(v) = \cup_{w\in LF(v)} M(w)$, which gives rise to the following lemma.

LEMMA 2.2. *For any vertex v, let S be a set of descendants of v that separates v from $LF(v)$, i.e., any path from a leaf in $LF(v)$ to v has to visit some vertex of S. Then $M(v) = \cup_{w\in S} M(w)$.*

LEMMA 2.3. *Given a ground set U and a family of k subsets $S_i \subseteq U$ such that $|S_i| \geq 2$ and $|S_i \cap S_{i'}| \leq 1$ for any $i, i' \in \{1, 2, \cdots, k\}$, there exists $H_i' \subseteq S_i$ such that $H_i \cap H_{i'} = \emptyset$, and $|H_i| \geq 1/3 \cdot |S_i|$.*

Proof. We first observe that it suffices to prove the lemma for $|S_i| \leq 3$. To see why, suppose the lemma is true for $|S_i| \leq 3$, we claim that it is also true for arbitrary large set size. Suppose $|S_i| \geq 4$, then we can divide S_i into disjoint subsets such that every subset is of size 2 or 3. Let S_j^i be these subsets. We replace every S_i with its subsets, then it follows directly that $|S_j^i \cap S_{j'}^{i'}| \leq 1$ due to $|S_i \cap S_{i'}| \leq 1$. Now according to the lemma for set size at most 3, we can find disjoint subsets $H_j^i \subseteq S_j^i$ such that $|H_j^i| \geq 1/3 \cdot |S_j^i|$. Let $H_i = \cup_j H_j^i$ and the claim is proved.

Now we prove the lemma for $|S_i| \leq 3$. It suffices to show that we can select a distinct element from every S_i. We prove it via Hall's theorem. We construct a bipartite graph $G = (V_1 \cup V_2, E)$ where every S_i corresponds to a vertex in V_1, every element of U corresponds to a vertex in V_2, and there is an edge between them if and only if the element is contained in S_i. Given that $|S_i| \geq 2$ and $|S_i \cap S_{i'}| \leq 1$, it is easy to verify that the union if every ℓ sets has a total size at least k, which implies a matching in the bipartite graph, and consequently the lemma is proved. □

Now we come to the proof of Lemma 2.1.

We introduce the notion of a fractional solution. In an integral solution, every agent owns one subtree $T_i(v)$ at every vertex v. In a fractional solution, however, we allow an agent to own a fraction of a v-tree at vertex v. For example, let $\Gamma(v) = \{\sigma_1(v), \sigma_2(v), \cdots, \sigma_\ell(v)\}$ be all the v-trees, then we allow $\theta_{ij} \in [0, 1]$ fraction of $\sigma_j(v)$ to be assigned to agent i. We remark that, in a fractional solution it is required that $\sum_i \theta_{ij} = 1$, i.e., each v-tree is assigned. However, we do not necessarily require that $\sum_j \theta_{ij} = 1$, i.e., an agent could be assigned multiple v-trees. If R is a fractional solution, then $M^R(v)$ is the set of agents which contains a positive fraction of a v-tree.

Proof. [of Lemma 2.1] Let $\Gamma_s^R(v) = \{\sigma_1(v), \sigma_2(v), \cdots, \sigma_\ell(v)\}$ be the set of all the simple v-trees for each $v \in D$. Notice that $w(\sigma_k(v)) \leq d_{max}$ by definition. In the following we will re-assign v-trees to achieve the lemma. Suppose $\{i, i'\} \in M^R(v)$ for some $v \in D$. Consider any v-tree owned by agent i in solution R. Re-assigning this v-tree to agent i' leads to another feasible solution since vertex v is also contained in the subtree owned by agent i'. Furthermore, after we re-assign v-trees for $v \in D$, we can still re-assign v'-trees for $v' \in D$. To see why, notice that either $v' = v$, and the claim is straightforward, or v' is incomparable to v, therefore the re-assignment of v-trees do not influence any v'-tree.

The re-assignment of v-trees is done in three steps. In the first and second steps, we allow fractional re-assignment of v-trees. For example, we can re-assign θ fraction of a v-tree from agent i to i'. By doing so the weight of $\theta w(\sigma)$ is moved from agent i to i' and we derive a solution where θ fraction of the v-tree is assigned to agent i', and $1 - \theta$ fraction is assigned to agent i. In general the re-assignment of step 1 and 2 leads to a fractional solution where $\theta_{ij}(v) \in [0, 1]$ fraction of a v-tree $\sigma_j(v)$ is assigned to agent i such that $\sum_i \theta_{ij} = 1$. It seems there is not really a natural combinatorial interpretation of a fractional solution. It is only an intermediate state of our rounding procedure and all the v-tree will be assigned integrally at the last step of the rounding procedure.

Step 1. Consider the current fractional solution and let it be F. For any two vertices $v, v' \in D$, as long as $|M^F(v) \cap M^F(v')| \geq 2$, we re-assign v-trees and v'-trees as follows. Let $\{i, i'\}$ be two arbitrary agents of $M^F(v) \cap M^F(v')$. Suppose currently $\theta_k^h(x)$ fraction of the x-tree $\sigma_k^h(x)$ is assigned to agent h at vertex x where $h = i, i'$, $x = v, v'$, $k = 1, 2, \cdots$. We define by $L^h(x) = \sum_k \theta_k^h(x) w(\sigma_k^h(x))$ as the weight of agent h at x. Without loss of generality we assume $L^i(v)$ is the smallest among $L^i(v), L^{i'}(v), L^i(v'), L^{i'}(v')$. Now we re-assign all the (fractions of) v-trees from machine i to machine i', and meanwhile re-assign $\theta \cdot \theta_k^{i'}(v')$ fraction of v'-tree $\sigma_k^{i'}(v')$ from agent i' to agent i at v', where $\theta = L^i(v)/L^{i'}(v')$. The weight of agent i (and i') at v decreases (increases), while the weight of i at v' increases (decreases). However, the summation of them remain the same.Further notice that applying the above re-assignment, we derive a new fractional feasible solution F' such that $\sum_{v \in D} |M^{F'}(v)| \leq \sum_{v \in D} |M^F(v)| - 1$, where $|S|$ denotes the cardinality of a set S.

We apply the above procedure until a fractional solution F is derived such that $|M^F(v) \cap M^F(v')| \leq 1$ and this completes step 1 of the rounding procedure.

Step 2. Let $Q^F(v)$ be the set of all the simple v-trees that are fractionally assigned in F. Obviously $Q^F(v) \subseteq \Gamma_s^F(v)$. For simplicity let $Q^F(v) = \{\sigma_1(v), \sigma_2(v), \cdots, \sigma_s(v)\}$ for $s \leq \ell$. Again we denote by $\theta_k^i(v)$ the fraction of v-tree $\sigma_k(v)$ assigned to agent $i \in M^F(v)$. If $q = |Q^F(v)| \leq |M^F(v)|$ we are done. Otherwise we re-assign v-trees using the rounding procedure from [16]. Consider the following linear system.

$$\sum_{i \in M^F(v)} \sum_{k=1}^{q} x_{ik} = 1 \qquad \forall 1 \leq k \leq |Q^F(v)|$$

$$\sum_{k=1}^{q} w(\sigma_k(v)) x_{ik} = L^i(v) \qquad \forall i \in M^F(v)$$

$$0 \leq x_{ik} \leq 1$$

Obviously $x_{ik} = \theta_k^i(v)$ is a feasible solution. According to [16], in the extreme point solution of the above linear system, at most $|M^F(v)|$ v-trees are fractionally assigned. Hence we can re-assign v-trees according to this extreme point solution and derive a new fractional solution F' such that $|Q^{F'}(v)| \leq |M^{F'}(v)|$ is true, and furthermore, for every agent the summation of its weights at all vertices in D remains the same.

Step 3. Consider $M^{F'}(v)$ for $v \in D$. In the following we will determine $H(v) \subseteq M^{F'}(v)$ such that all the v-trees that are fractionally assigned will be re-assigned integrally to agents in $H(v)$. If $|M(v)| = 1$ then $H(v) = M(v)$, and we know that all the simple v-trees are assigned to only one machine, implying that v-trees are already assigned integrally. Otherwise consider all the vertices of D such that $|M^{F'}(v)| \geq 2$ and let $D' \subseteq D$ be the set of these vertices. Recall that $|M^{F'}(v) \cap M^{F'}(v')| \leq 1$ for any $v, v' \in D' \subseteq D$, we can apply Lemma 2.3 to derive $H(v) \subseteq M^{F'}(v)$ such that $H(v) \cap H(v') = \emptyset$ for any $v, v' \in D'$, and $|H(v)| \geq 1/3 \cdot |M^{F'}(v)|$. Notice that there are at most $|M^{F'}(v)|$ simple v-trees fractionally assigned and the weight of each of them is at most d_{max}. We can re-assign these v-trees to agents in $H(v)$ such that at most three v-trees are assigned to one agent, whereas the weight of every agent in $H(v)$ increases by at most $3d_{max}$ at v.

Let A' be the solution we eventually derive. We compare R with A'. Although fractional assignments are created during the rounding procedure, eventually every single v-tree is assigned integrally. Therefore, from R to A' we only re-assign some of the simple v-trees. Each time we re-assign a (simple) v-tree from agent i to i', we create a new solution by removing this v-tree from T_i, and append it to $T_{i'}$. As vertices of D are mutually incomparable, for any $v, v' \in D$, the re-assignment of any v-tree does not influence the re-assignment of a v'-tree. Therefore, A' is a feasible solution. We now check the makespan of A'. Consider an arbitrary agent i. The summation of its weights at all vertices in D remain the same in step 1 and 2, and increase by at most $3d_{max}$ in step 3 since $H(v)$'s are disjoint. Hence, the makespan of A' is at most $O(d_{max})$ larger than that of R. $\square$

To prove Theorem 2.1, the basic idea is to select an appropriate D and apply Lemma 2.1, and then we may invoke Lemma 2.3 to select the correct subtrees to merge. Note that D has to be chosen in such a way that: i). $d_{max} = O(\epsilon)$; ii). By merging subtrees rooted at each vertex in D, we can get big subtrees; iii). Vertices in D are mutually disjoint; iv). Every subtree owned by some agent should contain some vertex in D.

Proof. [of Theorem 2.1] Consider the optimal solution. If the weight of an agent at r is no more than ϵ we simply lift its weight to ϵ, this causes the makespan to increase by at most ϵ. Now for each vertex v, we calculate the total weight of all agents at v and define it as the *whole weight* of v. For any leaf v, we consider its path from v to the root r. Starting from v, we check one by one the whole weight of each vertex on the path and pick the first vertex whose whole weight is larger than or equal to ϵ. We call such a vertex as a critical vertex and let

CR be the set of all critical vertices.

If all vertices in CR are mutually incomparable, we are done. Otherwise we modify the given tree so that critical vertices become mutually incomparable. Indeed, we modify T in the following way. Let v and v' be two arbitrary vertices in CR such that v' is a descendant of v. According to the definition of a critical vertex, v is the first vertex whose whole weight is larger than or equal to ϵ on the path from some (could be multiple) leaf to r. For any such leaf, say, v_1, we consider its path to r that passes v. Let v_2 be the vertex on this path right before v. Now we sub-divide the edge (v_2, v) by adding a new vertex v_3. We let the weight of edge (v_2, v_3) be the same as the weight of (v_2, v), and the weight of (v_3, v) be 0. By doing so a new tree T' is constructed such that any feasible solution of T implies a feasible solution of T' with the same makespan, and vice versa. If we consider the path from v to r in T', the first vertex whose whole weight is larger than or equal to ϵ becomes v_3 instead of v. Furthermore, we have the following two observations:

- any vertex that is incomparable to v is also incomparable to v_3;
- v_3 is incomparable to v'.

We apply the above procedure as long as there exist two critical vertices which are not incomparable. Eventually we derive a new tree together with a set of critical vertices which are mutually incomparable. For simplicity we still denote the new tree as T'. It is obvious that if we prove Theorem 2.1 for T', then Theorem 2.1 is also true for T.

From now on we restrict our attention on T'. For simplicity we still denote the set of critical vertices in T' as CR. We let $CH = \{w|w \text{ is a child of } v \in CR\}$ be the set of children of vertices in the critical set. In the following we construct a new feasible solution R of makespan $OPT + O(\epsilon)$ such that for any $w \in CH$ we have $|M^R(w)| = 1$, and for any $v \in CR$, each agent of $M^R(v)$ has a weight at least ϵ at v. We claim that, this new solution satisfies the theorem. To see why, notice that from any leaf of the tree to the root r the path has to visit at least one vertex of CR, consequently also one vertex of CH. Hence, any vertex $x \in V \setminus (CR \cup CH)$ is either a descendant of $w \in CH$ and thus $|M^R(x)| = 1$, or it is an ancestor of some vertices in CR and thus $M^R(x) = \cup_{v \in CR(x)} M^R(v)$ where $CR(x) \subseteq CR$ is the set of descendants of x in CR (note that any path from a leaf to v has to visit some vertex of $CR(x)$), and consequently $M^R(x)$ consists of only big agents. Hence, the claim is true.

Consider the optimal solution for T'. We claim that, the weight of each simple v-tree for $v \in CR$ is at most 3ϵ. To see why, notice that $v \in CR$ implies that the whole weight of each child of v is at most ϵ. Given the fact that the weight of each edge is bounded by ϵ, the claim follows. Furthermore, critical vertices are incomparable. Hence, we set $D = CR$ and apply Lemma 2.1 to derive a feasible solution R_1 of makespan $OPT + O(\epsilon)$ such that for any $v, v' \in CR$ we have $|M^{R_1}(v) \cap M^{R_1}(v')| \le 1$. Notice that this solution is constructed via re-assignment of v-trees, hence the whole weight of every vertex remains the same.

Consider CH. For any $w, w' \in CH$, it is easy to see that we also have $|M^{R_1}(w) \cap M^{R_1}(w')| \le 1$. Let $CH_1 = \{v \in CH||M^{R_1}(v)| = 1\}$ and $CH_2 = CH \setminus CH_1$. Applying Lemma 2.3 to CH_2 we can find $H^{R_1}(v) \subseteq M^{R_1}(v)$ such that $H^{R_1}(v) \cap H^{R_1}(v') = \emptyset$ for any $v, v' \in CH_2$. Recall that the whole weight of any $v \in CH_2 \subseteq CH$ is at most ϵ. For every $v \in CH_2$, we pick an arbitrary agent of $H^{R_1}(v)$ and assign all the v-trees to this agent, which will cause the makespan to increase by at most ϵ.

Now we have derived a feasible solution R_2 of makespan $OPT+O(\epsilon)$ such that $|M^{R_2}(w)| = 1$ for every $w \in CH$. We consider $M^{R_2}(v)$ for $v \in CR$. Notice that the whole weight of each vertex remains the same throughout the above procedure, consequently it is at least ϵ for $v \in CR$. If $M^{R_2}(v)$ consists of only big agents we are done, otherwise consider those vertices where $M^{R_2}(v)$ contains at least one small agent. Let $CR_1 \subseteq CR$ be the subset of vertices where $M^{R_2}(v)$ consists of only one small agent, and $CR_2 \subseteq CR$ be the subset of vertices where $M^{R_2}(v)$ consists at least two small agents.

Notice that $|M^{R_2}(v)| \ge 2$ for $v \in CR_1$. We apply Lemma 2.3 for sets $M^{R_2}(v)$ where $v \in CR_1$, and derive disjoint sets $H^{R_2}(v) \subseteq M^{R_2}(v)$. As there is only one small agent, there are two possibilities. If the small agent is in $H^{R_2}(v)$, we simply lift its weight at v to ϵ. Otherwise all the v-trees assigned to this small agent are re-assigned to an arbitrary machine of $H^{R_2}(v)$. By doing so for every $v \in CR_1$ there are only big agents in $M(v)$, while the makespan is increased by at most ϵ.

Consider $v \in CR_2$. Let $M^{R_2}(v)' \subseteq M^{R_2}(v)$ be the subset of small agents and we have $|M^{R_2}(v)'| \ge 2$. We apply Lemma 2.3 for sets $M^{R_2}(v)'$ where $v \in CR_2$, and derive disjoint sets $H^{R_2}(v) \subseteq M^{R_2}(v)'$ and $|H^{R_2}(v)| \ge 1/3 \cdot |M(v)'|$. We match agents of $M^{R_2}(v)'$ to $H^{R_2}(v)$ in an arbitrary way such that at most 3 agents of $M^{R_2}(v)'$ is matched to one agent of $H^{R_2}(v)$. Now we re-assign the v-trees of these at most 3 agents to that machine of $H^{R_2}(v)$. After the re-assignment, all the v-trees are assigned to agents of $H^{R_2}(v)$ for $v \in CR_2$ and the weight of every agent at v is at most 4ϵ. If there is still a small agent in $H^{R_2}(v)$, we lift its weight to ϵ. As $H^{R_2}(v)$ are disjoint for $v \in CR_2$, the weight of every agent at r, and hence the makespan, increases further by at most ϵ.

Overall, we derive a new (well-ordered) solution R of makespan $OPT + O(\epsilon)$ such that for $v \in CR$, $M^R(v)$ consists of only big agents. Notice that the re-assignment procedure we applied for $v \in CR$ will influence $M(w)$ for w being a descendant of some v, however, recall that before we handle $v \in CR$, $M(w)$ consists of only one agent. Hence, after the re-assignment procedure for v, the w-tree of any agent in $M(w)$ either remains the same or is re-assigned to another agent, in both cases $M(w)$ still consists of one agent. □

2.3 Dynamic Programming We provide a dynamic programming algorithm that returns a feasible solution of makespan $OPT+O(\epsilon)$. Towards this, we first modify the given tree T in the following way. We compute $w(T(v))$ for every vertex v. Recall that the weight of every edge is at most ϵ^2. For any pair of vertices v and v' such that v is the parent of v', if $w(T(v)) \geq \epsilon$ and $w(T(v')) < \epsilon$, we delete all descendants of v', then add a new vertex v'' that is only connected to v', and define the weight of edge (v', v'') as $w(T(v'))$. According to Theorem 2.1, the makespan of the optimal solution for the modified tree is $OPT(T') \leq OPT + O(\epsilon)$. Furthermore, any feasible solution of the modified tree implies a feasible solution of T with the same makespan. Therefore in the following part of this section we focus on this modified tree. For simplicity we still denote this modified tree as T. Based on our modification procedure, we have the following lemma.

LEMMA 2.4. *In the modified tree, if v is not a leaf, then $w(T(v)) \geq \epsilon$.*

On a very high level, the dynamic programming proceeds in the following way. It iteratively calculates all possible partial solutions at each vertex v, where a partial solution is a feasible subtree cover for the subproblem on $T(v)$, i.e., it indicates whether it is possible to cover the whole subtree $T(v)$ by using m subtrees of weight $\lambda_1, \lambda_2, \cdots, \lambda_m$ respectively, where λ_i is the weight of agent i at v. Once we calculate all possible partial solutions for every child of a certain vertex v, then each partial solution of v could be calculated by merging partial solutions of v's children.

Note that, however, we are not able to store the weight of every agent in a partial solution. To handle this, the traditional method rounds the weight of each agent at each vertex. At every vertex, agents of the same rounded weight are called agents of the same type, and it stores in the partial solution the number of agents of each type. However, this idea is not applicable to our problem. It is clear that if we aim to have a constant number of different types, every agent weight has to be rounded within an error of $f(\epsilon)$, say, ϵ if we round the values to be multiples of ϵ. However, the weight of an edge could be significantly smaller, e.g., an edge weight could be $\epsilon/\log n$. The addition of any single edge weight causes a very tiny increase of the agent weight, which could not be revealed via the rounded value, however, after adding $\log n$ such edges, the agent weight increases significantly by ϵ. To overcome such a difficulty, we introduce the notions of *initial weight*, *initial distance* to r, *type* and *type weight* for agents at each vertex. All of them are defined based on any fixed solution Sol that satisfies Theorem 2.1.

Let ξ and η be the initial weight and initial distance to r (their definitions will be provided later) for an agent at v, then the type weight of this agent at v is defined as $\xi + \eta - w(v, r)$. The type of this agent at v is the pair (ξ, η). In the following we define the initial weight and initial distance to r for each agent. For simplicity we restrict our attention to agent 1.

If the subtree owned by agent 1 does not contain a vertex v, then its initial weight and initial distance to r at v are both 0. Otherwise, v is contained in the subtree owned by agent 1 and the two values are defined iteratively starting from the leaves.

Let v be an arbitrary leaf. At v, the initial weight of agent 1 is 0, the initial distance to r is $w(v, r)$ rounded up to a multiple of ϵ^2.

To define the initial weight and initial distance for agent 1 at other vertices, we introduce the notion of *initial vertex*. For any two vertices v' and v where v' is a descendant of v, we say v' is the *initial vertex* of v for agent 1 if both of the followings are true:

- Agent 1 owns a subtree that contains v', and furthermore, for any vertex x such that i). v' is the descendant of x, and ii). x is the descendant of v or $x = v$, this subtree only contains one child of x.
- Either v' is a leaf, or the subtree owned by agent 1 contains at least two children of v'.

There are two possibilities regarding the recursive definition above. If $v' \neq v$ is the initial vertex of v for agent 1, then the initial weight of agent 1 at v is its type weight at v' rounded up to a multiple of ϵ^2, and the initial distance of this agent from v to r is the rounded distance from v' to r, i.e., $w(v', r)$ rounded up to a multiple of ϵ^2. Otherwise v is the initial vertex of itself. The initial weight and initial distance to r are already defined for leaves, therefore we suppose v is not a leaf. The following observation follows directly by definition.

OBSERVATION 1. *If v is not a leaf and is the initial vertex of itself for agent 1, then the subtree owned by agent 1 contains at least 2 children of v.*

Let v_1, v_2, $\cdots$, v_k be all the children of v that are contained in the subtree of agent 1, and let (ξ_i, η_i) be the type of this agent at v_i. Then the initial weight of agent 1 at v is the following value rounded up to a multiple of ϵ^2:

$$\sum_{i=1}^{k} [\xi_i + \eta_i - w(v_i, r) + w_{(v_i,v)}], \tag{2.3}$$

and the initial distance from v to r is $w(v, r)$ rounded up to a multiple of ϵ^2.

It is easy to see that the type of agent 1 remains the same along the path from v' to v if v' is the initial vertex of v for agent 1.

Using this method, given a solution, we can compute the type weight of each agent at the root r. This weight may be different from the exact weight, however, we can prove that they differ by at most $O(\epsilon)$. Thus, it suffices to store the type of agents at each vertex and our algorithm follows.

Estimation of Rouding Errors. In the following we focus on an arbitrarily fixed agent, say, agent 1. Let $E(x)$ be the rounding error of agent 1's weight at an arbitrary vertex x, i.e., its type weight minus its exact weight at x. Notice that we always round up values, therefore $E(x) \geq 0$.

LEMMA 2.5. *If v' is the initial vertex of v, then $E(v) = E(v')$.*

Proof. Let (ξ, η) be the type of agent 1 at v', which is also its type at v. According to the definition of the initial vertex, the exact weight of agent 1 at v is its exact weight at v' plus $w(v, v')$. Meanwhile, as the type of agent 1 remains the same from v' to v, the type weight of agent 1 at v equals its type weight at v' plus $w(v, v')$. Therefore, $E(v) = E(v')$. □

According to Equation (2.3), we have the following lemma.

LEMMA 2.6. *If the subtree of agent 1 contains vertex v together with its $k \geq 2$ children $v_1, v_2, \cdots, v_k$, then $E(v) \leq \sum_{i=1}^{k} E(v_i) + \epsilon^2$.*

Consider the subtree owned by agent 1. Let LF be the set of all leaves and $LF' = \{v | v \text{ is the parent of } v' \text{ for some } v' \in LF\}$. We define a function $\phi(v)$ for each vertex v such that

- $\phi(v) = 1$ if $v \in LF$;
- $\phi(v) = 2$ if $v \in LF'$;
- $\phi(v) = \sum_{i=1}^{k} \phi(v_i)$ if $v \notin LF \cup LF'$ and $v_1, v_2, \cdots, v_k$ are all the children of v in the subtree.

LEMMA 2.7. *$E(v) \leq (2\phi(v) - 1)\epsilon^2$.*

Proof. We prove by induction. It is easy to see that the lemma is true if $v \in LF$ or $v \in LF'$. Suppose $v_1, v_2, \cdots, v_k$ are all the children of v in the subtree of agent 1, and lemma holds for every v_i. If $k = 1$, then v and v_1 have the same initial vertex, by Lemma 2.5, $E(v) = E(v_1) \leq (2\phi(v_1) - 1)\epsilon^2 = (2\phi(v) - 1)\epsilon^2$. Otherwise $k \geq 2$, $E(v) \leq \sum_{i=1}^{k} E(v_i) + \epsilon^2 \leq \epsilon^2 \sum_{i=1}^{k}(2\phi(v_i) - 1) + \epsilon^2 \leq (2\phi(v) - 1)\epsilon^2$. □

Consider the substree of agent 1 and let if be T_1. We have the following lemma.

LEMMA 2.8. *$w(T_1(v)) \geq \phi(v)\epsilon/2$ for $v \notin LF$.*

Proof. We prove by induction. Consider $v \in LF'$. By Lemma 2.4 we know $w(T(v)) \geq \epsilon$. According to Theorem 2.1, in Sol every agent in $M(v)$ is big, therefore $w(T_1(v)) \geq \epsilon = \phi(v)\epsilon/2$. Suppose $v \notin LF \cup LF'$ and the lemma holds for all the children of v in T_1, namely $v_1, v_2, \cdots, v_k$. Then $w(T_1(v)) \geq \sum_{i=1}^{k} w(T_1(v_i)) \geq \epsilon/2 \cdot \sum_{i=1}^{k} \phi(v_i) = \phi(v)\epsilon/2$. □

Notice that $w(T_1(v)) \leq OPT + O(\epsilon) \leq 1 + O(\epsilon)$, we conclude that $\phi(v) = O(1/\epsilon)$, which implies that $E(v) = O(\epsilon)$.

The algorithm Notice that for any agent type (ξ, η), we have $\xi, \eta \leq 1 + O(\epsilon)$ and they are multiples of ϵ^2. Hence there are $q = O(1/\epsilon^2)$ different kinds of types. We order these types arbitrarily and define a configuration as a q-tuple $(\omega_1, \omega_2, \cdots, \omega_q)$, where $\omega_i \leq m$ is the number of agents of type i.

We say a configuration $(\omega_1, \omega_2, \cdots, \omega_q)$ is feasible at vertex v, if it is possible to cover the subtree $T(v)$ using ω_i agents of type i. We will iteratively calculate a set of feasible configurations for each vertex v, whereas a near optimal solution Sol could be found.

Initial states. As we already know the type of a single agent at every leaf in Sol, we store the corresponding configuration.

Recursive calculation. Let v_1, v_2, $\cdots$, v_k be all the children of v. Suppose we have calculated the set of feasible configurations for each v_i and let it be $C(v_i)$. We now calculate $C(v)$. We first calculate $C(v_1) + C(v_2)$, i.e., for every possible configuration $(\omega_1, \omega_2, \cdots, \omega_q)$, we check if it could be achieved by merging one configuration of $C(v_1)$ and one configuration of $C(v_2)$. Then we calculate $(C(v_1) + C(v_2)) + C(v_3)$, and so on.

To check if $(\omega_1, \omega_2, \cdots, \omega_q)$ could be achieved by merging one configuration of $C(v_1)$ and one configuration of $C(v_2)$, we check the merging of every feasible configuration in C_1 and every feasible configuration in $C(v_2)$. Let $(\omega_1(1), \omega_2(1), \cdots, \omega_q(1))$ and $(\omega_1(2), \omega_2(2), \cdots, \omega_q(2))$ be two arbitrary feasible configurations in $C(v_1)$ and $C(v_2)$, respectively. If the same agent has type i in the first configuration and type i' in the second configuration, then by merging the subtrees of the agent at v_1 and v_2 we get a new type of this agent, which could be calculated via formula (2.3). Hence, to see which configurations could be achieved by merging the two configurations, we try all possible mergings, i.e., Any agent of type i in the first configuration could be merged with any agent of type i' in the second configuration. We estimate the number of such mergings. Indeed, for any agent of type i in the first configuration, it has $q+1$ different choices, i.e., to merge with one type in the latter configuration or not to merge with any type. Given $\omega_i(1)$ agents of type i, the number of agents merged with type i' (or not merged with anyone) is at most $\omega_i(1)$, hence the overall possibilities are bounded by $\omega_i(1)^{(q+1)} = m^{O(q)}$. Combining the possibilities for every type in the first configuration, the overall possibilities are bounded by $m^{O(q^2)}$, i.e., $C(V_1) + C(V_2)$ could be calculated in $m^{O(q^2)}$ time. In a similar way we can calculate $C(V_1) + \cdots + C(V_k)$.

Overall, the algorithm can find a near optimal solution within $m^{O(q^2)} = m^{O(1/\epsilon^4)}$, and Theorem 1.1 follows.

3 The FPT algorithm

In this section, we show that the subtree cover problem is FPT parameterized by the makespan. Towards this, we formulate the problem as an ILP. We observe that the ILP we establish has a special structure, which generalizes the n-fold integer programming studied in the literature. We call it as a tree-fold integer programming. Indeed, when the input tree is a star, the tree-fold integer program we formulate becomes an n-fold integer program. We extend the FPT algorithm for the n-fold integer programming to derive an FPT algorithm for the tree-fold integer programming, which implies an FPT algorithm for the subtree cover problem. This result may be of separate interest.

Recall that when the given graph is a star, the subtree cover problem becomes FPT parameterized by the largest edge weight $w_{max} = \max_j\{w_j|1 \le j \le n\}$ [17]. However, this is no longer true even if the given graph is a tree of height 2, as is implied by the following theorem.

THEOREM 3.1. *The subtree cover problem remains NP-hard even if the given tree is of height* 2 *and every edge has unit weight.*

Proof. We reduce from 3-partition. In the 3-partition problem, given is a set of $3n$ integers $a_1, a_2, \cdots, a_{3n}$ with $B/4 < a_j < B/2$, $\sum_j a_j = 3nB$ where $B = n^{O(1)}$. The goal is to determine whether we can partition the $3n$ integers of n subsets $D_1, D_2, \cdots, D_n$, each of size 3, such that $\sum_{a_j \in D_i} a_j = B$ for every $1 \le i \le n$.

We construct a subtree cover instance as follows. There is a root r. The root has $3n$ children $v_1, v_2, \cdots, v_{3n}$. Each v_j further has a_j children. We let the weight of every edge be 1.

We show that the constructed subtree cover instance can be covered by n subtrees of makespan $B+3$ if and only if the given 3-partition instance admits a feasible partition.

Suppose the 3-partition instance admits a feasible partition, then each subtree consists of the root, $\{v_j|a_j \in S_i\}$ and their children. It is easy to verify that the weight of each subtree is exactly $B+3$.

Suppose the subtree cover instance admits a solution of makespan $B+3$. Since all edge weights sum up to $nB+3n$, we know each subtree consists of exactly $B+3$ edges, and each edge appears in one subtree. Therefore, if a subtree contains a vertex v_j, it must contain all the children of v_j. As v_j has $B/4 < a_j < B/2$ children, it is easy to see that each subtree contains exactly 3 children of the root, implying readily a solution for the 3-partition instance. □

The above hardness result excludes FPT algorithms parameterized by edge weight and tree height, and therefore we restrict our attention to makespan. We will first show that a tree-fold integer programming can be solved in FPT time. Then we establish a configuration ILP for the subtree cover problem and prove that the ILP falls exactly into the category of tree-fold integer programming, and is thus solvable in FPT time.

3.1 Tree-fold integer programming

The goal of this and next subsection is to prove Theorem 1.3. Towards this, we first introduce some basic concepts and techniques which are crucial for our proof.

3.2 Preliminaries for Tree-fold Integer Programming

We provide a brief introduction to the notions needed for solving a general integer programming. We refer the readers to a nice book [2] for details.

We define *Graver basis*, which was introduced in [4] by Graver and is crucial for our algorithm.

We define a partial order $\sqsubseteq$ in $\mathbb{R}^n$ in the following way:

For any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} \sqsubseteq \mathbf{y}$ if and only if for every $1 \le i \le n, |x_i| \le |y_i|$ and $x_i \cdot y_i \ge 0$.

Roughly speaking, $\mathbf{x} \sqsubseteq \mathbf{y}$ implies that $\mathbf{x}$ and $\mathbf{y}$ lie in the same orthant, and $\mathbf{x}$ is "closer" to the origin 0 than $\mathbf{y}$. The partial order $\sqsubseteq$, when restricted to $\mathbb{R}^n_+$, coincides with the classical coordinate-wise partial order $\le$.

Given any subset $X \subseteq \mathbb{R}^n$, we say $\mathbf{x}$ is an $\sqsubseteq$-minimal element of X if $\mathbf{x} \in X$ and there does not exist $\mathbf{y} \in X$, $\mathbf{y} \ne \mathbf{x}$ such that $\mathbf{y} \sqsubseteq \mathbf{x}$.

According to Gordan's Lemma, for any subset $Z \subseteq \mathbb{Z}^n$, the number of $\sqsubseteq$-minimal elements in Z is finite. Indeed, this fact is known as Dickson's Lemma for the coordinate-wise partial order $\preceq$.

Definition 1. *The Graver basis of an integer $m \times n$ matrix A is the finite set $\mathcal{G}(A) \subseteq \mathbb{Z}^n$ which consists of all the $\sqsubseteq$-minimal elements of $ker_{\mathbb{Z}^n}(A) = \{\mathbf{x} \in \mathbb{Z}^n | A\mathbf{x} = 0, \mathbf{x} \ne 0\}$.*

The Graver basis $\mathcal{G}(A)$ is only dependent on A. Let $||B||_\infty$ be the largest absolute value over all entries. For any $\mathbf{g} \in \mathcal{G}(A)$, we have the following rough estimation for some constant c_1, c_2 [19]:

$$|\mathcal{G}(A)| \le (c_1||A||_\infty)^{mn} \quad \text{and } ||g||_\infty \le (c_2||A||_\infty)^{mn}.$$

The Graver basis has the following *positive sum property*: for every $\mathbf{z} \in ker_{\mathbb{Z}^n}(A)$, there exist a subset $U \subseteq \mathcal{G}(A)$ such that for every $\mathbf{g}_i \in U$, $\mathbf{g}_i \sqsubseteq \mathbf{z}$, and furthermore, $\mathbf{z} = \sum_{\mathbf{g}_i \in U} \alpha_i \mathbf{g}_i$ for some $\alpha_i \in \mathbb{Z}_+$. See [2,19] for details.

Given is an integer programming of the following form:

$$(3.4) \quad \min\{\mathbf{c}^T\mathbf{x} | A\mathbf{x} = \mathbf{b}, \mathbf{l} \le \mathbf{x} \le \mathbf{u}, \mathbf{x} \in \mathbb{Z}^n\}.$$

Let $\mathbf{x}$ be an arbitrary feasible solution of (3.4). We say $\mathbf{q}$ is an *augmentation vector* for $\mathbf{x}$ if $\mathbf{x}+\mathbf{q}$ is a feasible solution of (3.4) that has an objective value strictly better than $\mathbf{x}$, i.e., $\mathbf{c}^T(\mathbf{x}+\mathbf{q}) < \mathbf{c}^T\mathbf{x}$. Therefore, $A\mathbf{q} = 0$ and $\mathbf{c}^T\mathbf{q} < 0$.

It is shown by Graver [4] that $\mathbf{x}^*$ is an optimal solution of (3.4) if and only if there does not exist $\mathbf{g} \in \mathcal{G}(A)$ which is an augmentation vector for $\mathbf{x}^*$. Later on, Hemmecke, Onn and Weismantel [6] proved that, starting from an arbitrary feasible solution x_0 for (3.4), the optimal solution $\mathbf{x}^*$ could be achieved by iteratively applying the *best augmentation via Graver basis*, i.e., augmenting $\mathbf{x}$ by using the best possible augmentation vector of the form $\gamma\mathbf{g}$, where $\gamma \in \mathbb{Z}_+$ and $\mathbf{g} \in \mathcal{G}(A)$. The total number of augmentation vectors needed is bounded by $O(nL)$, where L is the length of the binary encoding of the vector $(\mathbf{c}, \mathbf{b}, \mathbf{l}, \mathbf{u})$ (There may exist an augmentation vector which is better than any Graver basis, however, the result of [6] allows us to restrict our attention to Graver basis). This statement remains true if, instead of choosing the best possible augmentation vector of the form $\gamma\mathbf{g}$, say, $\gamma^*\mathbf{g}^*$, we choose an augmentation vector $\mathbf{q}$ which is at least as good as $\gamma^*\mathbf{g}^*$. That is, if in each augmentation vector we choose an augmentation vector $\mathbf{q}$ such that $\mathbf{c}^T\mathbf{q} \le \gamma^*\mathbf{c}^T\mathbf{g}^*$, the optimal solution $\mathbf{x}^*$ could also be achieved after $O(nL)$ augmentation vectors [2,5]. Notice that $\mathbf{q}$ does not necessarily belong to $\mathcal{G}(A)$. Such greedy algorithm is called *Graver-best augmentation algorithm.*

The results by Hemmecke et al. [2,5] imply that, to design a polynomial time algorithm for (3.4), it suffices to handle the following two problems:

a. finding a feasible initial solution for (3.4) in polynomial time;

b. finding a Graver-best augmentation algorithm that runs in polynomial time.

In Subsection 3.4 we show in detail how to find a feasible initial solution for (3.4) in polynomial time. Roughly speaking this could be handled by establishing another ILP with a trivial initial feasible solution and finding its optimal solution.

We focus on problem [b]. A natural algorithm is that, given the current feasible solution $\mathbf{x}$, for every $\mathbf{g} \in \mathcal{G}(A)$, we find integer $\gamma_\mathbf{g} \in \mathbb{Z}_+$ such that $\mathbf{x} + \gamma_\mathbf{g}\mathbf{g}$ is still feasible and $\mathbf{c}^T(\mathbf{x} + \gamma_\mathbf{g}\mathbf{g})$ is minimized, and among all the $\gamma_\mathbf{g}\mathbf{g}$ we pick the best one. For any fixed $\mathbf{g}$ we can easily find $\gamma_\mathbf{g}$ by solving an integer programming with only one integral variable $\gamma_\mathbf{g}$. Therefore the overall running time depends on the cardinality of the Graver bais $\mathcal{G}(A)$. Unfortunately $|\mathcal{G}(A)|$ could be huge in general. However, if the matrix A has some special structure, then $|\mathcal{G}(A)|$ could be significantly smaller.

From now on we focus on a tree-fold matrix A consisting of n copies of submatrices A_1, A_2, $\cdots$, A_τ and write it as $A = T[A_1, A_2, \cdots, A_\tau]$ for simplicity. Recall that each A_i is an $s_i \times t$-matrix, whereas we are restricting to the following

$$(3.5) \quad \min\{\mathbf{c}^T\mathbf{x} | A\mathbf{x} = \mathbf{b}, \mathbf{l} \le \mathbf{x} \le \mathbf{u}, \mathbf{x} \in \mathbb{Z}^{nt}\}.$$

Notice that if $\tau = 2$, A is called an n-fold matrix. In 2013, Hemmecke et al. provided a Graver-best augmentation algorithm for n-fold integer programming that runs in $O(n^3L)$ time (here the big-O hides all coefficients that only depend on A_1 and A_2). The following lemma is the key ingredient to their algorithm. It strengthens the *fitness theorem* in [8].

Consider any $\mathbf{x} \in \mathbb{Z}^{nt}$. We write $\mathbf{x}$ as a tuple $\mathbf{x} = (\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^n)$ where $\mathbf{x}^i \in \mathbb{Z}^t$. Each $\mathbf{x}^i$ is called a *brick* of $\mathbf{x}$.

LEMMA 3.1. ([5]) *Let $A = T[A_1, A_2]$. There exists some integer $\lambda = \lambda(A_1, A_2)$ that only depends on matrices A_1 and A_2, and*

$H(A) = \{\mathbf{h} \in \mathbb{Z}^t | \mathbf{h}$ *is the sum of at most λ elements of* $\mathcal{G}(A_2)\}$,

such that for any $\mathbf{g} = (\mathbf{g}^1, \mathbf{g}^2, \cdots, \mathbf{g}^n) \in \mathcal{G}(A)$ we have $\sum_{i \in I} \mathbf{g}^i \in H(A)$ for any $I \subseteq \{1, 2, \cdots, n\}$.

We further generalize the algorithm of Hemmecke et al. [5] to tree-fold integer programming. Towards this, we first give a generalization of the above lemma, and then we show how to further generalize their algorithm.

LEMMA 3.2. *Let $A = T[A_1, A_2, \cdots, A_\tau]$. There exists some integer $\lambda = \lambda(A_1, A_2, \cdots, A_\tau)$ that only depends on matrices A_1 A_2, $\cdots$, A_τ, and*

$H(A) = \{\mathbf{h} \in \mathbb{Z}^t | \mathbf{h}$ *is the sum of at most λ elements of* $\mathcal{G}(A_\tau)\}$,

such that for any $\mathbf{g} = (\mathbf{g}^1, \mathbf{g}^2, \cdots, \mathbf{g}^n) \in \mathcal{G}(A)$ we have $\sum_{i \in I} \mathbf{g}^i \in H(A)$ for any $I \subseteq \{1, 2, \cdots, n\}$.

Here $A = T[A_1, A_2, \cdots, A_\tau]$ means A is a tree-fold matrix consisting of A_1, $\cdots$, A_τ. Roughly speaking, Lemma 3.2 states that for any Graver basis element $\mathbf{g}$ of the matrix A, although it is of a very high dimension, it is sparse, i.e., among the n bricks $\mathbf{g}^1, \mathbf{g}^2, \cdots, \mathbf{g}^n$, only an "FPT" number of them can be nonzero. This lemma extends the structural lemma for n-fold integer programming in [5], which can be viewed as the case when $\tau = 2$.

Proof. [of Lemma 3.2] Throughout this proof, for an arbitrary matrix B, we list its Graver bases (in an arbitrary order) as $\mathbf{g}^1(B), \mathbf{g}^2(B), \cdots, \mathbf{g}^{|\mathcal{G}(B)|}(B)$, and let $\mathbf{G}(B) = (\mathbf{g}^1(B), \mathbf{g}^2(B), \cdots, \mathbf{g}^{|\mathcal{G}(B)|}(B))$ be the matrix with each of the bases being its column.

Consider A_τ. For any $\mathbf{g} = (\mathbf{g}^1, \mathbf{g}^2, \cdots, \mathbf{g}^n) \in \mathcal{G}(A)$, it follows directly that $A_\tau \mathbf{g}^i = 0$ for every $1 \le i \le n$. According to the positive sum property of the Graver basis, there exist $q^i_j(A_\tau) \in \mathbb{Z}_{\ge 0}$ such that

$$\begin{aligned} (3.6) \quad \mathbf{g}^i &= \sum_{j=1}^{|\mathcal{G}(A_\tau)|} q^i_j(A_\tau) \mathbf{g}^j(A_\tau) \\ &= \mathbf{G}(A_\tau) \mathbf{q}^i(A_\tau), \quad \forall 1 \le i \le d_\tau = n \end{aligned}$$

where $\mathbf{q}^i(A_\tau) = (q^i_1(A_\tau), \cdots, q^i_{|\mathcal{G}(A_\tau)|})^T$. Notice that $\mathbf{g}(A_\tau)$ is only dependent on matrix A_τ. In order to show that $\sum_{i \in I} \mathbf{g}^i \in H(A)$ for some λ, it suffices to show that $\sum_i ||\mathbf{q}^i(A_\tau)||_1 = \sum_{i,j} |q^i_j(A_\tau)|$ is upper bounded by some value that only depends on $A_1, A_2, \cdots, A_\tau$.

Step 1. We consider $A_{\tau-1}$. According to $A\mathbf{g} = 0$, we have

$$\sum_{i \in S^\ell_{\tau-1}} A_{\tau-1} \mathbf{g}^i = 0, \quad \forall 1 \le \ell \le d_{\tau-1}$$

Plugging in Equation 3.6, we have

$$(3.7) \sum_{i \in S^\ell_{\tau-1}} A_{\tau-1} \mathbf{G}(A_\tau) \mathbf{q}^i(A_\tau) = 0. \quad \forall 1 \le \ell \le d_{\tau-1}$$

We rewrite the above equation in the following way. Let matrix $A'_{\tau-1} = A_{\tau-1}\mathbf{G}(A_\tau)$, $\mathbf{Q}^\ell(A_\tau) = \sum_{i \in S^\ell_{\tau-1}} \mathbf{q}^i(A_\tau)$, we have

$$(3.8) \sum_{i \in S^\ell_{\tau-1}} \mathbf{g}^i = \mathbf{G}(A_\tau) \mathbf{Q}^\ell(A_\tau), \quad \forall 1 \le \ell \le d_{\tau-1}$$

$$(3.9) \quad A'_{\tau-1} \mathbf{Q}^\ell(A_\tau) = 0, \quad \forall 1 \le \ell \le d_{\tau-1}$$

Therefore, $\mathbf{Q}^\ell(A_\tau) \in ker_{\mathbb{Z}^{|\mathcal{G}(A_\tau)|}}(A'_{\tau-1})$. We replace the index ℓ by i. According to the positive sum property, we list the Graver basis of $A'_{\tau-1}$ as $\mathbf{g}^1(A'_{\tau-1})$, $\cdots$, $\mathbf{g}^{|\mathcal{G}(A'_{\tau-1})|}(A'_{\tau-1})$, then there exist $q^i_j(A'_{\tau-1}) \in \mathbb{Z}_{\ge 0}$ such that

$$\begin{aligned} (3.10) \mathbf{Q}^i(A_\tau) &= \sum_{j=1}^{|\mathcal{G}(A'_{\tau-1})|} q^i_j(A'_{\tau-1}) \mathbf{g}^j(A'_{\tau-1}) \\ &= \mathbf{G}(A'_{\tau-1}) \mathbf{q}^i(A'_{\tau-1}), \quad \forall 1 \le i \le d_{\tau-1} \end{aligned}$$

where $\mathbf{q}^i(A'_{\tau-1}) = (q^i_1(A'_{\tau-1}), \cdots, q^i_{|\mathcal{G}(A'_{\tau-1})|})^T$. Furthermore, as every entry of $\mathbf{Q}^i(A_\tau)$ is non-negative, the positive sum property ensures that $q^i_j(A'_{\tau-1}) > 0$ only if every entry of $\mathbf{g}^j(A'_{\tau-1})$ is non-negative.

Step 2. We consider $A_{\tau-2}$. According to $A\mathbf{g} = 0$, we have

$$\sum_{i_1 \in S^\ell_{\tau-2}} \sum_{i_0 \in S^{i_1}_{\tau-1}} A_{\tau-2} \mathbf{g}^{i_0} = 0, \quad \forall 1 \le \ell \le d_{\tau-2}.$$

Plugging in Equation 3.8 and 3.10, we have

$$\begin{aligned} &\sum_{i_1 \in S^\ell_{\tau-2}} \sum_{i_0 \in S^{i_1}_{\tau-1}} A_{\tau-2} \mathbf{g}^{i_0} \\ = &\sum_{i_1 \in S^\ell_{\tau-2}} A_{\tau-2} \mathbf{G}(A_\tau) \mathbf{Q}^{i_1}(A_\tau) \\ = &\sum_{i_1 \in S^\ell_{\tau-2}} A_{\tau-2} \mathbf{G}(A_\tau) \mathbf{G}(A'_{\tau-1}) \mathbf{q}^{i_1}(A'_{\tau-1}) \\ = &\ 0, \quad \forall 1 \le \ell \le d_{\tau-2} \end{aligned}$$

Let $A'_{\tau-2} = A_{\tau-2}\mathbf{G}(A_\tau)\mathbf{G}(A'_{\tau-1})$, $\mathbf{Q}^\ell(A'_{\tau-1}) = \sum_{i_1\in S^\ell_{\tau-2}} \mathbf{q}^{i_1}(A'_{\tau-1})$, we have

$$\begin{aligned}(3.11)\quad &\sum_{i_1\in S^\ell_{\tau-2}}\sum_{i_0\in S^{i_1}_{\tau-1}} \mathbf{g}^{i_0}\\ &= \mathbf{G}(A_\tau)\mathbf{G}(A'_{\tau-1})\mathbf{Q}^\ell(A'_{\tau-1}), \quad \forall 1\le \ell\le d_{\tau-2}\end{aligned}$$

$$(3.12)\quad A'_{\tau-2}\mathbf{Q}^\ell(A'_{\tau-1}) = 0, \quad \forall 1\le\ell\le d_{\tau-2}$$

Therefore, $\mathbf{Q}^\ell(A'_{\tau-1}) \in ker_{\mathbb{Z}^{|\mathcal{G}(A_{\tau-1})'|}}(A'_{\tau-2})$. Replacing the index ℓ by i, there exist $q^i_j(A'_{\tau-2})\in\mathbb{Z}_{\ge 0}$ such that

$$\begin{aligned}(3.13)\quad &\mathbf{Q}^i(A'_{\tau-1})\\ &= \sum_{j=1}^{|\mathcal{G}(A'_{\tau-2})|} q^i_j(A'_{\tau-2})\mathbf{g}^j(A'_{\tau-2})\\ &= \mathbf{G}(A'_{\tau-2})\mathbf{q}^i(A'_{\tau-2}), \quad \forall 1\le i\le d_{\tau-2}\end{aligned}$$

where $\mathbf{q}^i(A'_{\tau-2}) = (q^i_1(A'_{\tau-2}),\cdots,q^i_{|\mathcal{G}(A'_{\tau-2})|})^T$.

We can iteratively carry on the above argument.

Step $\tau-k$. In general, suppose we have shown the following three equations:

$$\begin{aligned}(3.14)\quad &\sum_{i_{\tau-k-2}\in S^\ell_{k+1}}\sum_{i_{\tau-k-3}\in S^{i_{\tau-k-2}}_{k+2}}\cdots\sum_{i_0\in S^{i_1}_{\tau-1}} \mathbf{g}^{i_0}\\ &= \mathbf{G}(A_\tau)\mathbf{G}(A'_{\tau-1})\cdots\mathbf{G}(A'_{k+2})\mathbf{Q}^\ell(A'_{k+2}),\\ &\forall \quad 1\le\ell\le d_{k+1}\end{aligned}$$

$$(3.15)\quad A'_{k+1}\mathbf{Q}^\ell(A'_{k+2}) = 0, \quad \forall 1\le\ell\le d_{k+1}$$

Replacing the index ℓ by i, there exist $q^i_j(A'_{k+1})\in\mathbb{Z}_{\ge0}$ such that

$$\begin{aligned}(3.16)\quad &\sum_{i'\in S^i_{k+1}} \mathbf{q}^{i'}(A'_{k+2})\\ &= \mathbf{Q}^i(A'_{k+2})\\ &= \sum_{j=1}^{|\mathcal{G}(A'_{k+1})|} q^i_j(A'_{k+1})\mathbf{g}^j(A'_{k+1})\\ &= \mathbf{G}(A'_{k+1})\mathbf{q}^i(A'_{k+1}), \quad \forall 1\le i\le d_{k+1}\end{aligned}$$

where $\mathbf{q}^i(A'_{k+1}) = (q^i_1(A'_{k+1}),\cdots,q^i_{|\mathcal{G}(A'_{k+1})|})^T$, and $A'_{k+1} = A_{k+1}\mathbf{G}(A_\tau)\mathbf{G}(A'_{\tau-1})\cdots\mathbf{G}(A'_{k+2})$.

When we consider A_k, $A\mathbf{g}=0$ implies that

$$\begin{aligned}&\sum_{i_{\tau-k-1}\in S^\ell_k}\sum_{i_{\tau-k-2}\in S^{\tau-k-1}_{k+1}}\sum_{i_{\tau-k-3}\in S^{i_{\tau-k-2}}_{k+2}}\cdots\sum_{i_0\in S^{i_1}_{\tau-1}} A_k\mathbf{g}^{i_0}\\ &= 0, \quad \forall 1\le\ell\le d_k.\end{aligned}$$

Indeed, if we view each $\mathbf{g}^i$ as the i-th leaf (from left to right) of the tree, the summation is taken over all the leaves of the sub-tree routed at the vertex corresponding to S^ℓ_k. Plugging Equation 3.14 and Equation 3.16 into the above equation, and replacing index $i_{\tau-k-1}$ by i, we have the following

$$\begin{aligned}&\sum_{i\in S^\ell_k} A_k\mathbf{G}(A_\tau)\mathbf{G}(A'_{\tau-1})\cdots\mathbf{G}(A'_{k+2})\mathbf{G}(A'_{k+1})\mathbf{q}^i(A'_{k+1})\\ &= 0, \quad \forall 1\le\ell\le d_k\end{aligned}$$

Let $A'_k = A_k\mathbf{G}(A_\tau)\mathbf{G}(A'_{\tau-1})\cdots\mathbf{G}(A'_{k+1})$, $\mathbf{Q}^\ell(A'_{k+1}) = \sum_{i'\in S^\ell_k}\mathbf{q}^{i'}(A'_{k+1})$, we have

$$\begin{aligned}(3.17)\quad &\sum_{i_{\tau-k-1}\in S^\ell_k}\sum_{i_{\tau-k-2}\in S^{\tau-k-1}_{k+1}}\cdots\sum_{i_0\in S^{i_1}_{\tau-1}} \mathbf{g}^{i_0}\\ &= \mathbf{G}(A_\tau)\mathbf{G}(A'_{\tau-1})\cdots\mathbf{G}(A'_{k+1})\mathbf{Q}^\ell(A'_{k+1}),\\ &\forall \quad 1\le\ell\le d_k\end{aligned}$$

$$(3.18)\quad A'_k\mathbf{Q}^\ell(A'_{k+1}) = 0, \quad \forall 1\le\ell\le d_k$$

Therefore, $\mathbf{Q}^\ell(A'_{k+1}) \in ker_{\mathbb{Z}^{|\mathcal{G}(A_{k+1})'|}}(A'_k)$. Replacing the index ℓ by i, there exist $q^i_j(A'_k)\in\mathbb{Z}_{\ge0}$ (by the positive sum property) such that

$$\begin{aligned}(3.19)\quad &\sum_{i'\in S^i_k}\mathbf{q}^{i'}(A'_{k+1})\\ &= \mathbf{Q}^i(A'_{k+1})\\ &= \sum_{j=1}^{|\mathcal{G}(A'_k)|} q^i_j(A'_k)\mathbf{g}^j(A'_k)\\ &= \mathbf{G}(A'_k)\mathbf{q}^i(A'_k), \quad \forall 1\le i\le d_k\end{aligned}$$

where $\mathbf{q}^i(A'_k) = (q^i_1(A'_k),\cdots,q^i_{|\mathcal{G}(A'_k)|})^T$, and $A'_k = A_k\mathbf{G}(A_\tau)\mathbf{G}(A'_{\tau-1})\cdots\mathbf{G}(A'_{k+1})$.

Specifically, we let $A'_\tau = A_\tau$, therefore the above equalities hold for any $1\le k\le\tau-1$.

Step $\tau-1$. Eventually we consider A_1 and derive the following based on the iterative argument.

$$\begin{aligned}(3.20)\quad &\sum_{i_{\tau-2}\in S^\ell_1}\sum_{i_{\tau-3}\in S^{i_{\tau-2}}_2}\cdots\sum_{i_0\in S^{i_1}_{\tau-1}} \mathbf{g}^{i_0}\\ &= \mathbf{G}(A_\tau)\mathbf{G}(A'_{\tau-1})\cdots\mathbf{G}(A'_2)\mathbf{Q}^\ell(A'_2),\\ &\forall \quad 1\le\ell\le d_1 = 1\end{aligned}$$

$$(3.21)\quad A'_1\mathbf{Q}^\ell(A'_2) = 0, \quad \forall 1\le\ell\le d_1 = 1$$

Replacing the index ℓ by i, there exist $q^i_j(A'_1)\in\mathbb{Z}_{\ge0}$

such that

$$\begin{aligned} (3.22) \qquad & \sum_{i' \in S_1^i} \mathbf{q}^{i'}(A_2') \\ = \; & \mathbf{Q}^i(A_2') \\ = \; & \sum_{j=1}^{|\mathcal{G}(A_1')|} q_j^i(A_1')\mathbf{g}^j(A_1') \\ = \; & \mathbf{G}(A_1')\mathbf{q}^i(A_1'), \quad \forall 1 \le i \le d_1 = 1 \end{aligned}$$

where $\mathbf{q}^i(A_1') = (q_1^i(A_1'), \cdots, q^i_{|\mathcal{G}(A_1')|})^T$, and $A_1' = A_1\mathbf{G}(A_\tau)\mathbf{G}(A_{\tau-1}')\cdots\mathbf{G}(A_2')$.

We make the following claim.

Claim 1. $\mathbf{Q}^i(A_2') \in \mathcal{G}(A_1')$.

Proof. [of the Claim] Suppose on the contrary that $\mathbf{Q}^i(A_2') \notin \mathcal{G}(A_1')$, then there exist $0 \ne \bar{\mathbf{Q}}^i(A_2') \sqsubset \mathbf{Q}^i(A_2')$ such that $A_1'\bar{\mathbf{Q}}^i(A_2') = 0$. In the following we will construct $0 \ne \bar{\mathbf{g}} \sqsubset \mathbf{g}$ such that $A\bar{\mathbf{g}} = 0$, which contradicts the fact that $\mathbf{g} \in \mathcal{G}(A)$. Hence, the claim is true.

We show how to construct $\bar{\mathbf{g}}$. According to Equation 3.22, $\sum_{i'\in S_1^i}\mathbf{q}^{i'}(A_2') = \mathbf{Q}^i(A_2')$. We know that every entry of $\mathbf{q}^{i'}(A_2')$, and consequently $\mathbf{Q}^i(A_2')$, is non-negative. Therefore every entry of $\bar{\mathbf{Q}}^i(A_2')$ is also non-negative. Consider every entry of the equation $\sum_{i'\in S_1^i}\mathbf{q}^{i'}(A_2') = \mathbf{Q}^i(A_2')$, we have $\sum_{i'\in S_1^i}\mathbf{q}_j^{i'}(A_2') = \mathbf{Q}_j^i(A_2')$. For $0 \le \bar{\mathbf{Q}}_j^i(A_2') \le \mathbf{Q}_j^i(A_2')$, we can easily find $0 \le \bar{\mathbf{q}}_j^{i'}(A_2') \le \mathbf{q}_j^{i'}(A_2')$ such that $\sum_{i'\in S_1^i}\bar{\mathbf{q}}_j^{i'}(A_2') = \bar{\mathbf{Q}}_j^i(A_2')$. Hence, there exist $\bar{\mathbf{q}}^{i'}(A_2') \sqsubseteq \mathbf{q}^{i'}(A_2')$ such that

$$\sum_{i'\in S_1^i}\bar{\mathbf{q}}^{i'}(A_2') = \bar{\mathbf{Q}}_j^i(A_2'), \quad \forall 1 \le i \le d_1 = 1,$$

and moreover, there exist some i_1' and i_2' such that $\bar{\mathbf{q}}^{i_1'}(A_2') \sqsubset \mathbf{q}^{i_1'}(A_2')$ and $\bar{\mathbf{q}}^{i_2'}(A_2') \ne 0$.

Replacing i' with i, we define

$$\bar{\mathbf{Q}}^i(A_3') = \mathbf{G}(A_2')\bar{\mathbf{q}}^i(A_2'), \quad 1 \le i \le d_2.$$

It is easy to see that $\bar{\mathbf{Q}}^i(A_3') \sqsubseteq \mathbf{Q}^i(A_3')$ for $1 \le i \le d_2$. As each $\bar{\mathbf{Q}}^i(A_3')$ is the weighted sum of the Graver basis of A_2', we know $A_2'\bar{\mathbf{Q}}^i(A_3') = 0$. Furthermore, there exist $1 \le i_1, i_2 \le d_2$ such that $\bar{\mathbf{Q}}^{i_1}(A_3') \sqsubset \mathbf{Q}^{i_1}(A_3')$ and $\bar{\mathbf{Q}}^{i_2}(A_3') \ne 0$.

Carry on the above argument, we can prove iteratively that there exist $\bar{\mathbf{q}}^{i'}(A_{k+1}') \sqsubseteq \mathbf{q}^{i'}(A_{k+1}')$ such that

$$\sum_{i'\in S_k^i}\bar{\mathbf{q}}^{i'}(A_{k+1}') = \bar{\mathbf{Q}}_j^i(A_{k+1}'), \quad \forall 1 \le i \le d_k.$$

Furthermore, there exist some i_1' and i_2' such that $\bar{\mathbf{q}}^{i_1'}(A_{k+1}') \sqsubset \mathbf{q}^{i_1'}(A_2')$ and $\bar{\mathbf{q}}^{i_2'}(A_{k+1}') \ne 0$.

Replacing the index i' with i, we define

$$\bar{\mathbf{Q}}^i(A_{k+2}') = \mathbf{G}(A_{k+1}')\bar{\mathbf{q}}^i(A_{k+1}'), \quad 1 \le i \le d_{k+1}.$$

Then $\bar{\mathbf{Q}}^i(A_{k+2}') \sqsubseteq \mathbf{Q}^i(A_{k+2}')$ for $1 \le i \le d_{k+1}$. As each $\bar{\mathbf{Q}}^i(A_{k+2}')$ is the weighted sum of the Graver basis of A_{k+1}', we know $A_{k+1}'\bar{\mathbf{Q}}^i(A_{k+2}') = 0$. Furthermore, there exist $1 \le i_1, i_2 \le d_{k+1}$ such that $\bar{\mathbf{Q}}^{i_1}(A_{k+2}') \sqsubset \mathbf{Q}^{i_1}(A_{k+2}')$ and $\bar{\mathbf{Q}}^{i_2}(A_{k+2}') \ne 0$.

Eventually, we can show that there exist $\bar{\mathbf{Q}}^i(A_\tau) = \mathbf{G}(A_{\tau-1}')\bar{\mathbf{q}}^i(A_{\tau-1}')$ for $1 \le i \le d_{\tau-1}$ such that $\bar{\mathbf{Q}}^i(A_\tau) \sqsubseteq \mathbf{Q}^i(A_\tau)$, $A_{\tau-1}'\bar{\mathbf{Q}}^i(A_\tau) = 0$. Furthermore, there exist $1 \le i_1, i_2 \le d_{\tau-1}$ such that $\bar{\mathbf{Q}}^{i_1}(A_\tau) \sqsubset \mathbf{Q}^{i_1}(A_\tau)$ and $\bar{\mathbf{Q}}^{i_2}(A_\tau) \ne 0$.

Given that $\mathbf{Q}^i(A_\tau) = \sum_{i'\in S_{\tau-1}^i}\mathbf{q}^{i'}(A_\tau)$, we can find $\bar{\mathbf{q}}^{i'}(A_\tau) \sqsubseteq \mathbf{q}^{i'}(A_\tau)$ such that $\bar{\mathbf{Q}}^i(A_\tau) = \sum_{i'\in S_{\tau-1}^i}\bar{\mathbf{q}}^{i'}(A_\tau)$, and moreover, there exist $1 \le i_1, i_2 \le n$ such that $\bar{\mathbf{q}}^{i_1}(A_\tau) \sqsubset \mathbf{q}^{i_1}(A_\tau)$ and $\bar{\mathbf{q}}^{i_2}(A_\tau) \ne 0$.

We define

$$\bar{\mathbf{g}}^i = \mathbf{G}(A_\tau)\bar{\mathbf{q}}^i(A_\tau), \quad \forall 1 \le i \le d_\tau = n$$

Note that by the positive sum property of the Graver basis, if $q_j^i(A_\tau) > 0$ then $\mathbf{g}^j(A_\tau)$ must lie in the same orthant as $\mathbf{g}^i$. Therefore $\bar{\mathbf{q}}^{i'}(A_\tau) \sqsubseteq \mathbf{q}^{i'}(A_\tau)$ implies that $\bar{\mathbf{g}}^i \sqsubseteq \mathbf{g}^i$. Further, $A_\tau\bar{\mathbf{g}}^i = 0$, and moreover, there exist $1 \le i_1, i_2 \le n$ such that $\bar{\mathbf{g}}^{i_1} \sqsubset \mathbf{g}^{i_1}$ and $\bar{\mathbf{g}}^{i_2} \ne 0$. Therefore, $0 \ne \bar{\mathbf{g}} = (\bar{\mathbf{g}}^1, \cdots, \bar{\mathbf{g}}^n) \sqsubset \mathbf{g}$.

Finally we show that $A\bar{\mathbf{g}} = 0$. This is equivalent as showing for every $1 \le k \le \tau - 1$,

$$\sum_{i_{\tau-k-1}\in S_k^\ell}\;\sum_{i_{\tau-k-2}\in S_{k+1}^{\tau-k-1}}\;\sum_{i_{\tau-k-3}\in S_{k+2}^{i_{\tau-k-2}}}\cdots\sum_{i_0\in S_{\tau-1}^{i_1}} A_k\bar{\mathbf{g}}^{i_0} = 0, \quad \forall 1 \le \ell \le d_k.$$

Using the equations $\bar{\mathbf{g}}^i = \mathbf{G}(A_\tau)\bar{\mathbf{q}}^i(A_\tau)$ and $\sum_{i'\in S_k^i}\bar{\mathbf{q}}^{i'}(A_{k+1}') = \bar{\mathbf{Q}}^i(A_{k+1}') = \mathbf{g}(A_k')\bar{\mathbf{q}}^i(A_k')$, we have the following equations (see the equations on the top.)

Therefore, the claim is proved. ☐

We now show that $\sum_{i=1}^n ||\mathbf{q}^i(A_\tau)||_1 = \sum_{i,j}|q_j^i(A_\tau)|$ is upper bounded by some value that only depends on $A_1, A_2, \cdots, A_\tau$. Using the fact that $\sum_{i'\in S_k^i}\mathbf{q}^{i'}(A_{k+1}') =$

$$\begin{aligned}
& \sum_{i_{\tau-k-1}\in S_k^\ell} \sum_{i_{\tau-k-2}\in S_{k+1}^{\tau-k-1}} \sum_{i_{\tau-k-3}\in S_{k+2}^{i_{\tau-k-2}}} \cdots \sum_{i_0\in S_{\tau-1}^{i_1}} A_k \bar{\mathbf{g}}^{i_0} \\
= & \sum_{i_{\tau-k-1}\in S_k^\ell} \sum_{i_{\tau-k-2}\in S_{k+1}^{\tau-k-1}} \sum_{i_{\tau-k-3}\in S_{k+2}^{i_{\tau-k-2}}} \cdots \sum_{i_0\in S_{\tau-1}^{i_1}} A_k \mathbf{G}(A_\tau)\bar{\mathbf{q}}^{i_0}(A_\tau) \\
= & \sum_{i_{\tau-k-1}\in S_k^\ell} \sum_{i_{\tau-k-2}\in S_{k+1}^{\tau-k-1}} \sum_{i_{\tau-k-3}\in S_{k+2}^{i_{\tau-k-2}}} \cdots \sum_{i_1\in S_{\tau-2}^{i_2}} A_k \mathbf{G}(A_\tau)\bar{\mathbf{Q}}^{i_1}(A_\tau) \\
= & \sum_{i_{\tau-k-1}\in S_k^\ell} \sum_{i_{\tau-k-2}\in S_{k+1}^{\tau-k-1}} \sum_{i_{\tau-k-3}\in S_{k+2}^{i_{\tau-k-2}}} \cdots \sum_{i_1\in S_{\tau-2}^{i_2}} A_k \mathbf{G}(A_\tau)\mathbf{G}(A'_{\tau-1})\bar{\mathbf{q}}^{i_1}(A_\tau) \\
= & \sum_{i_{\tau-k-1}\in S_k^\ell} \sum_{i_{\tau-k-2}\in S_{k+1}^{\tau-k-1}} \sum_{i_{\tau-k-3}\in S_{k+2}^{i_{\tau-k-2}}} \cdots \sum_{i_1\in S_{\tau-3}^{i_3}} A_k \mathbf{G}(A_\tau)\mathbf{G}(A'_{\tau-1})\bar{\mathbf{Q}}^{i_2}(A'_{\tau-1}) \\
= & \ldots \\
= & A_k \mathbf{G}(A_\tau)\mathbf{G}(A'_{\tau-1})\cdots\mathbf{G}(A'_{k+1})\bar{\mathbf{Q}}^\ell = A'_k\bar{\mathbf{Q}}^\ell = 0
\end{aligned}$$

$\mathbf{Q}^i(A'_{k+1}) = \mathbf{G}(A'_k)\mathbf{q}^i(A'_k)$,we have

$$\begin{aligned}
& \sum_{i=1}^{d_{k+1}} ||\mathbf{q}^i(A'_{k+1})||_1 \\
= & \sum_{i=1}^{d_k} ||\mathbf{Q}^i(A'_{k+1})||_1 \\
= & \sum_{i=1}^{d_k} ||\mathbf{G}(A'_k)\mathbf{q}^i(A'_k)||_1 \\
\le & \sum_{i=1}^{d_k} ||\mathbf{G}(A'_k)||_1 ||\mathbf{q}^i(A'_k)||_1 \\
= & \sum_{i=1}^{d_{k-1}} ||\mathbf{G}(A'_k)||_1 ||\mathbf{Q}^i(A'_k)||_1
\end{aligned}$$

Therefore,

$$\begin{aligned}
& \sum_{i=1}^{n} ||\mathbf{q}^i(A_\tau)||_1 \\
\le & \; ||\mathbf{G}(A'_{\tau-1})||_1 ||\mathbf{G}(A'_{\tau-2})||_1 \cdots ||\mathbf{G}(A'_2)||_1 ||\mathbf{Q}^i(A'_2)||_1.
\end{aligned}$$

Obviously each A'_k, and hence its Graver basis, and hence $||\mathbf{G}(A'_k)||$, is only dependent on $A_1, \cdots, A_\tau$. Furthermore, $\mathbf{Q}^i(A'_2) \in \mathcal{G}(A'_1)$, hence $||\mathbf{Q}^i(A'_2)||_1$, and consequently $\sum_{i=1}^n ||\mathbf{q}^i(A_\tau)||_1$, is only dependent on $A_1, \cdots, A_\tau$. Thus, for $\lambda = ||\mathbf{G}(A'_{\tau-1})||_1 ||\mathbf{G}(A'_{\tau-2})||_1 \cdots ||\mathbf{G}(A'_2)||_1 ||\mathbf{Q}^i(A'_2)||_1$ we have $\mathbf{g} \in H(A)$, and the lemma is proved. □

3.3 Dynamic programming in FPT time We provide a dynamic programming algorithm running in FPT algorithm for the tree-fold integer programming, and Theorem 1.3 follows. Towards this, we let $\lambda = \lambda(A_1, A_2, \cdots, A_\tau)$ and $H(A)$ be defined as in Lemma 3.2.

Given a feasible solution $\mathbf{x}$ of the integer programming (3.5), let $\gamma^* \in \mathbb{Z}_+$, $\mathbf{g}^* \in \mathcal{G}(A)$ satisfy that $\gamma^*\mathbf{g}^*$ is the best augmentation among Graver basis, i.e., the best possible augmentation vector of the form $\gamma\mathbf{g}$ where $\gamma \in \mathbb{Z}_+$ and $\mathbf{g} \in \mathcal{G}(A)$. The following lemma from [5] allows us to guess γ^* in $O(n)$ time:

LEMMA 3.3. ([5]) *In $O(n)$ time we can compute a set of integers Γ such that $\gamma^* \in \Gamma$ and $|\Gamma| \le n|H(A)|$.*

The proof in [5] is for the case when $\tau = 2$, however, it works directly for the general tree-fold matrices. For the completeness of the paper we give the proof as follows.

Proof. [of Lemma 3.3] Notice that if we fix $\mathbf{g} = \mathbf{g}^*$, then $\gamma = \gamma^*$ is the largest integer such that $\mathbf{l} \le \mathbf{x} + \gamma\mathbf{g}^* \le \mathbf{u}$ is still true. Therefore, if we consider each brick of the solution $\mathbf{x} = (\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^n)$, then there exists some $1 \le i \le n$ such that γ^* is the largest integer such that $\mathbf{l}^i \le \mathbf{x}^i + \gamma\mathbf{g}^{*i} \le \mathbf{u}^i$ is still true. As $\mathbf{g}^* \in H(A)$, $\mathbf{g}^{*i} \in H(A)$ for every i. Now for every $\mathbf{h} \in H(A)$ and every $1 \le i \le n$, we find out the largest integer $\gamma_{\mathbf{h},i}$ such that $\mathbf{l}^i \le \mathbf{x}^i + \gamma_{\mathbf{h},i}\mathbf{h}^i \le \mathbf{u}^i$ is true and add this integer to Γ. Obviously $\gamma^* \in \Gamma$ and $|\Gamma| \le n|H(A)|$. □

In the following we give a dynamic programming algorithm such that given a feasible solution $\mathbf{x}$ and any $\gamma \in \Gamma$, it finds out $\mathbf{h}_\gamma \in H(A)$ that minimizes $\mathbf{c}^T(\mathbf{x}+\gamma\mathbf{h}_\gamma)$, or equivalently, minimizes $\mathbf{c}^T\mathbf{h}_\gamma$ subject to the constraints that $\mathbf{l}^i \le \mathbf{x}^i + \gamma\mathbf{h}^i_\gamma \le \mathbf{u}^i$ and $A\mathbf{h}_\gamma = 0$. With such an algorithm, we can run it for every $\gamma \in \Gamma$

and pick γ' such that $\mathbf{c}^T(\mathbf{x}+\gamma'\mathbf{h}_{\gamma'})$ is minimal. By the definition of $H(A)$, $\gamma'\mathbf{h}_{\gamma'}$ is at least as good as the best augmentation via Graver basis and is thus the Graver-best augmentation that we desire.

The dynamic programming works in stages where in each stage it solves a subproblem. To define the subproblem, we define a matrix $\bar{A}$ as follows. Consider any small matrix A_i and all the rows in A that contain A_i. Suppose A_i appears consecutively in these rows from column $1 = d_0^i$ to column d_1^i, from column $d_1^i + 1$ to column $d_2^i, \cdots$, from column d_{k-1}^i to column $d_k^i = n$. We define $\bar{A}$ where each row of $\bar{A}$ is the summation of some rows in A. More precisely, $\bar{A}$ contains the same number of rows as A. If in the ℓ-th row of A some small matrix A_i appears consecutively from column d_j^i to column d_{j+1}^i, then in the ℓ-th row of $\bar{A}$ the small matrix A_i appears consecutively from 1 to d_{j+1}^i, that is, we construct $\bar{A}$ by extending the sequence of A_i in each row of A to column 1. It is obvious that $A\mathbf{h} = 0$ if and only if $\bar{A}\mathbf{h} = 0$.

Let $\bar{A}[1], \bar{A}[2], \cdots$ be all the rows in $\bar{A}$. Let ED_k be the set of rows $\bar{A}[\ell]$ where only the first k columns are non-zero. Obviously $ED_k \subseteq ED_{k+1}$. We define subproblem-k as follows:

Find some $\bar{\mathbf{h}}_\gamma$ such that

- $\bar{\mathbf{h}}_\gamma^i = 0$ for $i > k$, that is, only the first k bricks can be non-zero.
- $\bar{\mathbf{h}}_\gamma \in H(A)$.
- $\mathbf{l}^i \le \mathbf{x}^i + \gamma\bar{\mathbf{h}}_\gamma^i \le \mathbf{u}^i$ for $1 \le i \le k$.
- $\bar{A}[\ell] \cdot \bar{\mathbf{h}}_\gamma = 0$ for any $\bar{A}[\ell] \in ED_k$.
- $\mathbf{c}^T\bar{\mathbf{h}}_\gamma$ is minimized.

It is easy to see that the optimal solution for the subproblem-$(k+1)$ can be constructed by extending the optimal solution for the subproblem-k by one brick, and such a brick belongs to $H(A)$. Therefore, the optimal solution for subproblem-n can be found in $O(n|H(A)|)$ time.

The overall running time. We have shown in this subsection that the dynamic programming algorithm can find out a Graver best augmentation in $O(n^2)$ time (ignoring all the FPT-terms). By [6] the number of Graver best augmentations needed is $O(nL)$ where L is the encoding length of the integer programming, therefore tree-fold integer programming can be solved in $O(n^3L)$ time, and Theorem 1.3 is proved (if a feasible initial solution is given).

3.4 Constructing an initial feasible solution

We have proved the correctness of Theorem 1.3 if a feasible initial solution is given. In case a feasible solution is unknown, we construct an auxiliary tree-fold integer programming such that i). the initial feasible solution of the auxiliary programming is trivial; ii). the optimal solution of the auxiliary programming gives a feasible initial solution for the original tree-fold programming (1.1). The argument is essentially the same as that of [5].

We add auxiliary variables. For each $\mathbf{x}^i$, we add $2\sum_{k=1}^{\tau} s_k$ auxiliary variables and let them be $\mathbf{z}^i$. The new vector of variables becomes $(\mathbf{x}^1, \mathbf{z}^1, \mathbf{x}^2, \mathbf{z}^2, \cdots, \mathbf{x}^n, \mathbf{z}^n)$.

We introduce a lower bound of 0 and upper bound of $||\mathbf{b}||_\infty$ for each auxiliary variable. For each $1 \le k \le \tau$, we replace each A_k with $(A_k, 0_{s_k\times s_1}, 0_{s_k\times s_1}, 0_{s_k\times s_2}, 0_{s_k\times s_2}, 0_{s_k\times s_3}, \cdots, 0_{s_k\times s_{k-1}}, I_{s_k\times s_k}, -I_{s_k\times s_k}, 0_{s_k\times s_{k+1}}, 0_{s_k\times s_{k+1}}, \cdots, 0_{s_k\times s_\tau})$.

We change the objective function as the summation of all the auxiliary variables.

A feasible initial solution for the auxiliary ILP could be easily derived by setting $\mathbf{x} = 0$ and approperiate values to the auxiliary variables. Furthermore, the optimal solution of the auxiliary ILP is 0 if and only if there exists a feasible solution for (1.1). Therefore, we can apply our algorithm of the previous subsection to solve the auxiliary ILP and derive its optimal solution, which provides an initial feasible solution for the original tree-fold integer programming (1.1).

3.5 Subtree cover–integer programming formulation

The goal of this subsection is to derive an ILP formulation of the subtree cover problem which falls into the category of tree-fold integer programming. Given this result, applying Theorem 1.3, Theorem 1.2 is proved.

For ease of description, we let the root $r = v_1$. We define the unweighted distance between two vertices as the length the path connecting them in the same tree with all edge weights as 1. The *depth* of any vertex v_s is the unweighted distance of v_s to v_1.

Preprocessing. We consider the decision version of the problem which asks whether there exists a subtree cover of makespan B. We assume without loss of generality that the height of the tree, $h(T)$, is at most B, since otherwise we can conclude directly that there is no feasible solution of makespan at most B. For ease of presentation, we modify the problem in the following way. For any leaf whose depth is $h < h(T)$, we append a path to it which consists of $h(T) - h$ dummy vertices and $h(T) - h$ dummy edges of 0 weight. By doing so every leaf of T has a depth of $h(T)$. Next, we direct all the edges towards the root and move the weight of each edge to its source vertex. Specifically, the weight of the root is 0. Now the weight of any subtree is simply the total weight of its vertices. For simplicity, we still denote the modified tree as T and denote by n the number of its vertices.

Configurations. We define configurations. Any tree with at most $O(B^2)$ vertices can be encoded via an $O(B^2)$-vector as follows: We index all vertices from 1 to $O(B^2)$. For every vertex, we store its weight and its parent. We call such an $O(B^2)$-vector as a *configuration* and have the following simple observation.

Observation 2. *There are at most $\mu = B^{O(B^2)}$ different kinds of configurations.*

We index configurations arbitrarily as $CF_1, CF_2, \cdots, CF_\mu$ and denote by $|CF_j|$ the number of vertices in CF_j. Given an arbitrary configuration CF_j, we use (CF_j, k) to denote its vertex of index $k \in \{1, 2, \cdots, |CF_j|\}$. k is also called the *location* of this vertex. Let $\zeta = O(B^2)$ be the maximal number of vertices among all the configurations. A pair (CF_j, k) with $|CF_j| < k \leq \zeta$ is called invalid. For simplicity, 1 is always the index (location) of the root for every CF_j.

Given a configuration CF_j, we define a function f_j which maps a vertex of location k to the location of its parent (it shall be noted that here the function f_j has nothing to do with the function f in Theorem 1.3).

Now we revisit the subtree cover problem using the notion of configurations. Consider an arbitrary subtree of T rooted at $r = v_1$ whose weight is at most B. We first observe that there are at most $O(B^2)$ vertices in the subtree. To see why, we can first consider a subtree of weight at most B in the original tree before preprocessing. Since every vertex, except the root, has non-zero weight, the number of vertices is bounded by $B + 1$. As the preprocessing procedure will append at most $h(T) \leq B$ vertices below a vertex, the total number of vertices is thus bounded by $O(B^2)$. Hence, any subtree of weight at most B can be mapped to a configuration. Furthermore, any feasible solution can be interpreted as m subtrees that can be mapped to m configurations. Using this idea, we now establish an ILP formulation of the problem.

We define an integral variable $x_{i,(CF_j,k)}$ for every vertex v_i and every pair (CF_j, k). For $h \in \mathbb{Z}_+$, $x_{i,(CF_j,k)} = h$ implies that there are h subtrees in the solution which contain v_i, and furthermore, each of them can be mapped to the configuration CF_j such that v_i is mapped to the location k vertex in CF_j.

Obviously, v_i can not be mapped to an arbitrary vertex in CF_j. We say a vertex v_i is consistent with the pair (CF_j, k), if both of the following conditions are true:

- the depth of v_i in T is the same as the depth of the location k vertex in CF_j;
- the weight of v_i in T is the same as the weight of the location k vertex in CF_j.

Otherwise, we say they are inconsistent.

Let $CH(v_i)$ be the set of children of v_i, LF be the set of leaves. We establish the following $ILP(T)$ for the subtree cover problem (see the ILP on the top of the next page).

We explain the constraints. Constraint (II) ensures that every leaf is contained in one of the subtrees. Constraints (III) and (IV) are straightforward. We now explain constraint (I). Consider any feasible solution and let v_i be an arbitrary vertex. Let v_s be any child of v_i. If v_s is mapped to the vertex of location k in CF_j, then v_i must be mapped to the vertex of location $f_j(k)$ in CF_j. Therefore, if we consider the total number of configuration CF_j where a child of v_i is mapped to its vertex of location k, this should be equal to the number of configuration CF_j where v_i is mapped to its vertex of location $f_j(k)$. This is essentially what constraint (I) implies.

The following two lemmas ensures that the $ILP(T)$ we have derived indeed solves the subtree cover problem. One direction (Lemma 3.4) is staightforward, yet the other direction is a bit involved.

Lemma 3.4. *If there exists a feasible solution of the scheduling problem with makespan at most B, then there exists a feasible solution of the ILP with the objective value at most m.*

Lemma 3.5. *If there exists a feasible solution of the ILP with the objective value at most m, then there exists a feasible solution of the subtree cover problem with makespan at most B.*

Proof. Let LF be the set of leaves. In the following we show that it is possible to select a subset $LF' \subseteq LF$ such that there exists a subtree of weight at most B that contains each vertex of LF', and furthermore, if we delete LF' (together with the edge incident to them) from the tree T, there exists a feasible solution of the ILP for the remaining tree T' with the objective value at most $m-1$. If the above claim is true, we can iteratively carry on the argument to construct m subtrees that contain every vertex of LF and the lemma is proved.

We pick an arbitrary j_0 such that $x_{1,(CF_{j_0},1)} \geq 1$. Consider the children of the root v_1. According to constraint (I), for any location k such that $f_{j_0}(k) = 1$

$$
\begin{array}{lll}
ILP(T): & & \\
 & \min \sum_{j=1}^{\mu} x_{1,(CF_j,1)} & \\
(I) & \sum_{s:v_s\in CH(v_i)} x_{s,(CF_j,k)} = x_{i,(CF_j,f_j(k))}, & \forall 1\le i\le n, 1\le j\le \mu, 1\le k\le \zeta \\
(II) & \sum_{j=1}^{\mu}\sum_{k=1}^{\zeta} x_{i,(CF_j,k)} = 1, & \forall v_i \in LF \\
(III) & x_{i,(CF_j,k)} = 0, & \text{if } v_i \text{ and } (CF_j,k) \text{ are inconsistent, or } |CF_j| < k \le \zeta \\
(IV) & x_{i,(CF_j,k)} \in \mathbb{Z}_{\ge 0}, & 1\le i\le n, 1\le j\le \mu, 1\le k\le \zeta
\end{array}
$$

(i.e., the location of the vertices who are children of the root of CF_{j_0}), we have

$$\sum_{s:v_s\in CH(v_1)} x_{s,(CF_{j_0},k)} = x_{1,(CF_{j_0},1)} \ge 1.$$

Hence, for any k such that $f_{j_0}(k) = 1$, there exists at least one child of v_1, say, $v_{s(1,k)}$, such that $x_{s(1,k),(CF_{j_0},k)} \ge 1$. We pick an arbitrary one (if there are multiple) of such vertices for every k and let $H(1)$ be the set of these vertices.

Consider an arbitrary $v_{s(k_1)} \in H_1$ where $x_{s(k_1),(CF_j,k_1)} \ge 1$. According to constraint (I), for any k_2 such that $f_{j_0}(k_2) = k_1$, we have

$$\sum_{s:v_s\in CH(v_{s(k_1)})} x_{s,(CF_{j_0},k_2)} = x_{s(k_1),(CF_{j_0},k_1)} \ge 1.$$

Hence, for any k_2 such that $f_{j_0}(k_2) = k_1$, there exists at least one child of $v_{s(k_1)}$, say, $v_{s(k_2)}$ such that $x_{s(k_2),(CF_{j_0},k_2)} \ge 1$. We pick an arbitrary one of such vertices for every k_2 such that $f_{j_0}(k_2) = k_1$, and let $H(1,k_1)$ be the set of these vertices.

Suppose in general we have constructed the set of vertices $H(1,k_1,k_2,\cdots,k_i)$ such that

- for any $1 \le h \le i$, $f_{j_0}(k_h) = k_{h-1}$;
- for any k_{i+1} such that $f_{j_0}(k_{i+1}) = k_i$, there exists exactly one vertex $v_{s(k_{i+1})} \in H(1,k_1,k_2,\cdots,k_i)$ such that $x_{s(k_{i+1}),(CF_{j_0},k_{i+1})} \ge 1$.

If there exists at least one vertex of $H(1,k_1,k_2,\cdots,k_i)$ which is not a leaf, we proceed as follows. For any $v_{s(k_{i+1})} \in H(1,k_1,\cdots,k_i)$ which is not a leaf and any k_{i+2} such that $f(k_{i+2}) = k_{i+1}$, the following is true:

$$\sum_{s:v_s\in CH(v_{s(k_{i+1})})} x_{s,(CF_{j_0},k_{i+2})} = x_{s(k_{i+1}),(CF_{j_0},k_{i+1})} \ge 1.$$

Hence, there exists at least one child of $v_{s(k_{i+1})}$, say, $v_{s(k_{i+2})}$ such that $x_{s(k_{i+2}),(CF_{j_0},k_{i+2})} \ge 1$. We pick an arbitrary one of such vertices for every k_{i+2} and let $H(1,k_1,\cdots,k_{i+1})$ be the set of them. Otherwise every vertex of $H(1,k_1,k_2,\cdots,k_i)$ is a leaf and we stop.

Eventually we derive a sequence of sets $H(1,k_1,k_2,\cdots,k_i)$ and let H be the union of them.

Let $T[H]$ be the induced subgraph of T. Firstly, we claim that $T[H]$ is a subtree of the original tree T. To see why, it suffices to notice that every vertex of $H(1,k_1,k_2,\cdots,k_i)$ is connected to the root v_1.

Secondly, we claim that every leaf of the subtree $T[H]$ is also a leaf in T. This is straightforward. Let v_s be an arbitrary leaf of $T[H]$ which is not a leaf in the original graph, then according to our iterative construction, we will further consider the children of v_s and add some of them to H.

Thirdly, we claim that the weight of $T[H]$ is at most B. Indeed, the claim follows directly as every vertex of H is consistent to some vertex in CF_{j_0}.

Let $LF(H)$ be the set of leaves in $T[H]$. We delete $LF(H)$ and the edges incident to them in T and consider the ILP for the remaining subtree T'. It is easy to verify that the following solution $x'_{s,(CF_j,k)}$ is a feasible solution to $ILP(T')$ with the objective of at most $m-1$:

$$x'_{s,(CF_j,k)} = x_{s,(CF_j,k)}, \quad \text{if } j \ne j_0$$
$$x'_{s,(CF_{j_0},k)} = x_{s,(CF_{j_0},k)} - 1, \quad \text{if } v_s \in H \setminus LF(H)$$

Therefore given a feasible integer solution with the objective value at most m, we can iteratively construct at most m subtrees such that every vertex is covered, and the lemma is proved. □

Still, $ILP(T)$ is similar but not exactly the same as a tree-fold integer programming. We need to tune the ILP a bit. The tuning is essentially by replacing some of the variables with the equation in (I) it satisfies, i.e., we will remove some of the variables (See Section 3.6). Once transformed into a tree-fold integer programming, Theorem 1.3 can be applied and Theorem 1.2 is proved.

3.6 Tuning the ILP We alter the ILP a bit so that it becomes a tree-fold integer programming.

Given CF_j, we let $F_j^{-1}(k) = \{w|f_j(w) = k\}$. For $h \geq 2$, we define $F_j^{-h}(k) = \{w|f_j(w) \in F_j^{-h+1}(k)\}$. Recall that f_j is the function that maps the location of a vertex to the location of its parent in CF_j, therefore $F_j^{-h}(k)$ the set of locations of vertices satisfying the following: i). they are descendants of the location k vertex; ii). for each of them, the unweighted distance to the location k vertex is h.

We show that, it is possible to remove all the variables $x_{i,(CF_j,k)}$ where v_i is not a leaf and establish an equivalent ILP.

Let $LF(v_i)$ be the set of all leaves of the subtree rooted at v_i. By constraint (I), we have the following

$$x_{i,(CF_j,k)} = \sum_{s:v_s \in CH(v_i)} x_{s,(CF_j,w)}, \quad \forall w \in F_j^{-1}(k).$$

If $w \in F_j^{-1}(k)$ is not a leaf, we could further express $x_{s,(CF_j,w)}$ into the summation of other variables. In general, consider any vertex v_i whose depth is $h(T) - h$. As the depth of every leaf is $h(T)$, the unweighted distance of any leaf in $LF(v_i)$ to v_i is h, and we have the following:

$$x_{i,(CF_j,k)} = \sum_{s:v_s \in LF(v_i)} x_{s,(CF_j,w)}, \quad \forall w \in F_j^{-h}(k).$$

Specifically,

$$x_{1,(CF_j,1)} = \sum_{s:v_s \in LF} x_{s,(CF_j,w)}, \quad \forall w \in F_j^{-h(T)}(1).$$

Now every $x_{1,(CF_j,1)}$ could be expressed using $x_{s,(CF_j,w)}$ where v_s is a leaf. We replace the objective function using the above equations.

Let $L_h(CF_j)$ be the subset of locations of CF_j whose depth is $h(T) - h$, and let $L_h^{\geq 2}(CF_j) = \{k||F_j^{-h}(k)| \geq 2\}$, we replace constraint (I) by the following:

$$\sum_{s:v_s \in LF(v_i)} x_{s,(CF_j,w)} - \sum_{s:v_s \in LF(v_i)} x_{s,(CF_j,w')} = 0,$$
$$\forall v_i \in V_h, k \in L_h^{\geq 2}(CF_j), w, w' \in F_j^{-h}(k), \quad (I')$$

where V_h is the set of vertices of depth $h(T) - h$.

It is obvious that the new ILP is equivalent as the original ILP since we simply replace each $x_{s,(CF_j,w)}$ where v_s is not a leaf with the equality it satisfies.

In the following we show that the modified ILP belongs to the tree-fold integer programming. It suffices to consider constraints (I') and (II). Let $\mathbf{x}^i = (x_{i,(CF_1,1)}, x_{i,(CF_1,2)}, \cdots, x_{i,(CF_1,\zeta)}, x_{i,(CF_2,1)}, \cdots, x_{i,(CF_2,\zeta)}, \cdots, x_{i,(CF_\mu,\zeta)})^T$ and $\mathbf{x} = (\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^{|LF|})^T$.

Consider constraint (II):

$$\sum_{j=1}^{\mu} \sum_{k=1}^{\zeta} x_{i,(CF_j,k)} = 1, \quad \forall v_i \in LF$$

Let $\tau = |h(T)| + 1$. We define $A_1 = I_{\mu\zeta \times \mu\zeta}$, constraint (II) could be written as $\sum_i A_1 \mathbf{x}^i = (1, 1, \cdots, 1)_{1 \times \mu\zeta}$.

Consider constraint (I'). For any vertex $v_s \in LF(v_i)$ where $v_i \in V_h$, the constraint (I') could be rewritten as $\sum_{s:v_s \in LF(v_i)} A_{\tau-h}\mathbf{x}^s = 0$ where $A_{\tau-h}$ consists of $\sum_j \sum_{k \in L_h^{\geq 2}(CF_j)} (|F_j^{-h}(k)| - 1) \cdot |F_j^h(k)|/2$ different rows, and each row consists of $0, 1, -1$ such that the entry that becomes the coefficient of $x_{s,(CF_j,w)}$ after multiplication is 1, the entry that becomes the coefficient of $x_{s,(CF_j,w')}$ after multiplication is -1, and other entries are 0. Given the fact that $LF(v_i) = \cup_{s:v_s \in CH(v_i)} LF(v_s)$, it is not difficult to verify that contraints (I') and (II) could be written as $A\mathbf{x} = b$ where A is a tree-fold matrix consisting of submatrices A_1, A_2, $\cdots$, A_τ.

Now applying Theorem 1.3, an $f(B)n^4$ time algorithm for the subtree cover problem is derived for some function f, and Theorem 1.2 is proved.

4 Conclusion

We consider the subtree cover problem in this paper and provide a PTAS as well as an FPT algorithm parameterized by the makespan. Our FPT algorithm follows from a more general FPT result on the tree-fold integer programming, which extends the existing FPT algorithm on the n-fold integer programming. The running times of the PTAS and FPT algorithms are huge and are only of theoretical interest. It is an intriguing open problem whether there exists an efficient polynomial approximation scheme (EPTAS) for the subtree cover problem. Another important open problem is whether we can derive FPT algorithm for integer programming with the matrix A that has an even more general structure.

References

[1] Lin Chen, Klaus Jansen, and Guochuan Zhang. On the optimality of approximation schemes for the classical scheduling problem. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Al-*

gorithms, pages 657–668. Society for Industrial and Applied Mathematics, 2014.

[2] Jesús A De Loera, Raymond Hemmecke, and Matthias Köppe. *Algebraic and geometric ideas in the theory of discrete optimization*, volume 14. SIAM, 2013.

[3] Guy Even, Naveen Garg, Jochen Könemann, R Ravi, and Amitabh Sinha. Min–max tree covers of graphs. *Operations Research Letters*, 32(4):309–315, 2004.

[4] Jack E Graver. On the foundations of linear and integer linear programming i. *Mathematical Programming*, 9(1):207–226, 1975.

[5] Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. N-fold integer programming in cubic time. *Mathematical Programming*, 137(1-2):325–341, 2013.

[6] Raymond Hemmecke, Shmuel Onn, and Robert Weismantel. A polynomial oracle-time algorithm for convex integer minimization. *Mathematical Programming*, 126(1):97–117, 2011.

[7] Dorit S Hochbaum and David B Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)*, 34(1):144–162, 1987.

[8] Serkan Hoşten and Seth Sullivant. A finiteness theorem for markov bases of hierarchical models. *Journal of Combinatorial Theory, Series A*, 114(2):311–321, 2007.

[9] Bart MP Jansen and Stefan Kratsch. A structural approach to kernels for ilps: Treewidth and total unimodularity. In *Algorithms-ESA 2015*, pages 779–791. Springer, 2015.

[10] Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the gap for makespan scheduling via sparsification techniques. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 72:1–72:13, 2016.

[11] Klaus Jansen and Monaldo Mastrolilli. Scheduling unrelated parallel machines: linear programming strikes back. *University of Kiel, Technical Report 1004*, 2010.

[12] M Reza Khani and Mohammad R Salavatipour. Improved approximation algorithms for the min-max tree cover and bounded tree cover problems. *Algorithmica*, 69(2):443–460, 2014.

[13] Dušan Knop and Martin Kouteckỳ. Scheduling meets n-fold integer programming. *arXiv preprint arXiv:1603.02611*, 2016.

[14] Dušan Knop, Martin Kouteckỳ, and Matthias Mnich. Combinatorial n-fold integer programming and applications. *arXiv preprint arXiv:1705.08657*, 2017.

[15] Stefan Kratsch. On polynomial kernels for sparse integer linear programs. *Journal of Computer and System Sciences*, 82(5):758–766, 2016.

[16] Jan Karel Lenstra, David B Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1-3):259–271, 1990.

[17] Matthias Mnich and Andreas Wiese. Scheduling and fixed-parameter tractability. *Mathematical Programming*, 154(1-2):533–562, 2015.

[18] Hiroshi Nagamochi and Kohei Okada. Approximating the minmax rooted-tree cover in a tree. *Information Processing Letters*, 104(5):173–178, 2007.

[19] Shmuel Onn. Nonlinear discrete optimization. *Zurich Lectures in Advanced Mathematics, European Mathematical Society*, 2010.

[20] Sartaj K Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM (JACM)*, 23(1):116–127, 1976.

[21] Liang Xu, Zhou Xu, and Dongsheng Xu. Exact and approximation algorithms for the min–max k-traveling salesmen problem on a tree. *European Journal of Operational Research*, 227(2):284–292, 2013.