CrossMark

Submodular unsplittable flow on trees

Anna Adamaszek[1] · **Parinya Chalermsook[2,4]** ·
Alina Ene[3] · **Andreas Wiese[4]**

Abstract We study the Unsplittable Flow problem (UFP) on trees with a submodular objective function. The input to this problem is a tree with edge capacities and a collection of tasks, each characterized by a source node, a sink node, and a demand. A subset of the tasks is feasible if the tasks can simultaneously send their demands from the source to the sink without violating the edge capacities. The goal is to select a feasible subset of the tasks that maximizes a submodular objective function. Our main result is an $O(k \log n)$-approximation algorithm for Submodular UFP on trees where k denotes the pathwidth of the given tree. Since every tree has pathwidth $O(\log n)$, we obtain an $O(\log^2 n)$ approximation for arbitrary trees. This is the first non-trivial approximation guarantee for the problem, matching the best known approximation

✉ Parinya Chalermsook
 parinya.chalermsook@aalto.fi

 Anna Adamaszek
 anad@di.ku.dk

 Alina Ene
 a.ene@dcs.warwick.ac.uk

 Andreas Wiese
 awiese@mpi-inf.mpg.de

[1] University of Copenhagen, Copenhagen, Denmark

[2] Aalto University, Espoo, Finland

[3] University of Warwick, Coventry, UK

[4] Max-Planck-Institut für Informatik, Saarbrücken, Germany

🍃 Springer

ratio for UFP on trees with a linear objective function. Our main technical contribution is a new geometric relaxation for UFP on trees that builds on the recent work of Bonsma et al. (2014, FOCS), Anagnostopoulos et al. (Amazing $2+\epsilon$ approximation for unsplittable flow on a path, SIAM, pp 26–41, 2014) for UFP on paths with a linear objective. Our relaxation is very structured, so it can be combined with the contention resolution framework of Chekuri et al. (2009, STOC). Our approach is robust and extends to several related problems, such as UFP with bag constraints and the Storage Allocation Problem. Additionally, we study the special case of UFP on trees with a linear objective and upward instances where, for each task, the source node is a descendant of the sink node. Such instances generalize UFP on paths. We build on the work of Bansal et al. (A quasi-PTAS for unsplittable flow on line graphs, ACM, pp 721–729, 2006) for UFP on paths and obtain a QPTAS for upward instances when the input data is quasi-polynomially bounded. We complement this result by showing that, unlike the path setting, upward instances are APX-hard if the input data is arbitrary.

Keywords Unsplittable flow problem · Submodular functions · Approximation algorithms

Mathematics Subject Classification 68W25 · 68Q25 · 90C05

1 Introduction

Submodular functions are a rich class of functions with many applications both in theory and in practice. On the theoretical side, submodularity is a key concept in combinatorial optimization and economics with deep mathematical consequences. On the practical side, submodular functions arise naturally in a variety of settings such as data summarization, sensor placement, inference in graphical models, image segmentation, social networks, auctions, and exemplar clustering [8,17,20,21,23–26].

One of the main reasons for the success of submodularity is that it combines a significant modeling power with a certain degree of tractability. This delicate balance between generality and tractability has made submodular functions very appealing, and there has been a significant interest in optimizing submodular functions subject to a variety of constraints.

The traditional approach to submodular maximization makes extensive use of the classical Greedy algorithm of Nemhauser, Wolsey, and Fisher [27]. The Greedy algorithm and its continuous counterparts are well-suited for constraints such as cardinality, matroids, and knapsack, but they fail to handle other types of natural constraints. Thus there is an increasing need to develop algorithms for general constraints.

A major contribution in this direction comes from the work of Chekuri et al. [15] which has developed a powerful framework for submodular function maximization with general constraints. Their framework leverages the power of mathematical programming relaxations coupled with structured rounding schemes called *contention resolution (CR) schemes*. In particular, it unifies several previous results for special cases (e.g., matroids or knapsack constraints) and thus captures the types of con-

straints for which we know how to optimize submodular functions. This has led to the following very interesting meta-question:

For which type of constraints can we provide structured relaxations that admit good CR schemes?

In this paper, we address this question in the specific case of the unsplittable flow problem (UFP). In this setting, we are given an edge-capacitated, undirected graph and a collection of tasks; each task is specified by a source vertex, a sink vertex, and a demand. The goal is to select a maximum profit subset of the tasks that can be routed unsplittably, i.e., the task's demand is routed along a single path from the source to the sink subject to the edge capacities.

The problem is well-studied, and most of the results focus on linear objectives. Despite its apparent simplicity, already UFP on paths captures several well-studied problems, including the knapsack problem (when the graph is a single edge) and resource allocation problems.

UFP is quite challenging even on paths and trees, and one of the main reasons for the difficulty is the lack of LP relaxations with small integrality gaps. The natural LP relaxation for the problem has an $\Omega(n)$ integrality gap even on paths [11], and standard approaches for strengthening the LP by adding valid inequalities fail to improve the integrality gap significantly [14]. Chekuri et al. [14] gave a novel LP relaxation for UFP on paths that strengthens the standard LP using clique type of constraints, and they showed that it has an $O(\log n)$ integrality gap. The relaxation of [14] can also be extended to trees, and understanding this relaxation has been an interesting and challenging open question.[1]

The design of good relaxations for UFP is motivated not only by the goal of obtaining better approximations for linear objectives, but also by the need of handling more general constraints and objective functions. In particular, the current approaches for submodular objectives rely on structured relaxations with good CR schemes. As a result, there is a discrepancy between the approximation guarantees for linear and submodular objectives. There has been a long line of work for UFP on paths with a linear objective that led to a constant factor approximation [5,7]; these approaches combine the standard LP relaxation with dynamic programming techniques. Chekuri et al. [14] give a combinatorial greedy algorithm for UFP on trees with a linear objective that achieves an $O(\log^2 n)$ approximation. Chakaravarthy et al. [10] study the generalization of the UFP with a linear objective, called bagUFP, where tasks are partitioned into bags and a feasible solution is allowed to select at most one task per bag. They give $O(\log n)$ approximation algorithm for bagUFP on paths based on a primal-dual approach. Grandoni et al. [19] improve the approximation factor to $O(\log n/\log \log n)$, and to $O(1)$ in the uniform-weight setting. Note that bagUFP is a special case of UFP with a submodular objective. In general, for UFP with a submodular objective, an $O(\log n)$ approximation for paths can be easily obtained by combining the results of [14] and [15]. In contrast, *no non-trivial approximation was known for trees prior to our work*. Chekuri et al. [15] consider instances of submodular

---

[1] Friggstad and Zao [18] showed an $O(\log^2 n)$ upper bound on the integrality gap of the relaxation of [14] for UFP on trees with a linear objective. This upper bound is shown via a primal-dual analysis which is not suitable for designing a CR scheme.

UFP on trees that satisfy a certain assumption, called the no-bottleneck assumption (NBA),[2] and they give a constant factor approximation for such instances. However, the no-bottleneck assumption is very restrictive and removing this restriction poses several technical challenges, particularly for the design of mathematical programming relaxations.

Thus, there has been an extensive work on UFP on paths but relatively fewer results on trees. Since UFP models the allocation of communication bandwidths in networks, we believe that it is worthwhile to develop a better understanding for more complex network topologies, such as trees. Also, submodular objective functions are much richer than linear objectives, and can model for instance linear objective functions with additional constraints.

Our contributions. We give the first approximation algorithm for submodular UFP on trees and the first relaxation with a matching integrality gap. Our algorithm achieves an approximation ratio of $O(k \log n)$ on trees with pathwidth k. As each tree has pathwidth $O(\log n)$, this gives an $O(\log^2 n)$-approximation for arbitrary trees, matching the best known result for linear objective functions [14]. For several special cases of the problem, such as paths, spiders, and caterpillars, our approximation ratio improves to $O(\log n)$ (since in those cases $k = O(1)$), and such a ratio was not even known for linear objectives. Thus our result generalizes and improves the best approximations known for UFP on paths with a submodular objective and UFP on trees with a linear objective.

Theorem 1 *There is a $O(k \cdot \log n)$ approximation for Submodular UFP on trees, where k is the pathwidth of the tree and n is the number of nodes in the tree. Additionally, there is a polynomial-sized relaxation for the problem with a matching integrality gap.*

We obtain our result via a new geometric LP relaxation for UFP on trees that is very different from the clique-based approach of [14]. Our relaxation builds on a powerful two-dimensional geometric viewpoint developed in the context of the UFP problem on paths with a linear objective [7]. This viewpoint connects UFP to structured instances of the Maximum Independent Set of Rectangles (MISR) problem [1,12,13], which in turn allows one to handle instances of UFP on paths for which the standard LP relaxation fails. The geometry was exploited to obtain a combinatorial algorithm for such instances that is based on dynamic programming. A related two-dimensional visualization was used in [2], again as the basis of a dynamic program. These approaches, however, break down for submodular UFP on trees; in the two-dimensional viewpoints, an input path corresponds to a subinterval of the x-axis and this is no longer meaningful for trees. Also, dynamic programming approaches are not suitable for submodular objective functions. In contrast to previous work, the focus in this paper is to translate these geometric insights to an LP relaxation for UFP on trees. We give a CR scheme for our relaxation that can be combined with the framework of [15] to obtain approximation guarantees for submodular objectives. The core of our reasoning is that our LP-formulation not only decides which tasks to select, but also computes a drawing of them as non-overlapping rectangles on suitable subpaths

---

[2] The no-bottleneck assumption states that the maximum demand of any task is at most the minimum capacity of any edge.

of the tree. Our LP is a polynomial-sized extended formulation and, to our knowledge, this is the first time that an extended formulation is used in the context of CR schemes.

A very important feature of the CR scheme framework is that it allows one to combine several constraints, thus extending the applicability of our approach to two generalized settings. First, in the Submodular Bag-UFP on trees problem, the input tasks are partitioned into bags and a feasible solution is allowed to select at most one task per bag [10].[3] We obtain an $O(k \log n)$ approximation for Submodular Bag-UFP on trees of pathwidth k. Second, we obtain an $O(\log n)$ approximation for the Submodular Storage Allocation Problem on trees. This problem has the same input as UFP, with additional requirements that each selected task gets a private subinterval of width equal to the demand, contained in $[0, u_e]$ for each edge e used by the task. We require these subintervals to be disjoint for any two tasks sharing an edge of the tree, intuitively enforcing each task to get a contiguous portion of the resource spectrum.

Finally, we round up our contributions with the following results for a special case of UFP on trees with a linear objective function. An instance of UFP on tree is an *upward instance* if the input tree is rooted and, for every task, the source node of the task is an ancestor of the sink node (or vice-versa).

Theorem 2 *There is a* $(1 + \epsilon)$ *approximation algorithm for upward instances of* UFP *on trees with running time* $n^{\text{poly}(\log(n/\epsilon)) \log(d_{\max}/d_{\min})}$. *In particular, if the demands are quasi-polynomially bounded, this yields a* QPTAS.

Unlike for UFP on paths [5], we show that the dependency of the running time on the term $\log(d_{\max}/d_{\min})$ can *not* be removed for upward instances of UFP on trees. In fact, assuming the *Exponential Time Hypothesis (ETH)*, the running time of our approximation scheme is essentially tight. This illustrates an inherent distinction between paths and upward instances on trees. Also, this establishes the problem's status as a very rare problem that allows a QPTAS on quasi-polynomially bounded input but is APX-hard on general instances.

Theorem 3 *There is a constant* ϵ_0 *such that for all* $\delta > 0$ *any* $(1 + \epsilon_0)$-*approximation for upward instances of* UFP *on trees runs in time of at least* $n^{\text{poly}(\log n) \log^{1-\delta}(d_{\max}/d_{\min})}$, *unless ETH fails. Also, the problem is* APX-*hard.*

Other related work. The problem of maximizing submodular functions subject to various constraints is very well-studied; we refer the reader to [15] for an overview. UFP with a linear objective is also extensively studied. The best approximation is a $(2 + \varepsilon)$ approximation [2] and a QPTAS [4,6] for UFP on paths, and an $O(\log^2 n)$ approximation for UFP on trees [14].

---

[3] For linear objective functions, the bag constraints can be "glued" with the objective function, yielding an instance of Submodular UFP. It is not clear though whether this holds in general for any initial submodular objective function.

2 Preliminaries

Formal problem definitions. The input of Unsplittable Flow problem on trees (UFP-tree) consists of an undirected tree $T = (V, E)$ with edge capacities $u_e \in \mathbb{Z}_+$, and a set of tasks $\mathcal{T}$. Each task $i \in \mathcal{T}$ is characterized by a start vertex $s_i \in V$, an end vertex $t_i \in V$, a demand $d_i \in \mathbb{Z}_+$, and a profit $w_i \in \mathbb{Z}_+$. For each task $i \in \mathcal{T}$ denote by p_i the unique path between s_i and t_i in T. A subset of tasks $\mathcal{T}' \subseteq \mathcal{T}$ is said to be *feasible* if $\sum_{i \in \mathcal{T}': \, p_i \ni e} d_i \le u_e$ for each edge $e \in E$. The goal is to find a feasible solution maximizing $w(\mathcal{T}') := \sum_{i \in \mathcal{T}'} w_i$.

The Submodular UFP-tree problem is a generalization of UFP-tree, where instead of a linear weight function w we are given a submodular objective function $f : 2^{\mathcal{T}} \to \mathbb{R}_+$ and the goal is to select a feasible subset $\mathcal{T}' \subseteq \mathcal{T}$ maximizing $f(\mathcal{T}')$. A function $f : 2^{\mathcal{T}} \to \mathbb{R}_+$ is *submodular* if $f(A) + f(B) \ge f(A \cap B) + f(A \cup B)$ for any two subsets $A, B \subseteq \mathcal{T}$. We assume that f is given as a value oracle: In a unit of computational time, the algorithm may specify $S \subseteq \mathcal{T}$ and obtain the value $f(S)$.

Path Decomposition and pathwidth. A *path decomposition* of a graph G is given by a collection $\mathcal{X} = \{X_1, X_2, \ldots, X_m\}$ satisfying

- $X_i \subseteq V(G)$ for each $i = 1, \ldots, m$. We refer to each set X_i as a *bag*.
- For each edge $uv \in E(G)$, there is a bag X_i such that $u, v \in X_i$.
- For every three indices $i < j < k$, we have $X_i \cap X_k \subseteq X_j$.

The *pathwidth* of a graph G is a minimum value k for which there exists a path decomposition $\mathcal{X}$ of G where for each bag $X_i \in \mathcal{X}$ we have $|X_i| \le k + 1$. Any tree has pathwidth of at most $O(\log n)$ [22].

3 Geometric relaxation for submodular UFP on trees

In this section, we present our $O(k \cdot \log n)$ approximation algorithm for Submodular UFP-tree. We first describe a pseudo-polynomial sized LP relaxation for UFP-tree with a linear objective function. In Sect. 3.2, we show how to reduce the size of the LP to polynomial. In Sect. 3.3, we extend our algorithm to a submodular objective function.

3.1 A pseudo-polynomial sized relaxation

In the following, we give a geometric LP-relaxation for UFP-tree with a linear objective function. The relaxation has pseudo-polynomial size.

Reduction to intersecting instances. We reduce the general case to the case in which the path of each task contains the root of the tree. We call such instances *intersecting instances*. Chekuri et al. [14] showed that, via a standard centroid decomposition, we can reduce an arbitrary instance to a collection of intersecting instances at a loss of $O(\log n)$ in the approximation ratio.

Lemma 1 (Chekuri et al. [14]) *Suppose that there is a polynomial time α-approximation for UFP-tree on intersecting instances. Then there is a polynomial*

time $O(\alpha \cdot p \log n)$ approximation algorithm for the problem on arbitrary trees. More-over, this holds for the generalization of the problem in which the objective function is sub-additive.[4]

3.1.1 Partitioning into paths

In the remainder of this section, we assume that we are given an intersecting instance on a tree T of pathwidth k. Our goal is to compute a $O(k)$-approximation for such instances, so that using Lemma 1 we obtain a $O(k \log n)$-approximation for the general problem. First, we split a given tree into a collection $\mathcal{P}$ of paths such that each input task shares an edge with at most $O(k)$ paths in $\mathcal{P}$. Then, we define a new LP relaxation for the problem with a randomized rounding with alteration strategy. The relaxation will be based on a two-dimensional geometric viewpoint for each path in $\mathcal{P}$.

For our path partition $\mathcal{P}$ we require that each path $P \in \mathcal{P}$ is an *upward path*, i.e., one endpoint of the path is an ancestor in T of the other endpoint. The following observation follows from the property of an intersecting instance.

Observation 1 *For each task i and each upward path P, if i uses an edge of P then it uses the top edge of P.*

Definition 1 Consider an intersecting instance I of UFP-tree on a rooted tree T. Let $\mathcal{P} = \{P_1, \ldots, P_\ell\}$ be a collection of paths in T. We say that $\mathcal{P}$ is a *K-nice splitting* for I if it has the following properties.

- The paths in $\mathcal{P}$ are edge-disjoint, upward, and they partition the edges of the tree T.
- Each task of I uses an edge of at most K paths in $\mathcal{P}$.

The next lemma shows the existence of a $O(k)$-nice splitting.

Lemma 2 *Consider an intersecting instance I of UFP-tree on a rooted tree T of path-width k. There is a polynomial time algorithm that constructs an $O(k)$-nice splitting for I.*

Proof Consider an intersecting instance I of UFP-tree on a rooted tree T of pathwidth k. Our algorithm will assign *colors* to the edges of T. For each color j, edges colored with j will form a path P_j in the $O(k)$-nice splitting of T.

The algorithm is recursive and is applied to a subtree of the form T_{uv}, where uv is an edge of T such that u is a parent of v. Then T_{uv} is a tree $T_v \cup \{uv\}$, where T_v is a subtree of T rooted at v. The algorithm will color the edges of the subtree, and return an integer k_{uv} which is the maximum number of colors in any root-to-leaf path of T_{uv}. Now we describe the algorithm. If T_{uv} contains only a single edge uv, then uv is given a color and k_{uv} is set to 1. Otherwise uv has some children edges $vv_1, vv_2, \ldots, vv_q$ (i.e., the children of v are $v_1, \ldots, v_q$), and we recursively color the subtrees T_{vv_j} for all $j = 1, \ldots, q$. Assume that each subtree uses a distinct set of colors. Then we give

---

the edge uv the same color as an edge vv_j such that j maximizes k_{vv_j}. We set k_{uv} to $\max_j k_{vv_j}$. The following claim is immediate.

claim 1 When the algorithm finishes coloring a subtree T_{uv}, edges of the same color form an upward path. Moreover, any root-to-leaf path in T_{uv} contains edges from at most k_{uv} color classes.

Now we show that the number of colors used is proportional to k.

In order to achieve this, we show that if we run the above algorithm on a subtree T_{uv} and obtain $k_{uv} \geq 3k'$ for some positive integer k', then the pathwidth of the subtree T_v (i.e., excluding the vertex u) is at least k'.

We prove this by induction on k'. Since the pathwidth is at least 1, the base case (for $k' = 1$) holds.

Now, let $\mathcal{P}$ be a collection of paths output by the algorithm, and Q a root-to-leaf path that intersects the maximum number of paths in $\mathcal{P}$, where this number is at least $3k'$. We traverse the path $Q = e_1 e_2 \ldots e_q$ from root to leaf. Then we must have $k_{e_1} \geq 3k'$ and $k_{e_1} \geq k_{e_2} \geq \ldots \geq k_{e_q} = 1$. Due to the way we constructed our paths, since Q intersects the maximum number of paths in $\mathcal{P}$ we have that $k_{e_{j-1}} \leq k_{e_j} + 1$ for all j. Let e_a be the edge on Q closest to the root for which $k_{e_a} = 3k' - 1$. Our algorithm guarantees that its parent edge e_{a-1} must have another child edge e'_a such that $k_{e'_a} \geq k_{e_a} = 3k' - 1$ (e'_a is the edge that obtains the same color as e_{a-1}). Using the same argument, let e_b and e_c be the top-most edges on Q such that $k_{e_b} = 3k' - 2$ and $k_{e_c} = 3k' - 3$, respectively. Again, the algorithm guarantees that there are edges e'_b and e'_c with one endpoint at Q such that $k_{e'_b} \geq 3k' - 2$ and $k_{e'_c} \geq 3k' - 3$. The three subtrees $T_{e'_a}$, $T_{e'_b}$ and $T_{e'_c}$ are disjoint, and they are "attached" to the path Q via edges e'_a, e'_b, e'_c respectively. Denote by T_a, T_b, T_c the subtrees $T_{e'_a}$, $T_{e'_b}$ and $T_{e'_c}$ excluding the edges e'_a, e'_b, e'_c, respectively.

By the induction hypothesis, each of T_a, T_b, T_c has pathwidth at least $k' - 1$. Assume for contradiction that T_v has pathwidth smaller than k', i.e., it has a path splitting $\mathcal{X} = \{X_1, \ldots, X_m\}$ with $|X_j| \leq k'$ for all j.

claim 2 There are $X_{a'}, X_{b'}, X_{c'} \in \mathcal{X}$ s.t. $X_{a'} \subseteq T_a$, $X_{b'} \subseteq T_b$ and $X_{c'} \subseteq T_c$.

Proof Assume that this does not hold for T_a (the arguments are similar for T_b and T_c.) The decomposition $\mathcal{X}$ restricted to nodes in $V(T_a)$ gives a path decomposition for T_a of pathwidth less than $k' - 1$, a contradiction. ∎

W.l.o.g. we can assume that $a' < b' < c'$. Therefore, the set of vertices in the bag $X_{b'}$ forms a vertex cut in T_v separating the two components $X_{a'}$ and $X_{c'}$. This is impossible, because T_a and T_c are connected in $T_v \backslash V(T_b)$ via the path Q. We obtain a contradiction. □

3.1.2 Geometric viewpoint

Let $\mathcal{P} = \{P_1, \ldots, P_\ell\}$ be an $O(k)$-nice splitting of the instance that is guaranteed by Lemma 2. We use $\mathcal{P}$ to write an LP relaxation for the problem, based on the following geometric viewpoint. For a path $P \in \mathcal{P}$, let $\mathcal{T}_P$ be the set of tasks from $\mathcal{T}$ using an edge of P.

Fig. 1 A one-sided staircase instance of UFP-path, and a drawing of a feasible set of tasks using rectangles
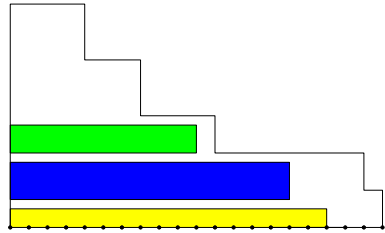
If we restrict the tasks in our instance to a path P in T, we get an instance of the Unsplittable Flow problem on paths (UFP-path) in a natural way. For each task $i \in \mathcal{T}_P$, the UFP-path instance has a corresponding task whose path is $p_i \cap P$. Notice that each task $i \in \mathcal{T}_P$ uses the top edge of P, so we can assume w.l.o.g. that when traversing the edges of P from top to bottom, their capacities are non-increasing. We call this a *one-sided staircase* instance.

We claim that for such UFP-path instance on a path P, each feasible subset of the tasks can be represented as a collection of non-overlapping rectangles drawn underneath the capacity profile, such that each task i has a corresponding rectangle of height d_i whose projection on P is the path of i. We interpret these rectangles as open sets and call such a drawing a *representing drawing* (see Fig. 1).

Lemma 3 *Consider an instance of UFP-path on a path P in which all of the tasks use the first edge of P. Any feasible subset of the tasks admits a representing drawing.*

Proof Let I be a feasible set of tasks for a path P, where each task $i \in I$ uses the first edge of P. We can construct a drawing for I as follows. We order the tasks in a non-increasing order with respect to the length of $p_i \cap P$, breaking ties arbitrarily. We consider the tasks in this order and, for the current task $j \in I$, we assign a height $h_i = \sum_{i < j} d_i$. Each task $i \in I$ corresponds to a rectangle whose projection on the height-axis is $[h_i, h_i + d_i)$, and projection on P is $p_i \cap P$. It corresponds to placing the newly drawn rectangle on top of the rectangles we have drawn so far.

Clearly, these rectangles are pairwise non-overlapping. We now show that the rectangles are drawn underneath the capacity profile. Consider an edge $e \in E$, and let $I_e \subseteq I$ be the subset of tasks using e. Due to our ordering procedure, the tasks in I_e are exactly the $|I_e|$ first tasks in I. Therefore the projection of all the corresponding rectangles on the height-axis is the interval $[0, \sum_{i \in I_e} d_i)$. That fits under the capacity profile, as $u_e \geq \sum_{i \in I : e \in p_i} d_i = \sum_{i \in I_e} d_i$. □

3.1.3 LP relaxation

Using this geometric viewpoint, we write an LP relaxation for intersecting instances of UFP-tree as follows. Recall that we have an $O(k)$-nice splitting $\mathcal{P}$ of the tree T. We add constraints to the relaxation to enforce that there is a representing drawing for the selected tasks on each path $P \in \mathcal{P}$; we remark that these constraints will automatically enforce the capacity constraints.

Variables. The IP has the following variables. For each task i, we have a variable $x_i \in \{0, 1\}$ with the interpretation that $x_i = 1$ if task i is in the solution. For each path

$P \in \mathcal{P}$, each task $i \in \mathcal{T}_P$, and each height h, we have a variable $y(i, h, P) \in \{0, 1\}$ with the interpretation that $y(i, h, P) = 1$ if the rectangle for task i is drawn at height h in the representing drawing for P. The allowed heights h are the ones satisfying $h + d_i \le u_e$ for each edge $e \in p_i \cap P$, i.e., such that the rectangle fits under the capacity profile. We introduce variables $y(i, h, P)$ only for such heights.

Constraints. For each $P \in \mathcal{P}$ and each $i \in \mathcal{T}_P$, we have a constraint

$$\sum_{h \text{ s.t. } \forall e \in p_i \cap P: \ h + d_i \le u_e} y(i, h, P) = x_i \ . \tag{1}$$

For each path $P \in \mathcal{P}$, we add constraints enforcing that in the representing drawing for P the rectangles do not overlap, by imposing constraints that any point q underneath the capacity profile is covered by at most one rectangle. Since all tasks use the first edge of P, it suffices to consider only points q on a vertical line going through the first edge of P, i.e., points $q = (x_0, h)$ where x_0 is any x-coordinate strictly between the first and the second vertex of P and h is an integral height that is at most the capacity of the first edge of P. We use $R(i, h, P)$ to denote a rectangle representing task i on P drawn at height h, i.e., $R(i, h, P)$ is a rectangle of height d_i, with a bottom y-coordinate h, and whose projection on the x-axis equal $p_i \cap P$. For each path $P \in \mathcal{P}$ and each point $q = (x_0, h)$ as described above we have a constraint

$$\sum_{i \in \mathcal{T}_P} \sum_{(h': \ q \in R(i, h', P))} y(i, h', P) \le 1. \tag{2}$$

We refer to the resulting LP relaxation as **Rectangle-LP** $(\mathcal{P})$. It clearly has pseudo-polynomial complexity. In the following, we show an $O(k)$-approximation based on LP rounding.

3.1.4 An LP rounding algorithm

Let (x, y) be a feasible solution to **Rectangle-LP** $(\mathcal{P})$. We use a randomized rounding with alteration strategy (as introduced in [9]) to select a subset of the tasks and a representing drawing for them on each path $P \in \mathcal{P}$. We proceed in two phases. In the selection phase, we pick a subset of the tasks and determine a drawing for them. The drawing in this phase may contain overlapping rectangles. In the alteration phase, we pick a subset of the selected tasks whose corresponding rectangles do not overlap.

Selection phase. We select a (not necessarily feasible) set S of tasks. For each task i, we add i to S independently at random with probability $x_i/(c_1 \cdot k)$, where $c_1 > 1$ is a sufficiently large constant that will be determined later. We refer to the tasks in the random sample S as the *selected* tasks. Additionally, for each task $i \in S$ and each path $P \in \mathcal{P}$ such that $i \in \mathcal{T}_P$, we choose a rectangle representing the drawing of i on P, as follows. We choose a height h for the rectangle independently at random according to the probability distribution $\{y(i, h, P)/x_i\}_h$. Note that the constraints (1) ensure that the values $y(i, h, P)/x_i$ form a probability distribution over the allowed heights h.

Let $h(i, P)$ be the height chosen for task i on the path P; we use the rectangle $R(i, h(i, P), P)$ to represent task i on the path P. Let $\mathcal{R}$ denote the resulting draw-

ing, i.e., $\mathcal{R}$ is the collection of rectangles selected for the tasks in S. Each rectangle $R(i, h, P)$ is in $\mathcal{R}$ with probability $x_i \cdot \frac{y(i,h,P)}{x_i} = y(i, h, P)$.

Alteration phase. In the alteration phase, we select a subset $S' \subseteq S$ of the tasks such that the rectangles $\mathcal{R}' \subseteq \mathcal{R}$ representing them on the paths are non-overlapping. Recall that we view the rectangles as open sets and thus two rectangles overlap iff they contain a common point in their interiors. We consider the paths of $\mathcal{P}$ in an arbitrary order. For each $P \in \mathcal{P}$, let $S(P) = \{i \in S : i \in \mathcal{T}_P\}$. Our goal is to choose a subset $S'(P) \subseteq S(P)$ such that the rectangles $\{R(i, h(i, P), P) : i \in S'(P)\}$ are non-overlapping. We choose the set of *accepted tasks* $S'(P)$ as follows.

We order the tasks in $S(P)$ in *non-increasing* order according to their demands, breaking ties arbitrarily. We consider the tasks in this order. Let i be the current task. We add i to $S'(P)$ if the rectangle $R(i, h(i, P), P)$ does not overlap with any of the rectangles $\{R(i', h(i', P), P) : i' \in S'(P)\}$ for the tasks we have accepted so far.

We refer to the tasks in $S'(P)$ as the tasks *accepted* on P, and the tasks in $S(P) - S'(P)$ as the tasks *rejected* on P. The following key lemma shows that each selected task $i \in S(P)$ is accepted with a constant probability. The main observation is that, for each task j, appearing before i in the ordering, if the rectangles $R(i, h(i, P), P)$ and $R(j, h(j, P), P)$ overlap, then $R(j, h(j, P), P)$ contains the top left or the bottom left corner of $R(i, h(i, P), P)$ since $d_j \geq d_i$; it now suffices to check the constraints only at two points.

Lemma 4 *For any path P and $i \in \mathcal{T}_P$, $\Pr[i \notin S'(P) \mid i \in S(P)] \leq 2/(c_1 \cdot k)$.*

Finally, we use the sets $\{S'(P) : P \in \mathcal{P}\}$ to select a subset $S' \subseteq S$ such that the rectangles $\mathcal{R}' \subseteq \mathcal{R}$ representing S' on each path of $\mathcal{P}$ are non-overlapping. We set $S' = \{i \in S : \forall_{P \in \mathcal{P} : i \in \mathcal{T}_P} \ i \in S'(P)\}$, i.e., a task is accepted if it was accepted for all paths. It follows from Lemma 4 and the union bound that each selected task is rejected with probability at most $|\{P \in \mathcal{P} : i \in \mathcal{T}_P\}| \cdot \frac{2}{c_1 k} \leq 1/2$ if c_1 is sufficiently large. More precisely, if $\mathcal{P}$ is ck-nice, then this happens when $c_1 \geq 4c$. We summarize the rounding step in the following lemma.

Lemma 5 *Consider a **UFP-tree** instance that has a K-nice splitting $\mathcal{P}$, and let (x, y) be a feasible solution to **Rectangle-LP** $(\mathcal{P})$. Let S be a random sample of the tasks such that each task i is in S independently at random with probability $x_i/(4K)$. There is a polynomial-time algorithm that constructs a feasible solution $S' \subseteq S$ such that, for each task i, $\Pr[i \in S' \mid i \in S] \geq 1/2$.*

For linear objective functions, this yields a pseudo-polynomial time LP-based $O(k)$-approximation for intersecting instances of **UFP-tree** and, with Lemma 1, a pseudo-polynomial time $O(k \log n)$-approximation for arbitrary instances of **UFP-tree**.

3.2 A polynomial-sized relaxation

In this section, we show how to turn the LP from the previous section to a polynomial sized one. Notice that the pseudo-polynomial running time is caused by the fact that the rectangles for the tasks in $\mathcal{T}$ can be drawn at pseudo-polynomially many heights.

We show that restricting to a polynomial sized set of heights incurs only an $O(1)$ factor loss in the approximation ratio.

Task classification. For a path $P \in \mathcal{P}$ and a task $i \in \mathcal{T}_P$, let $b_P(i) := \min_{e \in p_i \cap P} u_e$ be the *bottleneck capacity of i on P*. We say that a task $i \in \mathcal{T}_P$ is *big on P* if $d_i > \frac{1}{16} \cdot b_P(i)$. Otherwise we say that i is *small on P*.

Allowed heights. For each path $P \in \mathcal{P}$ and task $i \in \mathcal{T}_P$, we will now construct a set $\mathcal{H}(i, P)$ of *allowed heights* for drawing the rectangle corresponding to i on P. If i is big on P, we set $\mathcal{H}(i, P) = \{b_P(i) - d_i\}$, i.e., the only allowed height is obtained by drawing the rectangle for i as high as possible underneath the capacity profile. If i is small on P, for the integer j such that $b_P(i) \in [2^j, 2^{j+1})$, we set $\mathcal{H}(i, P) = \bigcup_{r \in \mathbb{N}_0 : r\lceil 2^{j-3}/n\rceil \leq 2^{j-1}} \{2^{j-1} + r\lceil 2^{j-3}/n\rceil\}$. We have $|\mathcal{H}(i, P)| \leq 8n$. Let $\mathcal{H}$ be the union of all sets $\mathcal{H}(i, P)$. By construction, $\mathcal{H}$ has polynomial size.

Restricted LP. Denote by Restricted-Rectangle-LP $(\mathcal{P}, \mathcal{H})$ the LP relaxation where we introduce variables $y(i, h, P)$ and the constraints (1) and (2) only for the heights $h \in \mathcal{H}$. As $\mathcal{H}$ has polynomial size, the size of Restricted-Rectangle-LP $(\mathcal{P}, \mathcal{H})$ is also polynomial. The following lemma shows that the LP restricted to these heights still admits a good fractional solution. Combining it with Lemma 5 yields the desired polynomial time approximation algorithm for linear UFP-tree.

Lemma 6 *For each feasible integral solution $\mathcal{T}' \subseteq \mathcal{T}$ of an instance of UFP-tree, there is a feasible fractional solution (x, y) for Restricted-Rectangle-LP $(\mathcal{P}, \mathcal{H})$ s.t. $(\forall i \in \mathcal{T}')x_i = \frac{1}{64}$.*

We devote the remainder of this subsection to proving Lemma 6. Let $\mathcal{T}' \subseteq \mathcal{T}$ be any feasible solution for an instance of UFP-tree. We define the fractional solution (x, y) as follows. First, for each task $i \in \mathcal{T}'$, we set $x_i := 1/64$; we set $x_i := 0$ for all other tasks. Thus it only remains to define the vector y. For each path P and each task $i \in \mathcal{T}'$ using P we identify a height $h(i, P) \in \mathcal{H}(i, P)$ such that if we set $y(i, h(i, P), P) := 1/64$ for each task $i \in \mathcal{T}'$ using P then all constraints of type (2) are satisfied. Note that the constraints of type (1) are then automatically satisfied.

Consider a task $i \in \mathcal{T}'$ that is big on P. In this case, only the height $b_P(i) - d_i$ is allowed for task i on the path P, and thus we set $h(i, P) := b_P(i) - d_i$. Lemma 4.4 in [7] implies that each point underneath the capacity profile of P can be overlapped by at most 32 rectangles corresponding to big tasks in $\mathcal{T}'$ when each of these rectangles is drawn at maximum height. This implies the following proposition.

Proposition 1 *Let $P \in \mathcal{P}$ and let $\mathcal{T}_B(P)$ denote all tasks in $\mathcal{T}'$ that are big on P. Then for each point $q = (x_0, h)$ such that x_0 is strictly between the first and the second vertex of P we have that*

$$\sum_{i \in \mathcal{T}_B(P)} \sum_{(h' : q \in R(i,h',P))} y(i, h', P) \leq 1/2. \tag{3}$$

We now define the heights for the small tasks. Let $j \in \mathbb{N}$ and let $\mathcal{T}_S^{(j)}(P)$ denote all tasks $i \in \mathcal{T}'$ that are small on P and that have the property that $b_P(i) \in [2^j, 2^{j+1})$. Due to the latter property and since all tasks use the first edge of P, there is an

edge $e \in P$ used by all tasks in $\mathcal{T}'$ s.t. $u_e < 2^{j+1}$. Therefore we must have that $\sum_{i \in \mathcal{T}_S^{(j)}(P)} d_i \leq 2^{j+1}$. Since they are small, for each task $i \in \mathcal{T}_S^{(j)}(P)$ it must hold that $d_i \leq 2^{j+1}/16 = 2^{j-3}$. We partition the tasks in $\mathcal{T}_S^{(j)}(P)$ greedily into 16 groups $\mathcal{T}_S^{(j,1)}(P), \ldots, \mathcal{T}_S^{(j,16)}(P)$ such that $\sum_{i \in \mathcal{T}_S^{(j,k)}(P)} d_i \leq 2^{j-2}$ for each $k \in \{1, \ldots, 16\}$ (we do not optimize the constants here in the interest of a cleaner presentation).

Lemma 7 *For each group $\mathcal{T}_S^{(j,k)}(P)$ we can assign a height $h(i, P) \in \mathcal{H}(i, P)$ to each task $i \in \mathcal{T}_S^{(j,k)}(P)$ such that $h(i, P) \in [2^{j-1}, 2^j)$, $h(i, P) + d_i \leq 2^j$, and the corresponding rectangles are non-overlapping.*

Proof Imagine that we increase the demand of each task $i \in \mathcal{T}_S^{(j,k)}(P)$ to the value d_i' that is the next higher integral multiple of $\lceil \frac{2^{j-3}}{n} \rceil$, i.e., $d_i' := \lceil d_i / \lceil \frac{2^{j-3}}{n} \rceil \rceil \cdot \lceil \frac{2^{j-3}}{n} \rceil$. Then

$$\sum_{i \in \mathcal{T}_S^{(j,k)}(P)} d_i' \leq d(\mathcal{T}_S^{(j,k)}(P)) + n \cdot \left\lceil \frac{2^{j-3}}{n} \right\rceil \leq 2^{j-2} + n \cdot \frac{2^{j-2}}{n} \leq 2^{j-1}.$$

Now consider the following drawing of $\mathcal{T}_S^{(j,k)}(P)$ using rectangles whose heights correspond to the new demands d_i'. We start drawing the rectangles at height 2^{j-1}. We consider the tasks in arbitrary order. Starting at height 2^{j-1}, we draw each task i using a rectangle of height d_i', and we place this rectangle on top of the rectangles we have already drawn. Let $h(i, P)$ denote the height of the rectangle for task i in the resulting drawing.

Since the demands d_i' are multiples of $\lceil 2^{j-3}/n \rceil$ and the starting height is 2^{j-1}, we have $h(i, P) = 2^{j-1} + r \lceil 2^{j-3}/n \rceil$ for some integer $r \in \{0, \ldots, \lfloor n/2^{j-3} \rfloor\}$. Therefore $h(i, P) \in \mathcal{H}(i, P)$. Moreover, for each task $i \in \mathcal{T}_S^{(j,k)}(P)$, we have $h(i, P) + d_i \leq 2^{j-1} + \sum_{i \in \mathcal{T}_S^{(j,k)}(P)} d_i' \leq 2^j$. □

For each task $i \in \mathcal{T}_S^{(j,k)}(P)$, we set $y(i, h(i, P), P) := 1/64$, where $h(i, P)$ is the height given by Lemma 7. Since the rectangles for tasks in a given group $\mathcal{T}_S^{(j)}(P)$ do not overlap and there are at most 16 groups, it follows that each point underneath the capacity profile is contained in at most 16 rectangles.

Proposition 2 *Let $P \in \mathcal{P}$. For each point $q = (x_0, h)$ such that x_0 is strictly between the first and the second vertex of P we have that*

$$\sum_{j \in \mathbb{N}} \sum_{i \in \mathcal{T}_S^{(j)}(P)} \sum_{(h' : q \in R(i,h',P))} y(i, h', P) \leq 1/2. \quad (4)$$

Propositions 1 and 2 imply that all constraints of type (2) are satisfied and thus (x, y) forms a feasible solution to Restricted-Rectangle-LP $(\mathcal{P}, \mathcal{H})$. Thus there is a fractional solution with a value of at least OPT/64. For a linear objective, the

rounding from Sect. 3.1 gives us an integral solution of value $\Omega(\mathrm{OPT}/(k \log n))$. For a submodular objective, we will show in Sect. 3.3 that, by combining the rounding from Sect. 3.1 with the CR framework of [15], we also obtain an integral solution with a value of $\Omega(\mathrm{OPT}/(k \log n))$.

3.3 Submodular objective via the CR scheme framework

In this section, we extend our results to submodular objectives by combining the results from the previous section with the framework from [15].

Let N be a finite ground set. Let $\mathcal{I} \subseteq 2^N$ be a family of subsets of N, and $\mathbf{P}_{\mathcal{I}}$ a convex relaxation for the constraints imposed by $\mathcal{I}$, such that $\mathbf{P}_{\mathcal{I}}$ is down-monotone and solvable.[5] Let $x \in \mathbf{P}_{\mathcal{I}}$ and let support$(x) = \{i \in N : x_i > 0\}$. For any $b \in [0, 1]$, let $b \cdot \mathbf{P}_{\mathcal{I}} = \{bx : x \in \mathbf{P}_{\mathcal{I}}\}$. Let $\mathbf{R}(x)$ be a random sample of N such that each element $i \in N$ is in $\mathbf{R}(x)$ independently at random with probability x_i. For a set function $f : 2^N \to \mathbb{R}_+$ let $F : [0, 1]^N \to \mathbb{R}_+$ denote the *multilinear extension* of f, which is defined as $F(x) := \mathbb{E}[f(\mathbf{R}(x))]$.

Definition 2 ([15]) For $b, c \in [0, 1]$, a (b, c)-balanced CR scheme π for a polytope $\mathbf{P}_{\mathcal{I}}$ is a procedure that for every $x \in b \cdot \mathbf{P}_{\mathcal{I}}$ and $A \subseteq N$ returns a random set $\pi_x(A)$ satisfying

 (i) $\pi_x(A) \subseteq$ support$(x) \cap A$ and $\pi_x(A) \in \mathcal{I}$ with probability 1, and
(ii) for all $i \in$ support(x), $\Pr[i \in \pi_x(\mathbf{R}(x)) \mid i \in \mathbf{R}(x)] \geq c$.

We use the CR schemes as in [15]. First, we compute a vector x^* with $F(x^*) \geq \Omega(\max\{F(x') : x' \in \mathbf{P}_{\mathcal{I}}\})$. Then, we compute a random sample $\mathbf{R}(x)$ with $x := b \cdot x^*$. We apply the CR scheme π and obtain the set $\pi_x(\mathbf{R}(x))$. We know that for each element i we have that $\Pr[i \in \mathbf{R}(x)] = b \cdot x_i^*$ and $\Pr[i \in \pi_x(\mathbf{R}(x)) \mid i \in \mathbf{R}(x)] \geq c$. Thus, $\Pr[i \in \pi_x(\mathbf{R}(x))] \geq bc \cdot x_i^*$ which can be used to show that $\mathbb{E}[f(\pi_x(\mathbf{R}(x)))] \geq \Theta(bc) \cdot \max\{F(x') : x' \in \mathbf{P}_{\mathcal{I}}\}$.

Theorem 4 *([15]) Let $f : 2^N \to \mathbb{R}_+$ be a submodular function. Let $\mathcal{I} \subseteq 2^N$ be a family of feasible solutions and let $\mathbf{P}_{\mathcal{I}} \subseteq [0, 1]^N$ be a convex relaxation for $\mathcal{I}$ that is down-monotone and solvable. Suppose that there is a (b, c)-balanced CR scheme for $\mathbf{P}_{\mathcal{I}}$. Then there is a polynomial time randomized algorithm that constructs a solution $I \in \mathcal{I}$ such that*

$$\mathbb{E}[f(I)] \geq \Theta(bc) \cdot \max\{F(x) : x \in \mathbf{P}_{\mathcal{I}}\}.$$

To apply the above framework, let $\mathbf{P}$ denote the set of points x for which there exists a vector y such that (x, y) is contained in the polytope defined by Restricted-Rectangle-LP $(\mathcal{P}, \mathcal{H})$. Clearly, $\mathbf{P}$ is down-monotone and solvable. Similarly as in the case of linear objective functions, $\mathbf{P}$ contains a fractional point with large profit

---

[5] We call a polytope $\mathbf{P} \subseteq [0, 1]^N$ *down-monotone* if for all $\mathbf{z}, \mathbf{z}' \in [0, 1]^N$ we have that $\mathbf{z} \leq \mathbf{z}'$ and $\mathbf{z}' \in \mathbf{P}$ implies that $\mathbf{z} \in \mathbf{P}$. The polytope is *solvable* if one can optimize any linear function over $\mathbf{P}$ in polynomial time.

according to F: Let $\mathcal{T}^*$ be an optimal integral solution. By Lemma 6, $\frac{1}{64} \cdot \mathbf{1}_{\mathcal{T}^*} \in \mathbf{P}$. Moreover, it is straightforward to verify that $F\left(\frac{1}{64} \cdot \mathbf{1}_{\mathcal{T}^*}\right) \geq \frac{1}{64} f(\mathcal{T}^*)$. So $\max\{F(x) : x \in \mathbf{P}_{\mathcal{I}}\} = \Omega(\text{OPT})$.

By Lemma 5, there is a $(1/\Theta(k), 1/2)$-balanced CR scheme for $\mathbf{P}$. Therefore we can apply Theorem 4 to obtain our main result for Submodular UFP-tree.

Theorem 5 *There is a polynomial time $O(k)$ approximation algorithm for Submodular UFP-tree on intersecting instances and, therefore, an $O(k \log n)$ approximation for arbitrary instances, where k is the pathwidth of the tree.*

4 Applications

We describe how our techniques can be applied to obtain approximation algorithms for Submodular Bag-UFP-tree and Submodular Bag-SAP-tree.

Bag-UFP. In the Submodular Bag-UFP-tree we have the same input as for UFP-tree and additionally the input tasks $\mathcal{T}$ are partitioned into *bags* $\mathcal{T}_1, \ldots, \mathcal{T}_s$ and we are allowed to select at most one task per bag. We start with the same approach as for Submodular UFP-tree and invoke Lemma 1 to reduce the general case to the case of intersecting instances at the expense of a factor $O(\log n)$ in the approximation ratio. Then we formulate the problem using Rectangle-LP $(\mathcal{P})$ and add to it the set of constraints

$$\sum_{i \in \mathcal{T}_j} x_i \leq 1 \quad \forall j \in \{1, \ldots, s\}. \tag{5}$$

Similarly to the case of UFP-tree, we round the resulting LP using randomized rounding with alteration. In the selection phase, we again sample each task i with probability $x_i/(c_1' \cdot k)$ for some large constant c_1' and choose the heights for its rectangles according to the distribution given by $y(i, h, P)/x_i$. In the alteration phase, we accept a task i in a bag $\mathcal{T}_j$ if none of its corresponding rectangles overlaps with a rectangle of a previously accepted task *and* if no task from $\mathcal{T}_j$ has been accepted before. Similarly as in Lemma 5 we can show that $\Pr[i \in S' | i \in S] \geq 1/2$ (as each selected task i is rejected due to overlapping with a previously accepted task with probability at most $|\{P \in \mathcal{P} : i \in \mathcal{T}_P\}| \cdot \frac{2}{c_1' k}$, and is rejected due to another task from the same bag being previously accepted with probability at most $\frac{2}{c_1' k}$). This yields a CR scheme and, thus, an algorithm for Submodular Bag-UFP-tree. As described in Sect. 3.2 we can restrict the set of allowed heights to polynomial size (observe that the reasoning there only argues about tasks from an optimal solution.) The following theorem summarizes our contribution on Submodular Bag-UFP-Tree.

Theorem 6 *There is a polynomial time $O(k \cdot \log n)$-approximation algorithm for Submodular Bag-UFP-tree where k is the pathwidth of an input tree.*

We note that one can model submodular bagUFP with an instance of submodular UFP for a different submodular function: in case that the computed set of tasks contains

several tasks from a bag, this new function selects only one task from each such bag such that the profit of the original submodular function is maximized. However, it is not clear how to evaluate this new submodular function efficiently if one has only oracle-access to the original submodular function. Therefore, we give a separate algorithm for Submodular Bag-UFP-Tree.

Storage Allocation. The input to SAP-tree is that of UFP-tree, with an additional requirement that for each selected task i in $\mathcal{T}' \subseteq \mathcal{T}$ we have to compute a value $h(i) \geq 0$ such that $h(i) + d_i \leq u_e$ for each edge $e \in p_i$, and $[h(i), h(i) + d_i) \cap [h(i'), h(i') + d_{i'}) = \emptyset$ for any two tasks $i, i' \in \mathcal{T}'$ with $p_i \cap p_{i'} \neq \emptyset$. This gives each task $i \in \mathcal{T}'$ the portion $[h(i), h(i) + d_i)$ of the resource spectrum.

We invoke Lemma 1 to reduce the instance to a collection of intersecting instances. In contrast to UFP-tree,

on such instances it essentially suffices to focus on edges incident to the root when checking feasibility.

Proposition 3 *For an intersecting instance of SAP-tree, a pair $(\mathcal{T}', h)$ with $\mathcal{T}' \subseteq \mathcal{T}$ is a feasible solution if and only if*

- *for any edge e incident to the root and for any two tasks $i, i' \in \mathcal{T}'$ such that $p_i, p_{i'} \ni e$ we have $[h(i), h(i) + d_i) \cap [h(i'), h(i') + d_{i'}) = \emptyset$, and*
- *for each task $i \in \mathcal{T}'$ we have that $h(i) \leq u_e - d_i$ for each edge $e \in p_i$.*

We set up a similar LP as Rectangle-LP $(\mathcal{P})$, however, taking into account that we model SAP-tree, rather than UFP-tree. To keep the notation close to Rectangle-LP $(\mathcal{P})$, we introduce a variable x_i for each task $i \in \mathcal{T}$ and a variable $y(i, h)$ for each task $i \in \mathcal{T}$ and a height $h \in \mathbb{N}$ such that $h \leq u_e - d_i$ for each edge $e \in p_i$ (as in the second property of Proposition 3), indicating that task i is selected and drawn at height h. For each task $i \in \mathcal{T}$, we have the constraints

$$\sum_{h: h \leq u_e - d_i \ \forall e \in p_i \cap P} y(i, h) = x_i \tag{6}$$

to enforce that the values $y(i, h)/x_i$ form a probability distribution for the possible heights of task i. For each task $i \in \mathcal{T}$ we add the constraint $x_i \leq 1$, to ensure that each task is selected at most once. Then, for each edge e incident to the root and each integer q, we add the constraint

$$\sum_{(i \in \mathcal{T}: \ p_i \ni e)} \sum_{(h': \ q \in [h', h' + d_i))} y(i, h') \leq 1. \tag{7}$$

Intuitively, the constraints (7) model that on edge e the rectangles corresponding to the selected tasks do not overlap (as required by the first property of Proposition 3). Denote by SAP-LP $(\mathcal{P})$ the resulting linear program. The key difference to Rectangle-LP $(\mathcal{P})$ is now that each task no longer appears on up to k different paths of $\mathcal{P}$ with up to k different rectangles with different heights, but only on two edges. Therefore, we can round SAP-LP $(\mathcal{P})$ using randomized rounding with alteration by losing only a *constant* factor, instead of a factor $O(k)$.

Formally, we sample a set of tasks S such that each task i is added to S independently at random with probability x_i/c_1'' (in contrast to the probability $x_i/(c_1 \cdot k)$ as in the case of UFP-Tree). Like above, we choose a height h for i independently at random according to the probability distribution $y(i, h)/x_i$. In the alteration phase, again we order the tasks non-increasingly by demands and reject a task if on one of its two edges incident to the root it conflicts with a previously accepted task, i.e., a constraint of type (7) would be violated. Denote by S' the set of accepted tasks.

Lemma 8 *For each task $i \in \mathcal{T}$ we have that* $\Pr[i \notin S' | i \in S] \leq 1/c_2''$ *for some constant* $c_2'' \in O(c_1'')$.

By choosing c_1'' sufficiently large we ensure that $\Pr[i \in S' | i \in S] \leq 1/2$ and with a similar reasoning as for UFP-tree we obtain a $O(1)$-approximation algorithm for intersecting instances of SAP-tree. We can also extend the above argumentation to submodular objective functions and to the setting with bag constraints, i.e., Submodular Bag-SAP-tree. Also, we can apply a similar reasoning as in Sect. 3.2 to get an LP-formulation with polynomial size.

Theorem 7 *There is a polynomial time $O(\log n)$-approximation algorithm for Submodular Bag-SAP-tree.*

5 Upward instances

5.1 An algorithm

In this section we first prove Theorem 2. Assume that we are given an upward instance of UFP-tree on a tree T with a set of tasks $\mathcal{T}$. By losing a factor $1 + O(\epsilon)$ in the approximation ratio we can assume that for each $i \in \mathcal{T}$ we have that w_i is a power of $1 + \epsilon$ and $w_i \in [1, n/\epsilon)$ (tasks with a weight of at most $\frac{\epsilon}{n} \cdot \max_i w_i$ can be discarded by losing only a factor of $1 + \epsilon$). We first show that we can assume w.l.o.g. that each vertex in the tree T has a degree of at most three.

Lemma 9 *There is a polynomial time algorithm that for any instance of UFP-tree on an n-vertex tree T produces an equivalent instance on an $O(n)$-vertex tree T', s.t. each vertex of T' has a degree of at most 3.*

Proof Suppose T has a vertex v with degree $d \geq 4$. Let $\{v_1, \ldots, v_d\}$ be neighbors of v and let v_1 be the parent of v. Replace v by a gadget containing v, a new vertex v', and an edge vv' with infinite capacity. We keep the edges vv_1 and vv_2 with the same capacities as before, and all vertices in $\{v_3, \ldots, v_d\}$ are defined to be children of v', where each edge $v'v_j$ has capacity c_{vv_j} for $j \geq 3$.

We repeat this operation until every vertex has a degree of at most three. □

For simplicity, assume that each vertex in T is either a leaf or has a degree of *exactly* three (enforcing this condition can increase the number of vertices by at most n). Let v_r denote the root of T. We devise a recursive procedure with $O(\log n)$ levels. In the first step, we identify a (center) vertex v_m of T with the property that $T \backslash \{v_m\}$ has three connected components, each of them having at most $n/2$ vertices. Such a vertex

always exists. Let V_1 denote the vertices of T in the connected component of $T \backslash \{v_m\}$ containing v_r, and V_2, V_3 the vertices in the other two components of T, respectively. For $j \in \{1, 2, 3\}$ let T_j denote $T[V_j \cup \{v_m\}]$. Denote by P the path from v_m to v_r.

Approximate profiles. We partition $\mathcal{T}$ into classes such that the tasks in each class have roughly the same demand and profit. For each pair (k, ℓ) with $k \in \{\lfloor \log_{1+\epsilon} d_{\min} \rfloor, \ldots, \lceil \log_{1+\epsilon} d_{\max} \rceil\}$, $\ell \in \{1, \ldots, \lceil \log_{1+\epsilon}(n/\epsilon) \rceil\}$, define a set of tasks $\mathcal{T}^{(k,\ell)} = \{i \in \mathcal{T} | (1 + \epsilon)^k \leq d_i < (1 + \epsilon)^{k+1} \wedge (1 + \epsilon)^\ell \leq w_i < (1 + \epsilon)^{\ell+1}\}$. Notice that there are $O_\epsilon(\log(d_{\max}/d_{\min}) \log(n))$ sets. For each pair $(T_j, \mathcal{T}^{(k,\ell)})$ with $j \in \{2, 3\}$ we guess an approximation of how much capacity is used on the edges of P by tasks from $\mathcal{T}^{(k,\ell)}$ having a start vertex in T_j.

Denote by $OPT^{(k,\ell,j)}$ the optimal solution restricted to tasks in $\mathcal{T}^{(k,\ell)}$ which start in T_j and use an edge of P. For each edge $e \in P$ we define a value $u_e^{(k,\ell,j)} := \sum_{i \in OPT^{(k,\ell,j)}: p_i \ni e} d_i$, which is the capacity used by $OPT^{(k,\ell,j)}$ on e. Observe that when traversing P from v_m towards v_r, the values of $u_e^{(k,\ell,j)}$ are non-increasing. These values yield a non-increasing profile $u^{(k,\ell,j)}$ on P with possibly polynomially many different values. We claim that there is a much simpler profile $\bar{u}^{(k,\ell,j)}$ that attains only $O_\epsilon(\log n)$ different values from some quasi-polynomial size set, underestimates the real profile, but is still large enough s.t. we can fit a $(1+\epsilon/\log n)$-approximate subset of $OPT^{(k,\ell,j)}$ into $\bar{u}^{(k,\ell,j)}$. The following lemma can be shown with a similar reasoning as in [4].

Lemma 10 *Let $k, \ell \in \mathbb{N}$ and $j \in \{2, 3\}$. There is a partition of the path P into $s = O(\log n/\epsilon)$ subpaths $P_1^{(k,\ell,j)}, \ldots, P_s^{(k,\ell,j)}$, a function $\bar{u}^{(k,\ell,j)} : P \to \mathbb{N}$, and a set of tasks $A^{(k,\ell,j)} \subseteq OPT^{(k,\ell,j)}$ such that*

- $w(A^{(k,\ell,j)}) \geq (1 + \frac{\epsilon}{\log n})^{-1} \cdot w(OPT^{(k,\ell,j)})$,
- *for each $s' \leq s$ and each edge $e \in P_{s'}^{(k,\ell,j)}$ it holds that $\sum_{i \in A^{(k,\ell,j)}: p_i \ni e} d_i \leq \bar{u}_e^{(k,\ell,j)} \leq u_e^{(k,\ell,j)}$,*
- *for each $s' \leq s$ we have that for any two edges $e, e' \in P_{s'}^{(k,\ell,j)}$ it holds that $\bar{u}_e^{(k,\ell,j)} = \bar{u}_{e'}^{(k,\ell,j)}$, and*
- *the values of $\bar{u}_e^{(k,\ell,j)}$ are from a quasi-polynomial size set.*

Proof Fix $s = \Theta(\log n/\epsilon)$. We can assume that $|OPT^{(k,\ell,j)}| \geq s$, as otherwise setting $A^{(k,\ell,j)} = OPT^{(k,\ell,j)}$ and $\bar{u}^{(k,\ell,j)}(e) = u^{(k,\ell,j)}(e')$ solves the problem. In that case each value $\bar{u}^{(k,\ell,j)}(e)$ is a sum of at most s demands d_i, and therefore the number of possible values of $\bar{u}^{(k,\ell,j)}(e)$ is quasi-polynomial in n.

Let $D = \sum_{i \in OPT^{(k,\ell,j)}} d_i$ be the total demand of the tasks in $OPT^{(k,\ell,j)}$. Notice that $D \in [(1 + \epsilon)^k, n \cdot (1 + \epsilon)^{k+1})$.

Define $d(e) = \sum_{i \in OPT^{(k,\ell,j)}: e \ni p_i} d_i$ for any edge $e \in P$. For $s' = 1, \ldots, s$ we set $P_{s'}^{(k,\ell,j)} = \{e \in P : d(e) \in [\frac{s-s'}{s}D, \frac{s-s'+1}{s}D)\}$. As the total capacity of the tasks of $OPT^{(k,\ell,j)}$ on the edges of P is non-increasing, each $P_{s'}^{(k,\ell,j)}$ forms a path. Notice that some of the paths $P_{s'}^{(k,\ell,j)}$ might be empty.

We define the function $\bar{u}^{(k,\ell,j)} : P \to \mathbb{N}$ as follows. Let P' be the path containing an edge $e \in P$. Then $\bar{u}^{(k,\ell,j)}(e) := \max\{(1+\epsilon)^k \cdot (1+\frac{1}{s})^i : i \in \mathbb{N} \text{ and } (1+\epsilon)^k \cdot (1+\frac{1}{s})^i \leq$

$\min_{e' \in P'} d(e')\}$, i.e., we take the value $\min_{e' \in P'} d(e')$ and round it down to the nearest value of the form $(1 + \epsilon)^k \cdot (1 + \frac{1}{s})^i$. From this, we obtain that for any edge $e \in P$, $\bar{u}^{(k,\ell,j)}(e) \leq d(e) = u^{(k,\ell,j)}(e)$, and for any pair of edges e, e' belonging to the same path P' we have $\bar{u}^{(k,\ell,j)}(e) = \bar{u}^{(k,\ell,j)}(e')$. The number of possible values of $\bar{u}^{(k,\ell,j)}(e)$ is $O(\log^2 n/\epsilon)$.

We define sets $A^{(k,\ell,j)}$ as follows. We order the tasks $i \in OPT^{(k,\ell,j)}$ non-increasingly by $|p_i \cap P|$, breaking ties arbitrarily. Then, we greedily select the set of tasks B from $OPT^{(k,\ell,j)}$, until $d(B) \geq 2D/s$. We set $A^{(k,\ell,j)} = OPT^{(k,\ell,j)} \setminus B$.

We will show that for any $e \in P$, it holds $\sum_{i \in A^{(k,\ell,j)}: p_i \ni e} d_i \leq \bar{u}^{(k,\ell,j)}(e)$. Let e be an edge belonging to $P' = \{e' \in P : d(e') \in [\frac{s-s'}{s}D, \frac{s-s'+1}{s}D)\}$. Then $d(e) < \frac{s-s'+1}{s}D$ and $\bar{u}^{(k,\ell,j)}(e) \geq \frac{s-s'}{s}D/(1 + \frac{1}{s}) > (d(e) - D/s)/(1 + \frac{1}{s}) \geq d(e) - 2D/s$. If all tasks from the set B use e, then $\sum_{i \in A^{(k,\ell,j)}: p_i \ni e} d_i = \sum_{i \in OPT^{(k,\ell,j)}: p_i \ni e} d_i - \sum_{i \in B} d_i \leq d(e) - 2D/s < \bar{u}^{(k,\ell,j)}(e)$. Otherwise, the only tasks of $OPT^{(k,\ell,j)}$ using e are the tasks in B, and $\sum_{i \in A^{(k,\ell,j)}: p_i \ni e} d_i = 0$.

It remains to show that $w(A^{(k,\ell,j)}) \geq (1 + \frac{\epsilon}{\log n})^{-1} \cdot w(OPT^{(k,\ell,j)})$. As all tasks in $OPT^{(k,\ell,j)}$ have approximately the same demand and weight (up to a factor $(1 + \epsilon)$), we get $|B| \leq 4|OPT^{(k,\ell,j)}|/s$ and $w(B) \leq 8w(OPT^{(k,\ell,j)})/s$. As $w(A^{(k,\ell,j)}) = w(OPT^{(k,\ell,j)}) - w(B)$ and $s = \Theta(\log n/\epsilon)$, the claim follows. □

In parallel we guess an approximative capacity profile due to Lemma 10 for each pair $(T_j, \mathcal{T}^{(k,\ell)})$.

There are only $n^{(\frac{1}{\epsilon} \log n)^{O(1)} \log(d_{\max}/d_{\min})}$ possible guesses.

Recursion. We recurse on the following three subproblems.

- The first subproblem is defined by T_1 and all tasks whose start and end vertices lie in T_1. For the edge capacities, we take into account that the edges of P have smaller capacity since the guessed capacity profiles use up some capacity. Each edge $e \in P$ has then a residual capacity of $u_e - \sum_{j=2,3} \sum_{k,\ell} \bar{u}_e^{(k,\ell,j)}$.
- The second subproblem is defined by $T_2 \cup P$ and all tasks having their start vertex in T_2. On each edge in T_2 we have its full capacity available, on each edge $e \in P$ the available capacity is the sum of the capacities due to the previously guessed profiles: $\sum_{k,\ell} \bar{u}_e^{(k,\ell,2)}$.
- The third subproblem is defined on $T_3 \cup P$ symmetrically to the second.

This defines the first step of our recursion. When we proceed, each arising subproblem is defined on a subtree T' of the input tree T and a path P' which is the path from v_r to the vertex in T' that is closest to v_r.

Each edge of $T' \cup P'$ has a given a capacity (that might be smaller than in T). Additionally, we are given a set of tasks $\mathcal{T}'$ that is exactly the set of input tasks that have their respective start vertex in T'. Our goal is to find a good approximation for this subproblem. When we recurse on it, we perform the procedure described above with the center vertex v_m' of T', i.e., we guess the approximative profiles on the path between v_m' and v_r in $T' \cup P'$ and then recurse on the three arising subproblems.[6]

---

[6] Note that selecting the center vertex of T' is *not* the same as selecting that of $T' \cup P'$.

We do this recursive procedure until we obtain subproblems in which T' consists of only one single edge. The resulting instance is equivalent to an instance of Unsplittable Flow on Paths. We compute a $(1 + \epsilon)$-approximation for it in time $n^{O_\epsilon(\log n)}$ using the algorithm in [5].

Analysis. When given one of the subproblems, the number of possible guesses for the profiles on the path between the center v'_m of T' and v_r is at most $n^{(\frac{1}{\epsilon} \log n)^{O(1)} \log(d_{\max}/d_{\min})}$. Thus, each node in the recursion tree has at most $n^{(\frac{1}{\epsilon} \log n)^{O(1)} \log(d_{\max}/d_{\min})}$ children. As in each recursive step we recurse on the center vertex of the respective tree T', our recursion depth is $O(\log n)$. Finally, solving the leaf subproblems requires at most $n^{O_\epsilon(\log n)}$ time, see [5].

Lemma 11 *The running time of the above algorithm is upper bounded by* $n^{O_\epsilon(\log n)(\frac{1}{\epsilon})^{O(1)} \log(d_{\max}/d_{\min})}$.

For the approximation factor, each level of the recursion incurs a loss by a factor of at most $(1 + \frac{\epsilon}{\log n})^{-1}$. Since our recursion depth is $O(\log n)$ this yields an approximation ratio of $(1 + O(\epsilon))$, completing the proof of Theorem 2.

5.2 Hardness

In this section we prove Theorem 3, based on the assumption that SAT is not in DTIME$(2^{o(n)})$. We start from the near-linear PCP theorem. There are many PCPs in the literature whose proof lengths are near-linear in the size of the original SAT formula, e.g., Dinur's PCP [16]. The following theorem can be derived by combining known tools in the literature, more precisely the PCP theorem and the bounded occurrences reduction from [3,16,28].

Theorem 8 *([3,16,28]) There is a universal constant $\epsilon_0 > 0$ such that there is a polynomial time algorithm that transforms any n-variable SAT formula ϕ into another formula ϕ' satisfying the following properties*

- ϕ' *is a 3SAT formula and each variable appears at most* 3 *times, i.e., it is a 3SAT(3) formula.*
- ϕ' *has at most $O(n \, \text{poly} \log n)$ variables and clauses.*
- *(Completeness:) If ϕ is satisfiable, then ϕ' is satisfiable.*
- *(Soundness:) If ϕ is not satisfiable, then any assignment to ϕ' satisfies at most* $(1 - \epsilon_0)$ *fraction of clauses.*

For a CNF formula ϕ, we denote by $\mathsf{SAT}(\phi)$ the maximum number of clauses that can be satisfied by any assignment.

Theorem 9 *There is a polynomial time reduction that transforms any 3SAT(3) formula ϕ' with N variables and M clauses into a UFP-tree upward instance $(T, \mathcal{T})$ such that* $\mathrm{OPT}(T, \mathcal{T}') = 4N + \mathsf{SAT}(\phi')$. *Moreover, the demand of every task $i \in \mathcal{T}'$ satisfies* $d_i \in [1, 2^{O(M)}]$.

We prove this theorem in the next subsection. We show now that this theorem yields our hardness result. Start from an n-variable SAT formula ϕ, invoke Theorem 8

to obtain ϕ', and then Theorem 9 to obtain $(T, \mathcal{T})$. Our reduction guarantees that $|T|, |\mathcal{T}| \leq n^{O(1)}$ and $\frac{d_{\max}}{d_{\min}} = 2^{O(M)} \leq 2^{n \, \text{poly} \log n}$.

All these steps so far have taken a polynomial running time. Assume for contradiction that there is an algorithm $\mathcal{A}$, that is an approximation scheme for UFP-trees running in time (for constant ϵ) $2^{\text{poly} \log |\mathcal{T}'|} n^{\log^{1-\delta}\left(\frac{d_{\max}}{d_{\min}}\right)}$. We have that $\log d_{\max}/d_{\min} = O(M) \leq n \, \text{poly} \log n$, and therefore, $\log^{1-\delta}(d_{\max}/d_{\min}) \leq n^{1-\delta/2}$ for sufficiently large n. Therefore, the running time of invoking the algorithm $\mathcal{A}$ is $N^{\text{poly} \log N \log^{1-\delta}(d_{\max}/d_{\min})} \leq 2^{o(n)}$ for sufficiently large n. Moreover,

- in the completeness case, we have $\text{OPT}(T, \mathcal{T}) = 4N + M$, and
- in the soundness case, we have $\text{OPT}(T, \mathcal{T}) < 4N + (1 - \epsilon_0)M < (1 - \epsilon/8)(4N + M)$. The second inequality follows from the fact that our formula ϕ' is a 3SAT(3) formula, so we have that $N \leq M \leq 3N$, which implies that $4N + (1 - \epsilon_0)M = 4N - \epsilon_0 M/2 + (1 - \epsilon_0/2)M \leq (4N - \epsilon_0 N/2) + (1 - \epsilon_0/2)M \leq (1 - \epsilon_0/8)(4N + M)$.

Since there is a gap of $(1 - \epsilon_0/8)$, running the approximation scheme $\mathcal{A}$ on $(T, \mathcal{T}')$ can distinguish the two cases in time $2^{o(n)}$, contradicting the ETH.

5.3 Proof of Theorem 9

Suppose that we are given a 3-BOUNDED-3-SAT formula ϕ with n variables $x_1, \ldots, x_n$ and m clauses $\{C_1, \ldots, C_m\} =: \mathcal{C}$ with the property that every literal appears at most three times. Note that this implies that $m \leq 3n$. We create an instance of UFP-tree. Our tree can be decomposed into a path P and a set of edges E' such that every edge in E' is connected to one vertex on P and each vertex on P is connected to at most one edge of E', see Fig. 2. The path P has three parts. The first part has m vertices and each vertex corresponds to exactly one clause $C \in \mathcal{C}$. For each clause $C \in \mathcal{C}$ we denote by v_C the vertex for C. The second part has exactly n vertices and each vertex corresponds to exactly one variable. For each variable x_j we denote by v_j the vertex corresponding to x_j in the second part. The third part of P has $2n$ vertices and for each variable x_j there is one vertex that we denote by v_{j+} and one vertex that we denote by v_{j-}. Additionally, each vertex in the third part is connected to exactly one edge in E'. For each vertex v_{j+} (resp. v_{j-}) in the third part of P, denote by v'_{j+} (resp. v'_{j-}) the corresponding vertex that it is connected to by an edge in E'.

Now we define the input tasks. First, for each clause we define three tasks. Let C be a clause containing the three variables x_j, x_k, x_ℓ (possibly in a negated form). We define three tasks for C: $i(C, j), i(C, k)$ and $i(C, \ell)$. Intuitively, if ϕ is satisfiable then we want to select the task among the three of them that corresponds to the variable satisfying C. All three tasks $i(C, j), i(C, k), i(C, \ell)$ have a weight of 1 and their start vertex is v_C. If the variable x_j appears positively in C then the end vertex of $i(C, j)$ is defined to be v'_{j+}, otherwise the end vertex of $i(C, j)$ is v'_{j-}. We define the end vertices of $i(C, k), i(C, \ell)$ accordingly. We do this procedure with all clauses in $\mathcal{C}$. We set the edge capacities on the first part of P and the task demands to ensure that

- for each clause C containing the three variables x_j, x_k, x_ℓ any feasible solution selects at most one of the tasks $i(C, j), i(C, k), i(C, \ell)$,
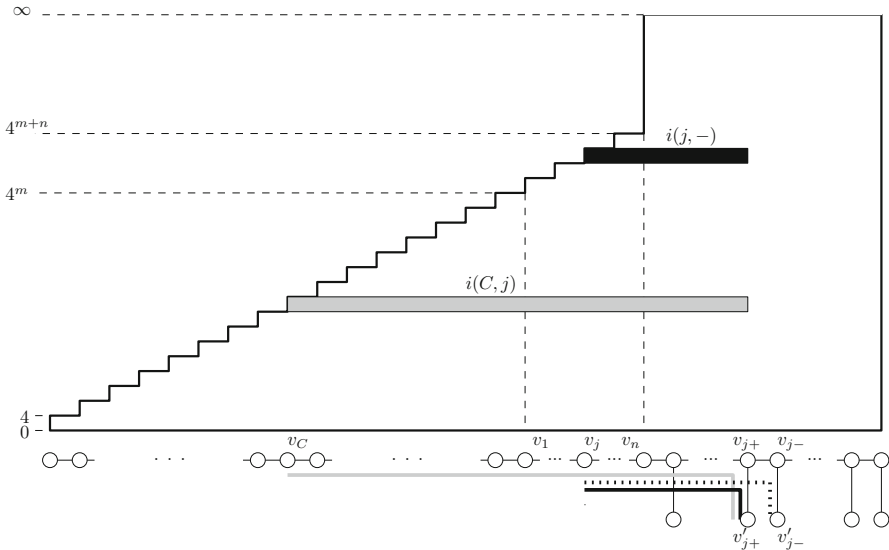
Fig. 2 A sketch of the reduction. The plot shows the capacity profile on the edges of the edges in E, drawn in logarithmic scale. Gray and black lines denote the paths of the tasks $i(C, j)$ and $i(j, -)$, respectively. The black dotted line shows the path of the task $i(j, +)$

- if we select exactly one task corresponding to each clause (no matter which one) then we do not violate the edge capacities on the first part of P.

We achieve this by setting the capacity of each edge $(v_{C_{m'}}, v_{C_{m'+1}})$ to be $4^{m'}$ and the capacity of the edge (v_{C_m}, v_1) to be 4^m. We set the demand of each of the three tasks corresponding to the clause $C_{m'}$ to be $\frac{3}{4} \cdot 4^{m'}$. It is clear then that any feasible solution can select at most one of the tasks for each clause. Moreover, the second property above follows from a geometric sum argument. Note that here we crucially need that the input values do not need to be quasi-polynomially bounded.

We define two tasks for each variable x_j, one task $i(j, +)$ and one task $i(j, -)$, both with a weight of 4. The start vertex of both of them is v_j and the end vertex of $i(j, +)$ is v'_{j-} and the end vertex of $i(j, -)$ if v'_{j+}. Intuitively, if ϕ is satisfiable and x_j is set to true in the satisfying assignment then we want to select the task $i(j, +)$, if x_j is set to false in the satisfying assignment then we want to select $i(j, -)$.

We set the edge capacities on the second part of P and the demands of the tasks $i(j, +), i(j, -)$ (for all variables x_j) such that

- for each x_j any feasible solution selects only one of the tasks $i(j, +), i(j, -)$,
- if we select exactly one task corresponding to each clause (no matter which one) and exactly one task corresponding to each variable then we do not violate the edge capacities on the second part of P.

We achieve this by setting the capacity of each edge (v_j, v_{j+1}) to be $4^m \cdot 4^j$ and the demand of each task $i(j, +)$ and $i(j, -)$ to be $\frac{3}{4} \cdot 4^m \cdot 4^j$. Like above, it is clear that at most one of the tasks $i(j, +)$ and $i(j, -)$ can be selected in a feasible solution. Also, the second property can be verified by a geometric sum argument.

We set the edge capacities of the third part of P to ∞. We define the capacity of each edge $(v_{j+}, v'_{j+}) \in E'$ (of $(v_{j-}, v'_{j-}) \in E'$) such that

- if $i(j, -)$ (resp. $i(j, +)$) is selected then it uses the whole capacity of (v_{j+}, v'_{j+}) (resp. of (v_{j-}, v'_{j-}))
- if $i(j, -)$ (resp. $i(j, +)$) is not selected then all three other tasks using (v_{j+}, v'_{j+}) (resp. of (v_{j-}, v'_{j-})) can be selected.

To this end, we set the capacity of (v_{j+}, v'_{j+}) to $\frac{3}{4} \cdot 4^m \cdot 4^j$. Since a total demand of tasks is at most $3 \cdot 4^m$ the second property also holds.

Lemma 12 *If ϕ is satisfiable, then the optimal solution to our* UFP-tree *instance has a value of at least $4n + m$.*

Proof Consider a satisfying assignment of ϕ. Let x_j be a variable. If in the assignment x_j is set to true then we select the task $i(j, +)$. Otherwise, we select the task $i(j, -)$. For each clause there must be one variable that satisfies the clause. Consider a clause C that is satisfied by a variable x_j. We select the task $i(C, j)$. The total profit of this solution is $4n + m$. It remains to argue that the solution is feasible. On the edges on P we satisfy the capacity constraints since we selected only one task corresponding to each variable and at most one task for each clause. Consider an edge $(v_{j+}, v'_{j+}) \in E'$ (edge (v_{j-}, v'_{j-}) can be argued similarly.) If we do not select the task $i(j, -)$ then the other three tasks using it do not use more than the available capacity on (v_{j+}, v'_{j+}). If we select the task $i(j, -)$ then we do not select any task $i(C, j)$ using (v_{j+}, v'_{j+}). The reason is that if we selected such a task then x_j would satisfy the clause C. But if $i(C, j)$ used (v_{j+}, v'_{j+}) then this would mean that x_j appeared positively in C. However, since we selected the task $i(j, -)$ this means that the satisfying assignment set x_j to false which contradicts that x_j satisfies C. $\qquad\square$

We prove that if any variable assignment can satisfy at most m' clauses then the optimal solution to our instance of UFP-tree has a value of at most $4n + m'$. First, we need the following lemma about the structure of an optimal solution.

Lemma 13 *For each x_j, any optimal solution selects either $i(j, +)$ or $i(j, -)$.*

Proof Assume by contradiction that there is an optimal solution S such that for a variable x_j neither $i(j, +)$ nor $i(j, -)$ are selected. Due to the way we defined the capacities on P and the task demands, the solution S can contain at most one task in $\{i(j', +), i(j', -)\}$ for each variable $x_{j'}$ and at most one task in $\{i(C, j), i(C, k), i(C, \ell)\}$ for each clause C containing variables x_j, x_k, x_ℓ. Moreover, if we select at most one task in $\{i(j', +), i(j', -)\}$ for each variable $x_{j'}$ and at most one task in $\{i(C, j), i(C, k), i(C, \ell)\}$ for each clause C containing variables x_j, x_k, x_ℓ then we do not violate the capacities on P. Thus, if we add $i(j, +)$ to S then on P we will not violate the capacity of any edge. We add $i(j, +)$ to S and remove all other tasks that use (v_{j-}, v'_{j-}). The task $i(j, +)$ yields a profit of 4 and the other three tasks using (v_{j-}, v'_{j-}) have a total profit of at most 3. Thus, we obtain a solution that is more valuable than S, a contradiction to the choice of S being optimal. $\qquad\square$

Lemma 14 *If any assignment can satisfy at most m' clauses, then the optimal solution to our instance of* **UFP-tree** *has a value of at most* $4n + m'$.

Proof We prove a contrapositive statement. Suppose that we have a solution S with $w(S) > 4n + m'$. By Lemma 13 for each variable x_j either $i(j, +) \in \mathcal{T}'$ or $i(j, -) \in \mathcal{T}'$. These tasks contribute a profit of $4n$ in $w(S)$, so a profit of more than m' must come from the tasks $i(C, j)$. We construct an assignment of the variables of ϕ by setting each variable x_j to true if and only if $i(j, +) \in S$. We claim that for each task $i(C, j) \in S$ the variable x_j satisfies C, implying that more than m' clauses are satisfied and thus completing the proof.

Assume that $i(C, j) \in S$ uses the edge (v_{j+}, v'_{j+}) (the other case that $i(C, j) \in S$ uses the edge (v_{j-}, v'_{j-}) can be proven similarly). This implies that x_j appears positively in C. Also, we know that the task $i(j, -)$ uses the full capacity of (v_{j+}, v'_{j+}). Thus, $i(j, -) \notin S$ and therefore we set x_j to be true in our variable assignment. Thus, C is satisfied by our assignment. □

References

1. Adamaszek, A., Wiese, A.: Approximation schemes for maximum weight independent set of rectangles. In: 54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, pp. 400–409. IEEE Computer Society (2013)
2. Anagnostopoulos, A., Grandoni, F., Leonardi, S., Wiese, A.: Amazing 2+ϵ approximation for unsplittable flow on a path. In: Chekuri, S. (Ed.) Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, pp. 26–41. SIAM (2014)
3. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and the hardness of approximation problems. J. ACM **45**(3), 501–555 (1998)
4. Bansal, N., Chakrabarti, A., Epstein, A., Schieber, B.: A quasi-PTAS for unsplittable flow on line graphs. In: Kleinberg, J,M. (Ed.), Proceedings of the 38th Annual ACM Symposium on Theory of Computing, 2006, pp. 721–729. ACM (2006)
5. Batra, J., Garg, N., Kumar, A., Mömke, T., Wiese, A.: New approximation schemes for unsplittable flow on a path. In: Indyk, P. (Ed.) Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, pp. 47–58. SIAM (2015)
6. Batra, J., Garg, N., Kumar, A., Mömke, T., Wiese, A.: New approximation schemes for unsplittable flow on a path. In: Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 47–58. SIAM (2015)
7. Bonsma, P.S., Schulz, J., Wiese, A.: A constant-factor approximation algorithm for unsplittable flow on paths. SIAM J. Comput. **43**(2), 767–799 (2014)
8. Boykov, Y., Jolly, M.P.: Interactive graph cuts for optimal boundary and region segmentation of objects in N-D images. In ICCV, pp. 105–112 (2001)
9. Călinescu, G., Chakrabarti, A., Karloff, H.J., Rabani, Y.: An improved approximation algorithm for resource allocation. ACM Trans. Algorithms **7**(4), 48 (2011)
10. Chakaravarthy, V.T., Choudhury, A.R., Gupta, S., Roy, S., Sabharwal, Y.: Improved algorithms for resource allocation under varying capacity. In Schulz, A.S., Wagner, D., (Eds.) Algorithms—ESA 2014—22th Annual European Symposium, Proceedings, volume 8737 of Lecture Notes in Computer Science, pp. 222–234. Springer (2014)

11. Chakrabarti, A., Chekuri, C., Gupta, A., Kumar, A.: Approximation algorithms for the unsplittable flow problem. Algorithmica **47**(1), 53–78 (2007)
12. Chalermsook, P., Chuzhoy, J.: Maximum independent set of rectangles. In: Mathieu, C. (Ed.) Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, pp. 892–901. SIAM (2009)
13. Chan, T.M., Sariel, H.P.: Approximation algorithms for maximum independent set of pseudo-disks. Discret. Comput. Geom. **48**(2), 373–392 (2012)
14. Chekuri, C., Ene, A., Korula, N.: Unsplittable flow in paths and trees and column-restricted packing integer programs. In: Dinur, I., Jansen, K., Naor, Rolim, J.D.P. (Eds.) Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 12th International Workshop, APPROX 2009, and 13th International Workshop, RANDOM 2009, Proceedings, volume 5687 of Lecture Notes in Computer Science, pp. 42–55. Springer (2009)
15. Chekuri, C., Vondrák, J., Zenklusen, R.: Submodular function maximization via the multilinear relaxation and contention resolution schemes. SIAM J. Comput. **43**(6), 1831–1879 (2014)
16. Dinur, I.: The PCP theorem by gap amplification. J. ACM **54**(3), 12 (2007)
17. Dueck, D., Frey, B.J.: cNon-metric affinity propagation for unsupervised image categorization. In: IEEE 11th International Conference on Computer Vision, ICCV 2007, pp. 1–8. IEEE (2007)
18. Friggstad, Z, Gao, Z: On linear programming relaxations for unsplittable flow in trees. In: Garg, N., Jansen, K., Rao A., Rolim, J.D.P. (Eds.) Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, volume 40 of LIPIcs, pp. 265–283. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2015)
19. Grandoni, F., Ingala, S., Uniyal, S.: Improved approximation algorithms for unsplittable flow on a path with time windows. In: Sanita, L., Skutella, M. (Eds.) Approximation and Online Algorithms—13th International Workshop, WAOA 2015, Patras, Greece, September 17–18, 2015. Revised Selected Papers
20. Jegelka, S., Bilmes. J.A.: Submodularity beyond submodular energies: coupling edges in graph cuts. In: The 24th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2011, pp. 1897–1904. IEEE Computer Society (2011)
21. Kempe, D., Kleinberg, J.M., Tardos, E.: Maximizing the spread of influence through a social network. Theory Comput. **11**, 105–147 (2015)
22. Korach, E., Solel, N.: Tree-width, path-width, and cutwidth. Discret. Appl. Math. **43**(1), 97–101 (1993)
23. Krause, A., Guestrin, C.: Near-optimal observation selection using submodular functions. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, 2007, pp. 1650–1654. AAAI Press (2007)
24. Krause, A., Guestrin, C.: Submodularity and its applications in optimized information gathering. ACM TIST **2**(4), 32 (2011)
25. Krause, A., Singh, A.P., Guestrin, C.: Near-optimal sensor placements in gaussian processes: theory, efficient algorithms and empirical studies. J. Mach. Learn. Res. **9**, 235–284 (2008)
26. Lin, H., Bilmes, J.A: A class of submodular functions for document summarization. In: Lin, D., Matsumoto, Y., Mihalcea, R. (Eds.) The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 2011, pp. 510–520. The Association for Computer Linguistics (2011)
27. Nemhauser, G.L., Wolsey, L.A., Fisher, M.L.: An analysis of approximations for maximizing submodular set functions-I. Math. Program. **14**(1), 265–294 (1978)
28. Papadimitriou, C.H., Yannakakis, M.: Optimization, approximation, and complexity classes. J. Comput. Syst. Sci. **43**(3), 425–440 (1991)