To Augment or Not to Augment: Solving Unsplittable Flow on a Path by Creating Slack

Fabrizio Grandoni* Tobias Mömke† Andreas Wiese ‡ Hang Zhou§

Abstract

In the Unsplittable Flow on a Path problem (UFP) we are given a path with non-negative edge capacities and a set of tasks, each one characterized by a subpath, a demand, and a profit. Our goal is to select a subset of tasks of maximum total profit so that the total demand of the selected tasks on each edge does not exceed the respective edge capacity. UFP naturally captures several applications in bandwidth allocation, job scheduling, and caching.

Following a sequence of improvements, the current best (polynomial time) approximation factor for UFP is $2+\varepsilon$ [Anagnostopoulos et al. SODA'14]. UFP also admits a QPTAS [Bansal et al. STOC'06, Batra et al. SODA'15], and finding a PTAS is considered a challenging open problem.

In this paper we make progress in the direction of the mentioned open problem. Informally, we introduce a technique to obtain real PTASs from PTASs with *resource augmentation* where edge capacities can be violated by a $1+\varepsilon$ factor. While unfortunately we do not have a resource-augmentation PTAS for the general case of UFP, for many relevant special cases we have such an algorithm or we provide one in this paper. For example, our approach leads to a PTAS for the rooted case of UFP, where all tasks share a common edge. This is one of the simplest natural restrictions of UFP where the best-known approximation was $2+\varepsilon$ (like for the general case).

At a high level, our technique is to sacrifice a few tasks in the optimal solution (with a small loss of profit) in order to create a sufficient amount of *slack capacity* on each edge. This slack turns out to be large enough to substitute the additional capacity we would gain from resource augmentation. Crucial for our approach is that we obtain slack from tasks with relatively small and relatively large demand simultaneously. In all prior polynomial time approximation algorithms the sacrificed tasks came from only one of these two groups.

*IDSIA, University of Lugano, Switzerland. fabrizio@idsia.ch.

†Saarland University, Saarland Informatics Campus, Germany. moemke@cs.uni-saarland.de. Funded by Deutsche Forschungsgemeinschaft grant MO 2889/1-1.

‡University of Chile, Santiago de Chile, Chile. This research was carried out while the author was at the Max Planck Institute for Informatics, Saarland Informatics Campus, Germany. awiese@mpi-inf.mpg.de.

§Max Planck Institute for Informatics, Saarland Informatics Campus, Germany. hzhou@mpi-inf.mpg.de. Research supported in part by the Lise Meitner Award Fellowship.

1 Introduction

In the *Unsplittable Flow on a Path* problem (UFP) we are given an undirected path $G=(V,E)$, with edge capacities $u(e)\in\mathbb{N}$ for $e\in E$. Furthermore, we are given a set T of n tasks. Each task $i\in T$ is specified by a subpath $P(i)$ between the start (i. e., leftmost) vertex $s(i)\in V$ and the end (i. e., rightmost) vertex $t(i)\in V$, a demand $d(i)\in\mathbb{N}$, and a profit (or weight) $w(i)\geq 0$. Our goal is to select a subset $T'\subseteq T$ of tasks of maximum total profit such that for each edge e the total demand of the selected tasks using e does not exceed $u(e)$.

UFP and its variations are motivated by several applications in bandwidth allocation [8, 17, 24], caching [18], scheduling [4], multi-commodity demand flow [16], and resource allocation [7, 11, 19, 25]. For example, one can interpret edges as time-slots, and their capacity as the amount of a given resource whose supply varies over time. Tasks can be interpreted as jobs that have to be executed in a given time interval, and that need a fixed amount of the considered resource. Note that the problem is identical to Knapsack if G consists of a single edge only.

UFP attracted a lot of attention in the last few years in the approximation algorithms community. The problem admits a QPTAS [5, 9] which (in some non-trivial sense) builds up on a QPTAS for the *rooted* case of the problem, where all tasks share a common edge. The existence of a QPTAS gives some evidence that UFP might indeed admit a PTAS: finding it is considered as a challenging open problem in the area.

In terms of polynomial-time approximation, the first non-trivial result was an $O(\log n)$-approximation by Bansal et al. [6]. To achieve that, Bansal et al. reduced the general case to the rooted case of the problem and provided an $O(1)$-approximation for the latter. After

an improvement to $(7+\varepsilon)$-approximation by Bonsma et al. [10], the current best approximation ratio is $2+\varepsilon$ due to Anagnostopoulos et al. [3].

Traditionally, tasks are classified into relatively large and relatively small. A task i is δ-*large* if there is an edge $e \in P(i)$ such that $d(i) \geq \delta \cdot u(e)$ and δ-*small* otherwise. All previous polynomial time approximation algorithms for the problem and its special cases [3, 10, 11, 13, 16] essentially treat these two groups separately, which inherently lose a factor of 2 in the approximation ratio. It is challenging to handle the two groups in a combined way by a polynomial time algorithm. This can be seen from the fact that the ratio of $2+\varepsilon$ is the best known polynomial time result even for the special case of uniform edge capacities [7, 11, 25] and for the mentioned rooted case. No better result is known even under resource augmentation, i. e., if we are allowed to increase the capacity of all edges by a factor $1+\varepsilon$ and the compared optimal solution does not have this privilege.

1.1 Our Results and Techniques In most prior work on UFP the tasks are classified into large and small tasks as defined above, based on the input alone. The *sparsification lemma* by Batra et al. [9, Lemma 3.1] allows us to deviate from this path. Let OPT be the optimal solution. For each edge e, consider the $1/\varepsilon$ tasks in OPT with largest demands using e. We call those tasks *locally large*. Note that this definition depends on OPT. The sparsification lemma states that for any $\kappa \in \mathbb{N}$ one can *globally* remove a set of tasks with total weight $O(\kappa\varepsilon) \cdot w(\mathrm{OPT})$ such that *for each edge* e at least κ of its locally large tasks are removed. Then, intuitively, in a dynamic programming approach the remaining locally large tasks can be guessed and the gained slack can be used to simplify the computation for the locally small tasks.

In this paper we push this approach further. By using the sparsification lemma we remove $\kappa = 2/\varepsilon^4$ out of the $1/\varepsilon^5$ largest tasks on each edge e. Then, each locally small task on an edge has a *very* small demand compared to the gained slack. If a task is locally small on all of its used edges we call it *shrinkable*. Using LP-based arguments (critically using that such tasks are locally *very* small) we reduce the total demand of the shrinkable tasks on each edge by a factor $1-\varepsilon$ while losing only a factor $1+O(\varepsilon)$ in the total profit. Hence, we sacrifice on each edge e some fraction of both the locally large and locally small tasks. This construction is described in Section 2.

From the point of view of the shrinkable tasks, the available capacity on each edge is by a factor $1+\Theta(\varepsilon)$ larger than the actually needed capacity. In some informal sense, we use this to achieve the following goal. We start with an approximation algorithm in the *resource augmentation* setting, where the computed solution is allowed to violate edge capacities by a factor $1+\varepsilon$ (while comparing the profit with the optimal solution that cannot do this). Then we transform it to an algorithm that up to a factor $1+O(\varepsilon)$ achieves the same approximation ratio *without resource augmentation*.

Typically, resource augmentation helps substantially when designing approximation algorithms. Unfortunately, a resource augmentation PTAS for the general case of UFP is not known. Still, we believe that this is a much easier task than achieving (directly) a PTAS without resource augmentation for the same problem. For instance, the additional capacity can be used to round and discretize the demands of tasks, or to allow errors when estimating the required capacities for subsets of tasks (a common step for small tasks of a given instance). Meanwhile, we achieve PTASs for the following special cases and variants of UFP:

- We design a resource augmentation PTAS for the rooted case of UFP, and from there we derive a PTAS for the setting without resource augmentation. We recall that here all task subpaths share a common edge, and the previous best approximation algorithm for this case was the $(2+\varepsilon)$-approximation for the general case. As an extension to our approach, we also obtain a PTAS for the case that all tasks use at least one out of a subset of $O(1)$ edges. In turn, this PTAS can be used to obtain a PTAS for the *non-containment* case of the problem, where no two tasks i, j satisfy $P(i) \subseteq P(j)$. See Section 3.

- We consider an unlimited-supply special case of UFP (unl-UFP), where we are allowed to include multiple copies of the same task in the solution.[1] This is motivated by scenarios where each task models a type of client, and there are many clients of each type (enough to saturate the capacity of some edge). To the best of our knowledge, this unlimited-supply notion was not addressed before in the framework of UFP, while it is common in some related pricing problems [23]. We first design a resource augmentation PTAS for unl-UFP via a structural lemma that shows that there are near-optimal solutions in which the small tasks are cho-

[1] This problem can be reduced in polynomial time to UFP via a binary encoding trick: for a task i that can appear in at most n_i copies in a feasible solution, create tasks i_k, $k = 0, \ldots, \lfloor \log_2 n_i \rfloor$, with $P(i_k) = P(i)$, $d(i_k) = 2^k d(i)$ and $w(i_k) = 2^k w(i)$. This reduction takes polynomial time. It is easy to see that any feasible solution to one problem can be mapped to an equal profit solution of the other problem and vice versa.

sen in *bundles*. Using our technique above, we then obtain a PTAS without resource augmentation.

- We obtain a PTAS for the special case of UFP where the profit of each task i is proportional to its *area* $d(i) \cdot |P(i)|$ (area-UFP). This special case is also well-motivated: for example, considering $P(i)$ as a time interval as mentioned before, $d(i) \cdot |P(i)|$ can be interpreted as the total volume of the considered resource used to process task i (hence it makes sense to charge i by that amount). In this case the PTAS with resource augmentation is already quite involved and requires some careful charging arguments. We obtain a PTAS without resource augmentation using the same technique.

Due to space constraints, we refer to the full version of this paper for the last two results (i.e., the PTASs for unl-UFP and for the case where the profit of each task is proportional to its area).

1.2 Other Related Work UFP is strongly NP-hard, even in the case of uniform edge capacities and unit profits [10, 18, 19]. If all tasks are sufficiently small, Chekuri et al. [16] proved that one can achieve a $(1+\varepsilon)$-approximation via LP-rounding. This generalizes (and improves) previous results for the cases of uniform edge capacities [11] and the no-bottleneck-assumption (NBA) [13] (the NBA requires that $\max_{i \in T} d(i) \leq \min_{e \in E} u(e)$). Since relatively large tasks are easy to handle, $O(1)$- and $(2+\varepsilon)$-approximation algorithms were known for these two cases [11, 13, 16] before they were known for the general case [3, 10].

An interesting question related to UFP is finding LP relaxations with small integrality gap. The standard LP has an integrality gap of $\Omega(n)$ [13]. By adding extra constraints, Chekuri et al. [15] reduced the gap to $O(\log^2 n)$ (later improved to $O(\log n)$ by the same authors [14]). Anagnostopoulos et al. [2] described a compact LP for the cardinality case of UFP with constant integrality gap. This directly implied an $O(\log n)$ gap for the weighted case, matching [14]. They also presented an extended (i. e., with extra variables besides the task decision variables) compact LP relaxation for weighted UFP with constant integrality gap. The current best integrality gap for an LP-relaxation of the problem is $O(\log n / \log \log n)$ [22]. Finding a compact non-extended LP relaxation with $O(1)$ integrality gap remains an interesting open problem.

Another interesting generalization of UFP, called *bag UFP*, is obtained by grouping tasks into bags, and requiring that at most one task per bag is selected. Chakaravarthy et al. [12] gave an $O(\log n)$-approximation for the problem. This was improved by Grandoni et al. [22] to $O(\log n / \log \log n)$ in the weighted version, and to $O(1)$ in the cardinality version, via LP-relaxations with the same integrality gaps.

The *Unsplittable Flow on a Tree* problem (UFT) is the natural generalization of UFP where the input graph is a tree rather than a path. This problem is APX-hard, even for unit demands, edge capacities being either 1 or 2, and trees with depth 3, see [21]. UFT admits an $O(\log^2 n)$-approximation [15], and a 48-approximation under the NBA [16]. Recently, an $O(k \cdot \log n)$-approximation was presented in [1], where k denotes the path-width of the tree (which is bounded by $O(\log n)$ for any tree). The results holds even for submodular objective functions. The Unsplittable Flow problem has also been studied on more general graph classes, see for instance [9] for a survey.

1.3 Preliminaries and Notation In a (unl-)UFP instance, for each edge $e \in E$, let $T_e \subseteq T$ be the (multi-)subset of tasks i using edge e, i. e., with $e \in P(i)$.[2] For every (multi-)set of tasks T', we define $w(T') := \sum_{i \in T'} w(i)$ and $d(T') := \sum_{i \in T'} d(i)$. The goal of (unl-)UFP is to select a (multi-)set of tasks T' with maximum profit $w(T')$ such that $d(T' \cap T_e) \leq u(e)$ for each edge e. In particular, in the unl-UFP model, we are allowed to select each task from T arbitrarily often. For an instance (G, T) of UFP or unl-UFP, OPT denotes an optimal solution and we define $\text{opt} = w(\text{OPT})$ to be the weight of OPT. Without loss of generality, we may assume that $|V| = 2n$ and that each vertex is either the start-vertex or the end-vertex of exactly one input task.

Throughout this paper, we use the notation $O_\varepsilon(f(n))$ for functions that are in $O(f(n))$ if ε is a constant.

We will crucially use the following *sparsification lemma* [9, Lemma 3.1].

LEMMA 1.1. (SPARSIFICATION LEMMA [9]) *Let $\varepsilon' > 0$ and $\kappa \in \mathbb{N}$ be constants. Given a UFP instance with optimal solution* OPT*, there exists a feasible solution* OPT' *with profit* $\text{opt}' \geq (1 - O(\varepsilon')\kappa) \cdot \text{opt}$ *such that for each edge e there exists a value $\delta'_e \geq 0$ satisfying the following conditions:*

- $d(T_e \cap \text{OPT}') \leq u(e) - \kappa \cdot \delta'_e$;
- *there are at most $1/\varepsilon'$ tasks $i \in T_e \cap \text{OPT}'$ such that $d(i) \geq \delta'_e$.*

2 Creating More Slack

In this section we present an approach to sacrifice both large and small tasks from the optimal solution to create

[2] We sometimes slightly abuse notation by identifying a subpath with the corresponding subset of edges.

some slack capacity on each edge. Compared to prior work, we generate a larger amount of slack, while we still lose only an $O(\varepsilon)$-fraction of the optimal profit. The main result of this section is Lemma 2.1.

LEMMA 2.1. *Let $\varepsilon > 0$ be a constant. For a UFP instance with an optimal solution* OPT*, there exists a feasible solution* OPT'' *with profit at least* $(1-O(\varepsilon))\cdot\mathrm{opt}$ *such that for each edge e there is a value $\delta_e \geq 0$ satisfying the following conditions:*

1. *either $\delta_e = (1/\varepsilon^2)^j$ for some integer $j \geq 0$, or $\delta_e = 0$;*
2. *$d(T_e \cap \mathrm{OPT}'') \leq u(e) - \delta_e$;*
3. *there are at most $1/\varepsilon^5$ tasks $i \in T_e \cap \mathrm{OPT}''$ such that $d(i) \geq \varepsilon^2 \cdot \delta_e$;*
4. *the total demand of all tasks $i \in T_e \cap \mathrm{OPT}''$ such that $d(i) < \varepsilon^2 \cdot \delta_e$ is at most $5\delta_e/\varepsilon^3$.*

In the above lemma OPT'' represents a near-optimal solution, and δ_e is the slack that we create on each edge. Points (2) and (3) above are similar in spirit to the sparsification lemma (Lemma 1.1). Point (1) is introduced only to simplify the presentation (though this is not essential to get the main qualitative result). Point (4) is what makes our approach really different: it states that the total demand of locally small tasks is not much larger than the slack δ_e.

In the remainder of this section we prove Lemma 2.1. We start with the optimal solution OPT and remove some of its tasks to generate some slack. First we apply Lemma 1.1 with $\varepsilon' = \varepsilon^5$ and $\kappa = 2/\varepsilon^4$ and we obtain a feasible solution OPT' with $\mathrm{opt}' \geq (1 - O(\varepsilon)) \cdot \mathrm{opt}$ such that for each edge e there is a value $\delta'_e \geq 0$ satisfying the following conditions: (1) $d(T_e \cap \mathrm{OPT}') \leq u(e) - 2\delta'_e/\varepsilon^4$; (2) there are at most $1/\varepsilon^5$ tasks $i \in T_e \cap \mathrm{OPT}'$ such that $d(i) \geq \delta'_e$.

At each edge e, we define $\hat{\delta}_e$ by rounding δ'_e to the next higher power of $(1/\varepsilon^2)$: if $\delta'_e > 0$, then define $\hat{\delta}_e := (1/\varepsilon^2)^j$ where $j \in \mathbb{N}$ is such that $\delta'_e \in ((1/\varepsilon^2)^{j-1}, (1/\varepsilon^2)^j]$; and if $\delta'_e = 0$, then define $\hat{\delta}_e := 0$. We observe that $\delta'_e \leq \hat{\delta}_e \leq \delta'_e/\varepsilon^2$. Therefore, we have the following properties on $\hat{\delta}_e$.

PROPOSITION 2.1. *For every edge e, we have:*

- *$d(T_e \cap \mathrm{OPT}') \leq u(e) - 2\hat{\delta}_e/\varepsilon^2$;*
- *there are at most $1/\varepsilon^5$ tasks $i \in T_e \cap \mathrm{OPT}'$ such that $d(i) \geq \hat{\delta}_e$.*

In a second step, we generate even more slack. To achieve that, we classify the tasks into *shrinkable* and *non-shrinkable*. This definition is similar in spirit to the usual classification into small and large tasks. However, it is not exactly the same and in particular it is based on the values $\{\hat{\delta}_e\}_e$. We say a task $i \in \mathrm{OPT}'$ is *shrinkable* if for every edge $e \in P(i)$, we have $d(i) < \hat{\delta}_e$; otherwise we say that it is *non-shrinkable*. Denote by OPT'_S and OPT'_{NS} the shrinkable tasks and the non-shrinkable tasks in OPT', respectively.

Intuitively, we want to shrink the shrinkable tasks by a factor $1 - \varepsilon$. The gained slack on each edge will then be proportional to the total demand of the shrinkable tasks using this edge (an ε-fraction of the latter). However, we cannot simply reduce the demand of each shrinkable task by a factor $1 - \varepsilon$. Instead, we construct an integral solution OPT''_S by removing some shrinkable tasks from OPT'_S in such a way that we lose only a factor $1 - O(\varepsilon)$ in the profit while still gaining the desired amount of slack. Key here is that each shrinkable task i uses less than an ε^2-fraction of the available capacity for the shrinkable tasks (i. e., the capacity not used by the non-shrinkable tasks) on each edge $e \in P(i)$. Thus, each shrinkable task is ε^2-small. For such tasks it is known that the canonical LP has an integrality gap of $1 - O(\varepsilon)$, see [16]. In our analysis, we first construct a *fractional* solution for the shrinkable tasks in which each shrinkable task is reduced by a factor $1 - \varepsilon$, and then round this solution to an integral one.

LEMMA 2.2. *There is a set $\mathrm{OPT}''_S \subseteq \mathrm{OPT}'_S$ with profit at least $(1 - O(\varepsilon)) \cdot \mathrm{opt}'_S$ such that on each edge e we have $d(T_e \cap \mathrm{OPT}''_S) \leq (1 - \varepsilon) \cdot d(T_e \cap \mathrm{OPT}'_S) + \hat{\delta}_e/\varepsilon^2$.*

Proof. In order to construct OPT''_S, we consider the following linear program, where $\tilde{u}(e)$ denotes $(1 - \varepsilon) \cdot d(T_e \cap \mathrm{OPT}'_S) + \hat{\delta}_e/\varepsilon^2$.

$$\begin{aligned} \max \sum_{i \in \mathrm{OPT}'_S} w(i) \cdot x_i & \\ \text{s.t.} \sum_{i \in T_e \cap \mathrm{OPT}'_S} d(i) \cdot x_i \leq \tilde{u}(e) & \qquad \forall e \in E \\ 0 \leq x_i \leq 1 & \qquad \forall i \in \mathrm{OPT}'_S \end{aligned}$$

Obviously, setting every x_i to $1 - \varepsilon$ gives a feasible fractional solution of profit $(1 - \varepsilon) \cdot \mathrm{opt}'_S$. We note that for every task $i \in \mathrm{OPT}'_S$ and for every edge $e \in P(i)$, we have $d(i) < \hat{\delta}_e \leq \varepsilon^2 \cdot \tilde{u}(e)$. Thus a result in [16, Corollary 3.4] shows that the integrality gap of the above LP is at least $1 - O(\varepsilon)$. Therefore, there is an integral solution OPT''_S to the linear program with weight at least $1 - O(\varepsilon)$ times the weight of the fractional solution above which implies that $\mathrm{opt}''_S \geq (1 - O(\varepsilon)) \cdot \mathrm{opt}'_S$. □

At each edge e, we define the final slack δ_e with respect to the term $z_e := \hat{\delta}_e/\varepsilon^2 + \varepsilon \cdot d(T_e \cap \mathrm{OPT}'_S)$: If $z_e > 0$, then define $\delta_e := (1/\varepsilon^2)^j$ where $j \in \mathbb{N}$ is such that $z_e \in [(1/\varepsilon^2)^j, (1/\varepsilon^2)^{j+1})$; and if $z_e = 0$, then define $\delta_e := 0$. Noting that $\hat{\delta}_e$ is either 0 or a power of $1/\varepsilon^2$, we have:

FACT 2.1. $\delta_e \geq \hat{\delta}_e/\varepsilon^2$.

The following lemma bounds the total demand of the small tasks on each edge. Intuitively, these tasks are roughly the shrinkable tasks. Since each unit of a shrinkable task contributes ε unit to the slack, the total demand of these tasks is roughly δ_e/ε (up to a constant factor).

LEMMA 2.3. *For every edge $e \in E$, the total demand of all tasks i in $T_e \cap \mathrm{OPT}'$ such that $d(i) < \varepsilon^2 \cdot \delta_e$ is at most $5\delta_e/\varepsilon^3$.*

Proof. We classify the tasks i in $T_e \cap \mathrm{OPT}'$ such that $d(i) < \varepsilon^2 \cdot \delta_e$ into three kinds: (1) The tasks i such that $d(i) < \hat{\delta}_{e'}$ for all $e' \in P(i)$; (2) The tasks i such that $d(i) \geq \hat{\delta}_e$; (3) The tasks i such that $d(i) < \hat{\delta}_e$ and $d(i) \geq \hat{\delta}_{e'}$ for some $e' \in P(i)$ that is different from e.

The tasks of kind (1) are the shrinkable tasks. From the definition of δ_e, their total demand $d(T_e \cap \mathrm{OPT}'_S)$ is at most $z_e/\varepsilon \leq \delta_e/\varepsilon^3$.

The number of tasks of kind (2) is at most $1/\varepsilon^5$ by Proposition 2.1. Each of them has demand at most $\varepsilon^2 \cdot \delta_e$ from the lemma assumption. Thus their total demand is at most δ_e/ε^3.

To analyze the tasks of kind (3), we construct a sequence $\{e_k\}_{k \geq 0}$ of edges to the right (resp. left) of e: define e_0 as e; for every $k \geq 1$, define e_k as the leftmost edge to the right of e_{k-1} (resp. the rightmost edge to the left of e_{k-1}) such that $\hat{\delta}_{e_k} < \hat{\delta}_{e_{k-1}}$. Consider a task i of kind (3). In one of the two sequences above, there exists some $k \geq 1$ such that $\hat{\delta}_{e_k} \leq d(i) < \hat{\delta}_{e_{k-1}}$. We then associate task i to the edge e_k. For every edge e_k ($k \geq 1$) in either sequence, there are at most $1/\varepsilon^5$ tasks associated to e_k by Proposition 2.1. Since every such task has demand at most $\hat{\delta}_{e_{k-1}}$, their total demand is at most $\hat{\delta}_{e_{k-1}}/\varepsilon^5$. Summing over all edges $e_k (k \geq 1)$ from both sequences, and noting that $\hat{\delta}_{e_k}$ decreases exponentially in k and using Fact 2.1, we then have the total demand of the tasks of kind (3) is at most

$$2 \cdot \sum_{k \geq 1} \hat{\delta}_{e_{k-1}}/\varepsilon^5 \leq 3\hat{\delta}_e/\varepsilon^5 \leq 3\delta_e/\varepsilon^3.$$

Summing the demand over the three kinds of tasks leads to the statement. □

Finally, we define $\mathrm{OPT}'' := \mathrm{OPT}''_S \cup \mathrm{OPT}'_{NS}$. We note that $\mathrm{OPT}'' \subseteq \mathrm{OPT}'$.

Proof (sketch) of Lemma 2.1. Point (1) comes directly from the definition of δ_e. Point (2) combines Proposition 2.1, Lemma 2.2, and the definition of δ_e. Point (3) follows from Proposition 2.1 and Fact 2.1. Point (4) follows from Lemma 2.3. □

3 A PTAS for Rooted UFP

In this section we present a PTAS for rooted UFP. We start by presenting a PTAS under resource augmentation for the problem, where we are allowed to violate the edge capacities by a factor $1 + \mu$ for some $\mu > 0$. Then we turn it into a standard PTAS by means of our approach. Finally, we discuss some useful consequences of our algorithm, such as a PTAS for non-containment UFP.

Recall that in rooted UFP all tasks share a common edge. We denote it by e_M. Without loss of generality, we assume that the edge capacities are non-decreasing from the leftmost edge to e_M and non-increasing from e_M to the rightmost edge.

3.1 With Resource Augmentation

One benefit of resource augmentation is that we can partition the given instance into a polynomial number of subinstances with a constant range of edge capacities, as in the following lemma.[3] Its proof is in Appendix A.

LEMMA 3.1. *Let $\varepsilon, \mu > 0$ be constants where $\mu \leq 1$. By losing an ε fraction of the profit and increasing the edge capacities by a factor $1+\mu$ we can reduce the given instance to a polynomial number of subinstances such that for each of them there is some parameter U such that $u(e) \in [(\mu/3)^{1/\varepsilon} \cdot U, U]$ for each edge e.*

We solve each of the new instances separately. Consider such an instance. Resource augmentation allows us to round the edge capacities to powers of $1 + \mu$ such that they form a step function. This function has a constant number of steps, using the properties of the rooted case.

FACT 3.1. *By increasing the edge capacities by a factor $1 + \mu$ we can assume that they attain only $(\frac{1}{\varepsilon\mu})^{O(1)}$ different values and that they remain non-decreasing on the left of e_M and non-increasing on the right of e_M.*

Now the problem becomes easy: as long as a solution is feasible on the $(\frac{1}{\varepsilon\mu})^{O(1)}$ edges where the profile decreases when moving from e_M to the right or to the left, it is globally feasible. Thus the remaining problem is equivalent to an instance of constant-dimensional

[3] Indeed, Lemma 3.1 holds also for the general UFP (under resource augmentation), not only for rooted UFP.

knapsack. For the latter, there is a PTAS [20]. Therefore, we have:

THEOREM 3.1. *Let $\varepsilon, \mu > 0$ be constants where $\mu \leq 1$. There is a polynomial time $(1+\varepsilon)$-approximation algorithm for rooted UFP under $1+\mu$ resource augmentation.*

3.2 Without Resource Augmentation We study now the case without resource augmentation. The following lemma is a strengthening of Lemma 2.1 where we crucially use the extra properties of the rooted case.

LEMMA 3.2. *Let $\varepsilon > 0$ be a constant. For a rooted UFP instance with an optimal solution* OPT*, there exists a feasible solution* OPT'' *with slack $\{\delta_e\}_{e \in E}$ satisfying all the conditions of Lemma 2.1 plus the following condition: for any two edges e and e' with e to the left of e' and e' to the left of e_M, we have $\delta_e \leq \delta_{e'}$. A symmetric claim holds for the edges to the right of e_M.*

Proof. Consider the same construction as in Lemma 2.1 and with the same notation. To show the additional claim, let us first give some intuition about the proof of the sparsification lemma (Lemma 1.1) in [9]: for each edge e, there are κ tasks out of the $1/\varepsilon'$ tasks with largest demand in $\mathrm{OPT} \cap T_e$ that are removed, and the value δ'_e is the smallest demand among those removed tasks. Consider two edges e and e' with e to the left of e' and e' to the left of e_M. Since in our setting all tasks use e_M we have $\delta'_e \leq \delta'_{e'}$. Thus $\hat{\delta}_e \leq \hat{\delta}_{e'}$. Also, it holds that $d(T_e \cap \mathrm{OPT}'_S) \leq d(T_{e'} \cap \mathrm{OPT}'_S)$. Therefore, $z_e \leq z_{e'}$ and thus $\delta_e \leq \delta_{e'}$. Similarly, we can show that for any two edges e and e' with e to the right of e' and e' to the right of e_M we have $\delta_e \leq \delta_{e'}$. □

Based on the near-optimal solution OPT'' and the slack $\{\delta_e\}_{e \in E}$ from Lemma 3.2, we define a recursive algorithm that computes a solution with weight at least $\mathrm{opt}''(1 - O(\varepsilon))$ and might use the *full* capacity of the edges. So compared to OPT'' we use a kind of resource augmentation. On a high level, our algorithm proceeds in phases and each phase corresponds to the tasks of a certain *type* (with respect to the slack amount at the edges, see Section 3.2.1). In each phase, we guess the *huge* tasks and the *tiny* tasks of that type in the near-optimal solution, see Section 3.2.2. However, this recursive algorithm might need exponential running time. We describe in Section 3.2.3 how to turn it into a polynomial time dynamic program. The key is that, when we proceed through the phases, due to the slack we can "forget" some of the tasks that we guessed previously. All the "forgotten" tasks will fit into the slack. This ensures that the computed solution is still feasible while the number of DP-cells is bounded by a polynomial.

3.2.1 Types and their Properties We define the *type* of an edge e with respect to the slack δ_e.

DEFINITION 3.1. (*type*) *For each edge $e \in E$, we define its* type $\mathrm{type}(e)$ *as follows: If $\delta_e = (1/\varepsilon^2)^j$ for some $j \in \mathbb{N}$, then define $\mathrm{type}(e) := j$; and if $\delta_e = 0$, then define $\mathrm{type}(e) := -1$. For every type j, the* slack of type j *is $\delta^{(j)} := (1/\varepsilon^2)^j$ if $j \in \mathbb{N}$ and $\delta^{(j)} := 0$ if $j = -1$. We say that a task $i \in T$ is of* type j *if $P(i)$ uses an edge of type j and no edge of type $j-1$ or smaller. Let $T^{(j)} \subseteq T$ denote all tasks of type j.*

DEFINITION 3.2. (*huge* AND *tiny*) *For a task $i \in T$ of type j, it is* huge *if $d(i) \geq \varepsilon^2 \cdot \delta^{(j)}$ and* tiny *otherwise.*

We partition the tiny tasks in OPT'' according to their types: for every type j, we define $\mathrm{OPT}''^{(j)}_T$ to be the tiny tasks in OPT'' that have type j. We classify the huge tasks in OPT'' slightly differently: for every type j, we define $\mathrm{OPT}''^{(j)}_H$ to be the huge tasks i in OPT'' that have type *at most* j and such that $d(i) \geq \varepsilon^2 \cdot \delta^{(j)}$. Intuitively, $\mathrm{OPT}''^{(j)}_H$ contains all huge tasks of type j *plus* some *very* huge tasks of type less than j. We note that $\mathrm{OPT}''^{(j)}_H$ may have some overlap with $\mathrm{OPT}''^{(\ell)}_H$ for $\ell < j$. The overlap later enables us to "forget" some sets $\mathrm{OPT}''^{(\ell)}_H$ in the dynamic program.

For simplicity of presentation assume that for each type j there is at least one edge of type j on the left (resp. on the right) of e_M. Let $e^{(j)}_L$ and $e^{(j)}_R$ denote the "innermost type-j edges," i. e., the rightmost edge e of type j on the left of e_M and the leftmost edge of type j on the right of e_M, respectively.

FACT 3.2. *A task has type at most j if and only if it uses $e^{(j)}_L$ or $e^{(j)}_R$.*

Using Fact 3.2 and the definition of $\mathrm{OPT}''^{(j)}_H$ and $\mathrm{OPT}''^{(j)}_T$ we have:

PROPOSITION 3.1. *For every type j we have:*

- $\mathrm{OPT}''^{(j)}_H$ *is the set of the tasks $i \in \left(T_{e^{(j)}_L} \cup T_{e^{(j)}_R}\right) \cap \mathrm{OPT}''$ such that $d(i) \geq \varepsilon^2 \cdot \delta^{(j)}$;*

From Proposition 3.1 and Lemma 3.2 we have:

PROPOSITION 3.2. *For every type j we have $|\mathrm{OPT}''^{(j)}_H| \leq 2/\varepsilon^5$ and $d(\mathrm{OPT}''^{(j)}_T) \leq 10\delta^{(j)}/\varepsilon^3$.*

3.2.2 Guessing Tasks According to Types In the initial phase $k = -1$, every task of this type is huge. We first guess the two edges $e^{(-1)}_L, e^{(-1)}_R$ and then we guess the set $\mathrm{OPT}''^{(-1)}_H$ of the tasks using $e^{(-1)}_L$ or $e^{(-1)}_R$. This

can be done in time $n^{O_\varepsilon(1)}$ since there are only $O_\varepsilon(1)$ such tasks by Proposition 3.2.

Now we describe what we do in a general phase $k \geq 0$. Our goal is to select tasks of type k whose profit and used capacities on each edge are almost the same as those of OPT''. First, we guess the two edges $e_L^{(k)}, e_R^{(k)}$ and the set of huge tasks $\mathrm{OPT}_H''^{(k)}$. Again this can be done in time $n^{O_\varepsilon(1)}$. To compute the tiny tasks $\mathrm{OPT}_T''^{(k)}$ approximately, we define the function $f^{(k)}\colon E \mapsto \mathbb{N}$ as $f^{(k)}(e) := d(T_e \cap \mathrm{OPT}_T''^{(k)})$ which describes how much capacity these tasks use on *all* edges in the instance. Ideally, we would like to guess $f^{(k)}$. This is not achievable in polynomial time since $f^{(k)}$ can have linearly many steps. Instead, we round each $f^{(k)}(e)$ to the next higher integral multiple of $\delta^{(k)}/4$ and thus obtain a *profile* $\bar{f}^{(k)}$ as follows:

$$\bar{f}^{(k)}(e) := \left\lceil \frac{f^{(k)}(e)}{(\delta^{(k)}/4)} \right\rceil \cdot (\delta^{(k)}/4), \qquad \text{for each } e \in E.$$

See Figure 1. On each edge e we want to allocate $\bar{f}^{(k)}(e)$ units of capacity for the tiny tasks of type k. Recall that on all edges e with $f^{(k)}(e) > 0$ (i. e., the edges used by tasks in $\mathrm{OPT}_T''^{(k)}$) we have at least $\delta^{(k)}$ units of slack available. This justifies that we round up $f^{(k)}(e)$. Now we summarize some properties of the profile $\bar{f}^{(k)}$.

LEMMA 3.3. *The profile $\bar{f}^{(k)}$ is a step function with $O(1/\varepsilon^3)$ steps that satisfies*

1. $\max_e \bar{f}^{(k)}(e) \leq 10\delta^{(k)}/\varepsilon^3$;
2. *$\bar{f}^{(k)}$ is non-decreasing (resp., non-increasing) between the leftmost (resp., rightmost) edge of E and e_M;*
3. *the image of $\bar{f}^{(k)}$ contains only integral multiples of $\delta^{(k)}/4$.*

Proof. The first property follows from Proposition 3.2; the second property comes from the fact that all tasks in $\mathrm{OPT}_T''^{(k)}$ use e_M; and the third property is by the definition of $\bar{f}^{(k)}$. From the three properties, we know the step function has $O(1/\varepsilon^3)$ steps. □

Note that for each type k the profile $\bar{f}^{(k)}$ uses slightly more capacity on each edge compared with the tasks of $\mathrm{OPT}_T''^{(k)}$. However, using a geometric sum argument we show in the next lemma that on each edge e the edge capacity $u(e)$ is still satisfied when $\bar{f}^{(j)}(e)$ units of capacity is allocated to the tiny tasks of each type j. In fact, there is still a bit of the remaining slack.

LEMMA 3.4. *Let e be an edge of type k. Then $\sum_{j=-1}^{k} \bar{f}^{(j)}(e) + d\big(\bigcup_{j=-1}^{k} T_e \cap \mathrm{OPT}_H''^{(j)}\big) \leq u(e) - 2\delta_e/3$.*

Proof. For every type $-1 \leq j \leq k$, we have

$$\bar{f}^{(j)}(e) \leq f^{(j)}(e) + \delta^{(j)}/4 = d(T_e \cap \mathrm{OPT}_T''^{(j)}) + \delta^{(j)}/4.$$

Therefore, we have

$$\begin{aligned}
&\textstyle\sum_{j=-1}^{k} \bar{f}^{(j)}(e) + d\big(\bigcup_{j=-1}^{k} T_e \cap \mathrm{OPT}_H''^{(j)}\big) \\
=&\, d(T_e \cap \mathrm{OPT}'') + \textstyle\sum_{j=-1}^{k} \delta^{(j)}/4 \\
\leq&\, (u(e) - \delta_e) + \delta_e/3 \\
=&\, u(e) - 2\delta_e/3,
\end{aligned}$$

where the second last step follows from Lemma 3.2 and a geometric sum argument. □

It only remains to find a set of tiny tasks that fit into a guessed profile $\bar{f}_{\mathrm{ALG}}^{(k)}$. We observe that $T^{(k)}$ (see Definition 3.1) contains all tasks i that use $e_L^{(k)}$ or $e_R^{(k)}$ but neither $e_L^{(k-1)}$ nor $e_R^{(k-1)}$ (by Fact 3.2). Thus $T^{(k)}$ can inferred from our guesses for $e_L^{(k)}$, $e_R^{(k)}$, $e_L^{(k-1)}$, and $e_R^{(k-1)}$. The following lemma shows that there is an algorithm that fills the profile $\bar{f}_{\mathrm{ALG}}^{(k)}$ using tiny tasks from $T^{(k)}$ in a near-optimal way.

LEMMA 3.5. *Let $\bar{f}_{\mathrm{ALG}}^{(k)}$ be a profile satisfying the properties in Lemma 3.3. There is a polynomial time algorithm that computes a subset $\mathrm{ALG}_T^{(k)}$ of the tiny tasks in $T^{(k)}$ such that $d(T_e \cap \mathrm{ALG}_T^{(k)}) \leq \bar{f}_{\mathrm{ALG}}^{(k)}(e)$ for each edge e and that $w(\mathrm{ALG}_T^{(k)})$ is at least $(1 - O(\varepsilon))$-fraction of the maximum profit achievable.*

Proof. Let T' be the set of tiny tasks in $T^{(k)}$, i. e., $T' := \{i \in T^{(k)} \mid d(i) < \varepsilon^2 \cdot \delta^{(k)}\}$. We set up a linear program to formulate the problem of filling the profile $\bar{f}_{\mathrm{ALG}}^{(k)}$ using tasks from T'.

$$\begin{aligned}
\max \sum_{i \in T'} & w(i) \cdot x_i \\
\text{s. t.} \sum_{i \in T_e \cap T'} & d(i) \cdot x_i \leq \bar{f}_{\mathrm{ALG}}^{(k)}(e) && \forall e \in E \\
& 0 \leq x_i \leq 1 && \forall i \in T'
\end{aligned}$$

For every tasks $i \in T'$, we have $d(i) < \varepsilon^2 \cdot \delta^{(k)}$; and for every edge $e \in E$ such that $\bar{f}_{\mathrm{ALG}}^{(k)}(e) > 0$, we have $\bar{f}_{\mathrm{ALG}}^{(k)}(e) \geq \delta^{(k)}/4$ from Lemma 3.3. Therefore using a result in [16, Corollary 3.4] we can compute in polynomial time an integral solution $\mathrm{ALG}_T^{(k)}$ with weight at least $(1 - O(\varepsilon))$ times the objective value of the linear program. □

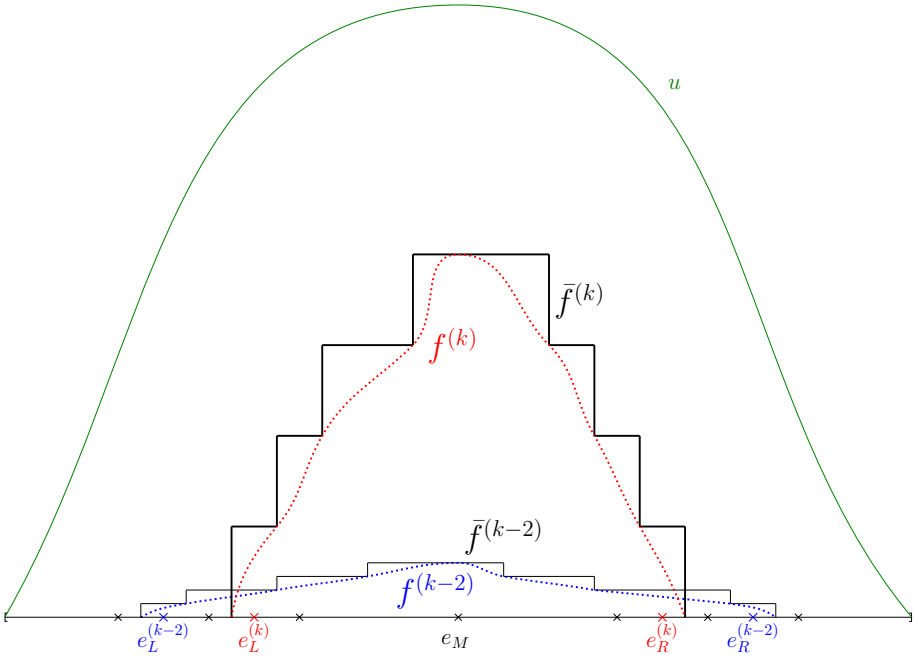

Figure 1: Profile $\bar{f}^{(k)}$ in phase k. The horizontal base line represents the entire path E and the outermost curve represents the capacity $u(e)$ at every edge e. The upper dotted curve represents the function $f^{(k)}$ and the upper solid curve (consisting of horizontal and vertical segments) represents the profile $\bar{f}^{(k)}$. We remark that the height of the profile in phase $k-2$ (the lower solid curve) or in a phase less than $k-2$ is much smaller than that of $\bar{f}^{(k)}$.

3.2.3 Dynamic Program via Forgetting Tasks

The naive recursive algorithm does not have polynomial time performance since in each phase there are polynomially many possibilities for our guesses and the depth of our recursion can be $\omega_\varepsilon(1)$. Therefore, we want to embed our routine into a dynamic program. Ideally, we want to have one DP-cell for each arising subproblem, specified by the current phase k and the previously guessed huge tasks and profiles in phases up to $k-1$. However, the number of DP-cells is super-polynomial. To bound this number by a polynomial, there are two major difficulties: First, the number of the previously guessed profiles is $k = \omega_\varepsilon(1)$ and we could afford to remember $O_\varepsilon(1)$ profiles but not all of them. Secondly, the number of previously guessed huge tasks can be $\Omega_\varepsilon(k) = \omega_\varepsilon(1)$ and again we could afford to remember $O_\varepsilon(1)$ tasks.

We fix these issues by using the slack. Intuitively, we argue that we are able to "forget" some of the above information if in phases k, $k+1$, and so on, we do not use the full capacity of each edge but leave a bit of slack, just like OPT'' does. The "forgotten" information will represent part of previously selected tasks but we argue that those have small total demand compared to the slack.

First, we argue that we only need to remember one previous profile thanks to the slack. Indeed, every edge e of type k has enough slack to accommodate the total demand of all profiles in phases up to $k-2$, as shown in the following lemma.

Lemma 3.6. *Let e be an edge of type k. We have $\sum_{j=-1}^{k-2} \bar{f}^{(j)}(e) \leq \delta^{(k)}/3$.*

The proof of Lemma 3.6 combines Lemma 3.3 with a geometric sum argument. Therefore, from Lemma 3.6, in phase k it is sufficient to remember the profile in phase $k-1$ only.

Next, we argue that we only need to remember $O_\varepsilon(1)$ previous huge tasks thanks to the slack. We can afford to remember the tasks in $\mathrm{OPT}_H''^{(k-1)}$ since there are only $O_\varepsilon(1)$ of them. A possibly problematic huge task is a task $i \in \mathrm{OPT}_H''^{(j)} \setminus \mathrm{OPT}_H''^{(j+1)}$ for some $j < k-1$. Consider an index j where such tasks exist. On one hand, there are at most $2/\varepsilon^5$ such tasks by Proposition 3.2; on the other hand, every such task i satisfies $d(i) < \varepsilon^2 \cdot \delta^{(j+1)}$. Therefore, for $j = k-3$ we have $d(\mathrm{OPT}_H''^{(j)} \setminus \mathrm{OPT}_H''^{(j+1)}) \leq 2\varepsilon \cdot \delta^{(k)}$. Using a geometric sum argument, we obtain the following

lemma.

LEMMA 3.7. *The demand of* $\left(\bigcup_{j=-1}^{k-3} \mathrm{OPT}_H''^{(j)}\right) \setminus \mathrm{OPT}_H''^{(k-2)}$ *is at most* $\delta^{(k)}/3$.

Therefore, from Lemma 3.7, in phase k it is sufficient to remember the huge tasks in phases $k-2$ and $k-1$ only, where there are $O_\varepsilon(1)$ many tasks for each of them.

DP-cells. Each cell in our DP-table is characterized by

- two edges e and e' representing the subpath E' of the current subproblem (including e and e'),
- an integer $k \geq -1$ indicating that $\mathrm{type}(e) = \mathrm{type}(e') = k$,
- a function $\bar{f}_{\mathrm{ALG}}^{(k-1)} \colon E \mapsto \mathbb{N}$ satisfying the profile properties listed in Lemma 3.3,
- sets of huge tasks $\mathrm{ALG}_H^{(k-2)}$ and $\mathrm{ALG}_H^{(k-1)}$ each of size at most $2/\varepsilon^5$.

The function $\bar{f}_{\mathrm{ALG}}^{(k-1)}$ represents a previously guessed profile and the sets $\mathrm{ALG}_H^{(k-2)}$ and $\mathrm{ALG}_H^{(k-1)}$ represent previously guessed sets of huge tasks. They are counterparts of $\bar{f}^{(k-1)}$, $\mathrm{OPT}_H''^{(k-2)}$, and $\mathrm{OPT}_H''^{(k-1)}$.

LEMMA 3.8. *The number of DP-cells is bounded by a polynomial.*

Proof. There are $O(n)$ choices for e and for e' and $O(n^{2/\varepsilon^5})$ choices for $\mathrm{ALG}_H^{(k-2)}$ and for $\mathrm{ALG}_H^{(k-1)}$. The type $k \geq -1$ is at most $O(\log \max_e u(e))$, which is at most polynomial to the size of the input. There are $n^{O(1/\varepsilon^3)}$ choices for $\bar{f}_{\mathrm{ALG}}^{(k-1)}$ by Lemma 3.3. □

DP-solution. A *solution for a DP-cell* assigns a type $\mathrm{type}(e) \geq k$ to each edge $e \in E'$ (which then defines $\delta_e := (1/\varepsilon^2)^{\mathrm{type}(e)}$), chooses profiles $\bar{f}_{\mathrm{ALG}}^{(j)}$ for all $j \geq k$, and selects sets of tasks $\mathrm{ALG}_H^{(j)}$ and $\mathrm{ALG}_T^{(j)}$ for all $j \geq k$ with maximum total profit, such that

1. on the left (resp., right) of e_M the types of the edges are non-decreasing (resp., non-increasing);
2. for every type $j \geq k$, the function $\bar{f}_{\mathrm{ALG}}^{(j)} \colon E \mapsto \mathbb{N}$ satisfies the profile properties listed in Lemma 3.3;
3. for every type $j \geq k$ and every $e \in E$, $d(T_e \cap \mathrm{ALG}_T^{(j)}) \leq \bar{f}_{\mathrm{ALG}}^{(j)}(e)$;
4. for every type $j \geq k$, $\mathrm{ALG}_H^{(j)}$ contains at most $2/\varepsilon^5$ huge tasks;
5. for every type $j \geq k$ and every edge $e \in E'$ of type j, the total demand of the tasks from $T_e \cap (\mathrm{ALG}_H^{(j)} \cup \mathrm{ALG}_H^{(j-1)} \cup \mathrm{ALG}_H^{(j-2)})$ plus $\bar{f}_{\mathrm{ALG}}^{(j)}(e) + \bar{f}_{\mathrm{ALG}}^{(j-1)}(e)$ is at most $u(e) - 2\delta^{(j)}/3$.

Computation for DP-cell. To compute the solution for a DP-cell, first, we guess $e_L^{(k)}$, $e_R^{(k)}$, the profile $\bar{f}_{\mathrm{ALG}}^{(k)}$, and the set $\mathrm{ALG}_H^{(k)}$ for the huge tasks. Next, we compute a $(1 - O(\varepsilon))$-approximate solution $\mathrm{ALG}_T^{(k)}$ to the problem of selecting tiny tasks that are feasible w.r.t. the edge capacities given by $\bar{f}_{\mathrm{ALG}}^{(k)}$ (see Lemma 3.5). Finally, we recurse on the subproblem defined by the subpath strictly between $e_L^{(k)}$ and $e_R^{(k)}$, by the type $k+1$, by the profile $\bar{f}_{\mathrm{ALG}}^{(k)}$, and by the sets $\mathrm{ALG}_H^{(k-1)}$ and $\mathrm{ALG}_H^{(k)}$ for the huge tasks.

Now we obtain the main theorem of this section.

THEOREM 3.2. *There is a PTAS for UFP if all input tasks use a single edge.*

Proof. Consider the root DP-cell where $E' = E$, $k = -1$, $\bar{f}_{\mathrm{ALG}}^{(k-1)} \equiv 0$, and $\mathrm{ALG}_H^{(k-2)} = \mathrm{ALG}_H^{(k-1)} = \emptyset$. Let $\{\mathrm{ALG}_H^{(j)}\}_{j \geq -1}$ and $\{\mathrm{ALG}_T^{(j)}\}_{j \geq -1}$ be the sets in the DP-solution at this cell. Let T' be the union of these sets.

First, we show that T' is a feasible solution to UFP, i. e., $d(T_e \cap T') \leq u(e)$ for every $e \in E$. Consider an edge e of type k. The set $T_e \cap T'$ consists of two parts: huge tasks $\bigcup_{-1 \leq j \leq k} T_e \cap \mathrm{ALG}_H^{(j)}$ and tiny tasks $\bigcup_{-1 \leq j \leq k} T_e \cap \mathrm{ALG}_T^{(j)}$.

The huge tasks $\bigcup_{-1 \leq j \leq k} T_e \cap \mathrm{ALG}_H^{(j)}$ belong to either $T_e \cap (\mathrm{ALG}_H^{(k)} \cup \mathrm{ALG}_H^{(k-1)} \cup \mathrm{ALG}_H^{(k-2)})$ or $\left(\bigcup_{-1 \leq j \leq k-3} \mathrm{ALG}_H^{(j)}\right) \setminus \mathrm{ALG}_H^{(k-2)}$. The demand of the latter is at most $\delta^{(k)}/3$ which can be shown similarly as Lemma 3.7.

From the third property of DP-solutions, the demand of the tiny tasks $\bigcup_{-1 \leq j \leq k} T_e \cap \mathrm{ALG}_T^{(j)}$ is at most $\sum_{-1 \leq j \leq k} \bar{f}_{\mathrm{ALG}}^{(j)}(e)$, which is the sum of $(\bar{f}_{\mathrm{ALG}}^{(k)}(e) + \bar{f}_{\mathrm{ALG}}^{(k-1)}(e))$ and $(\sum_{-1 \leq j \leq k-2} \bar{f}_{\mathrm{ALG}}^{(j)}(e))$. The latter term in this sum is at most $\delta^{(k)}/3$ using Lemma 3.6.

Combining the above, we have $d(T_e \cap T')$ is at most $d\left(T_e \cap (\mathrm{ALG}_H^{(k)} \cup \mathrm{ALG}_H^{(k-1)} \cup \mathrm{ALG}_H^{(k-2)})\right) + \left(\bar{f}_{\mathrm{ALG}}^{(k)}(e) + \bar{f}_{\mathrm{ALG}}^{(k-1)}(e)\right) + 2\delta^{(k)}/3$, which is at most $u(e)$ by the fifth property of DP-solution.

Next, we show that T' is near-optimal, i. e., $w(T') \geq (1 - O(\varepsilon)) \cdot \mathrm{opt}$. Consider the solution OPT''' obtained from OPT'' by replacing the tiny tasks in OPT'' by those obtained by filling the profiles $\bar{f}^{(k)}$ in a near-optimal way, see Lemma 3.5. From Lemmas 3.2 and 3.5, we have $\mathrm{opt}''' \geq (1 - O(\varepsilon)) \cdot \mathrm{opt}$. By Lemma 3.2, Proposition 3.1, and Lemmas 3.3 and 3.4, OPT''' corresponds to a feasible solution at the root DP-cell. Thus we have $w(T') \geq \mathrm{opt}''' \geq (1 - O(\varepsilon)) \cdot \mathrm{opt}$ since T' is a solution for the root DP-cell with maximum profit.

Finally, we analyze the running time of the DP. By

Lemma 3.8, the number of DP cells is polynomial. For each DP cell, guessing the huge tasks and the profile takes polynomial time and computing a near-optimal set of tiny tasks to fill a guessed profile again takes polynomial time (Lemma 3.5). Therefore, the overall running time of the DP is polynomial. □

3.3 Some Extensions It is possible to generalize our PTAS for rooted UFP to the case where each task uses at least one edge in an edge-subset $R = \{r_1, \ldots, r_\ell\}$ where the number ℓ of edges is $O(1)$ (subsequently called *ℓ-rooted UFP*).

One of the core reasons why our machinery for rooted UFP works is the following: for each k the edges of type k in OPT$''$ form at most two subpaths. For ℓ-rooted UFP with $\ell = O(1)$ we can find a near-optimal solution OPT$''$ with the same properties as in Lemma 3.2 (with larger constants though) such that the edges of type k in OPT$''$ form $O(\ell) = O(1)$ subpaths. We can adjust our previous DP such that it also works in this case, and still runs in polynomial time. The proof of following theorem is in the full version of the paper.

THEOREM 3.3. *There is a PTAS for ℓ-rooted UFP with $\ell = O(1)$.*

The above result has the following consequence on *no-containment UFP*. An instance of no-containment UFP is defined as an instance of UFP in which there are no two tasks i, j such that $P(i) \subseteq P(j)$. Therefore, for each pair of tasks i, j such that $s(i)$ is on the left of $s(j)$, we have $t(i)$ is on the left of $t(j)$. We reduce the case of no-containment UFP to the case of $(1/\varepsilon)$-rooted UFP by losing only a factor $1+\varepsilon$ in the approximation.

LEMMA 3.9. *Given a no-containment UFP instance, we can compute a set of edges $\bar{E} \subseteq E$ such that each input tasks uses exactly one edge in $\bar{E}$.*

Proof. We identify the set of vertices V with consecutive integers from left to right. Let i be the task with left-most start vertex $s(i)$; note that by our assumption in Section 1.3 this task is unique. We define the first edge of $\bar{E}$ to be $\{s(i), s(i)+1\}$ and the second edge to be $\{t(i), t(i)+1\}$.

We construct the remaining instance inductively. Let e be the right-most edge of the current $\bar{E}$. Then we set r to be the right-most end-vertex among all tasks in T_e and we add the edge $\{r, r+1\}$ to $\bar{E}$. If at some point $T_e = \emptyset$ and we have not reached the end of the instance, we add to $\bar{E}$ the first edge e' on the right of e such that $T_{e'} \neq \emptyset$.

To see that each task uses at most one edge, let us consider the left-most edge $e \in \bar{E}$ used by a task i. In particular, $i \in T_e$. Therefore the next edge e' we added to $\bar{E}$ (after adding e) will be on the right of $t(i)$ and thus $i \notin T_{e'}$.

Now suppose by contradiction that there was a task i that does not use an edge of $\bar{E}$. By construction there must be two consecutive edges $e, e' \in \bar{E}$ such that $s(i)$ lies on the right of e and $t(i)$ lies on the left of e'. Let $i' \in T_e$ be the task such that $e' = \{t(i'), t(i')+1\}$. Then $e \in P(i)$ and thus $P(i)$ is a subpath of $P(i')$, contradicting the no-containment assumption. □

Let $\bar{e}_1, \ldots, \bar{e}_{n'}$ denote the edges in $\bar{E}$ in order. See Figure 2. Now we draw an offset $a \in \{0, \ldots, 1/\varepsilon - 1\}$ uniformly at random and delete all tasks crossing $\bar{e}_{a+\ell/\varepsilon}$ for each $\ell \in \mathbb{N}$. Since each task uses exactly one edge in $\bar{E}$, the profit is decreased by an ε fraction in expectation.

LEMMA 3.10. *Let $\bar{T}$ denote the set of deleted tasks. Then $\mathbb{E}[w(\text{OPT} \cap \bar{T})] \leq \varepsilon \cdot \text{opt}$.*

We can derandomize the algorithm by trying all $1/\varepsilon$ offsets a. The remaining instance splits into at most n independent sub-instances where each of them is a $(1/\varepsilon)$-rooted-UFP instance. We apply our PTAS in Theorem 3.3 to each of them.

THEOREM 3.4. *There is a PTAS for no-containment UFP.*

Acknowledgements. We would like to thank the anonymous reviewers for their comments and suggestions.

References

[1] A. Adamaszek, P. Chalermsook, A. Ene, and A. Wiese. Submodular unsplittable flow on trees. In *IPCO*, volume 9682 of *Lecture Notes in Computer Science*, pages 337–349, 2016.

[2] A. Anagnostopoulos, F. Grandoni, S. Leonardi, and A. Wiese. Constant integrality gap LP formulations of unsplittable flow on a path. In *IPCO*, pages 25–36, 2013.

[3] A. Anagnostopoulos, F. Grandoni, S. Leonardi, and A. Wiese. A mazing 2+ε approximation for unsplittable flow on a path. In *SODA*, pages 26–41, 2014.

[4] E. M. Arkin and E. B. Silverberg. Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics*, 18:1–8, 1987.

[5] N. Bansal, A. Chakrabarti, A. Epstein, and B. Schieber. A quasi-PTAS for unsplittable flow on line graphs. In *STOC*, pages 721–729. ACM, 2006.

[6] N. Bansal, Z. Friggstad, R. Khandekar, and R. Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. In *SODA*, pages 702–709, 2009.

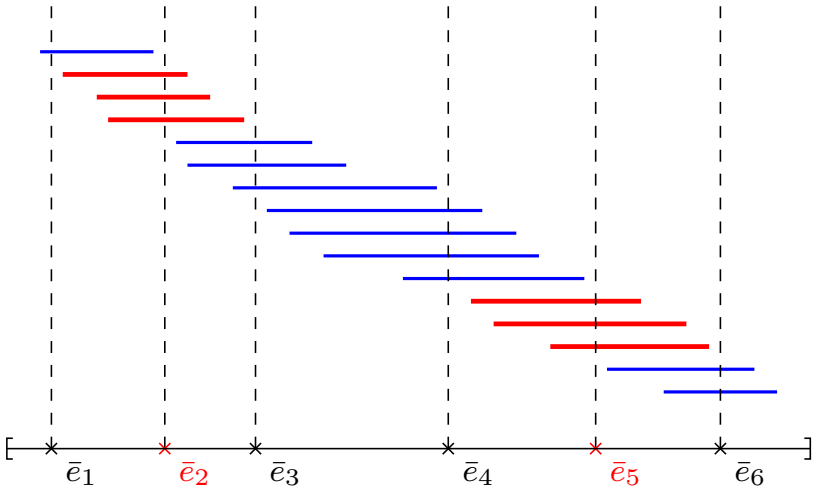

Figure 2: The horizontal base line represents the entire path E. The set of tasks are represented by the solid segments. (We omit their heights in the drawing.) In this example, the set $\bar{E}$ (obtained from Lemma 3.9) consists of the edges $\bar{e}_1, \bar{e}_2, \bar{e}_3, \bar{e}_4, \bar{e}_5, \bar{e}_6$. Suppose $\varepsilon = 1/3$ and $\bar{T} = \{\bar{e}_2, \bar{e}_5\}$. The set of deleted tasks are in bold.

[7] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. In *STOC*, pages 735–744, 2000.

[8] R. Bar-Yehuda, M. Beder, Y. Cohen, and D. Rawitz. Resource allocation in bounded degree trees. In *ESA*, pages 64–75, 2006.

[9] J. Batra, N. Garg, A. Kumar, T. Mömke, and A. Wiese. New approximation schemes for unsplittable flow on a path. In *SODA*, pages 47–58, 2015.

[10] P. Bonsma, J. Schulz, and A. Wiese. A constant-factor approximation algorithm for unsplittable flow on paths. *SIAM Journal on Computing*, 43:767–799, 2014.

[11] G. Călinescu, A. Chakrabarti, H. J. Karloff, and Y. Rabani. An improved approximation algorithm for resource allocation. *ACM Transactions on Algorithms*, 7:48:1–48:7, 2011.

[12] V. T. Chakaravarthy, A. R. Choudhury, S. Gupta, S. Roy, and Y. Sabharwal. Improved algorithms for resource allocation under varying capacity. In *ESA*, pages 222–234, 2014.

[13] A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar. Approximation algorithms for the unsplittable flow problem. *Algorithmica*, 47:53–78, 2007.

[14] C. Chekuri, A. Ene, and N. Korula. Unsplittable flow in paths and trees and column-restricted packing integer programs. Manuscript.

[15] C. Chekuri, A. Ene, and N. Korula. Unsplittable flow in paths and trees and column-restricted packing integer programs. In *APPROX-RANDOM*, pages 42–55, 2009.

[16] C. Chekuri, M. Mydlarz, and F. Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Transactions on Algorithms*, 3, 2007.

[17] B. Chen, R. Hassin, and M. Tzur. Allocation of bandwidth and storage. *IIE Transactions*, 34:501–507, 2002.

[18] M. Chrobak, G. Woeginger, K. Makino, and H. Xu. Caching is hard, even in the fault model. In *ESA*, pages 195–206, 2010.

[19] A. Darmann, U. Pferschy, and J. Schauer. Resource allocation with time intervals. *Theoretical Computer Science*, 411:4217–4234, 2010.

[20] A. M. Frieze and M. R. B. Clarke. Approximation algorithms for the m-dimensional $0-1$ knapsack problem: worst-case and probabilistic analyses. *European J. Oper. Res.*, 15:100–109, 1984.

[21] N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.

[22] F. Grandoni, S. Ingala, and S. Uniyal. Improved approximation algorithms for unsplittable flow on a path with time windows. In *WAOA*, pages 13–24, 2015.

[23] F. Grandoni and T. Rothvoß. Pricing on paths: A PTAS for the highway problem. In D. Randall, editor, *SODA*, pages 675–684. SIAM, 2011.

[24] S. Leonardi, A. Marchetti-Spaccamela, and A. Vitaletti. Approximation algorithms for bandwidth and storage allocation problems under real time constraints. In *FSTTCS*, pages 409–420, 2000.

[25] C. A. Phillips, R. N. Uma, and J. Wein. Off-line admission control for general scheduling problems. In *SODA*, pages 879–888, 2000.

A Proof of Lemma 3.1

Proof. The proof is based on a random shift argument. Here we restrict ourselves to the case of rooted UFP. However, it is not hard to extend the proof to general UFP.

For each edge $e \in P$, we define $\text{type}(e) \in \mathbb{N}$ as the integer $j \in \mathbb{N}$ such that $u(e) \in [(3/\mu)^j, (3/\mu)^{j+1})$. Then $\text{type}(e)$ is non-decreasing from the leftmost edge of E to e_M and non-increasing from e_M to the rightmost edge of E. We say that a task i is of type j if $P(i)$ uses an edge of type j and no edge of type $j-1$ or lower. For every $j \in \mathbb{N}$, let $T^{(j)} \subseteq T$ denote all tasks of type j. We choose $j^* \in [0, 1/\varepsilon)$ that minimizes the weight of $\bigcup_{j \equiv j^* \bmod (1/\varepsilon)} T^{(j)} \cap \text{OPT}$. This weight is at most $\varepsilon \cdot \text{opt}$. Thus by losing an ε fraction of the profit, we may assume that a solution contains no task of type j such that $j \equiv j^* \bmod (1/\varepsilon)$.

In the reduction, we remove from T all the tasks of type j such that $j \equiv j^* \bmod (1/\varepsilon)$ and classify the remaining tasks into groups such that every group is the union of $T^{(j)}$ for the types j that are between two consecutive removed types.[4] This then decomposes the initial problem into subproblems, each with a constant range of edge capacities.

Formally, for every integer k, we construct a subproblem as follows: let $A_k \subseteq T$ be the set of the tasks whose types are in $[a_k + 1, a_k + (1/\varepsilon))$, where $a_k = k/\varepsilon + j^*$, and let $E_k \subseteq E$ be the set of the edges whose types are at least $a_k + 1$. For each edge $e \in E_k$, its capacity $u_k(e)$ in the subproblem is defined by $u_k(e) := \min(u(e), U)$, where $U := 2 \cdot (3/\mu)^{a_k + (1/\varepsilon)}$. We observe that $u_k(e) \in [(\mu/3)^{1/\varepsilon} \cdot U, U]$ for every $e \in E_k$. We claim that $\text{OPT} \cap A_k$ is a solution to the subproblem defined by A_k, E_k, and u_k: every task in $\text{OPT} \cap A_k$ must use one out of two edges of type at most $a_k + (1/\varepsilon) - 1$,[5] so the total demand of these tasks is less than U. Let S_k be the optimal solution to this subproblem. Then $w(S_k) \geq w(\text{OPT} \cap A_k)$.

Let S be the union of the solutions S_k to all subproblems. Then $w(S) \geq (1-\varepsilon)\text{opt}$. It only remains to show that S is a solution to the initial problem if the edge capacities are increased by a factor $1 + \mu$. Consider an edge e of type j. Let $k \in \mathbb{N}$ be such that $j \in [a_k + 1, a_k + (1/\varepsilon)]$. We observe that a task in S using e can only belong to a set $S_{k'}$ for $k' \leq k$. The total demand of the tasks in S_k using the edge e is at most $u_k(e) \leq u(e)$. For every type $k' < k$, the total demand of the tasks in $S_{k'}$ using the edge e is at most $u_{k'}(e) \leq 2 \cdot (3/\mu)^{a_{k'} + (1/\varepsilon)}$. Summing over all types $k' \leq k$, we have the total demand of all tasks in S using e is at most $u(e) + 3 \cdot (3/\mu)^{a_k} \leq (1+\mu) \cdot u(e)$.

The number of subproblems is bounded by $\max_e \text{type}(e)$, which is polynomial to the size of the input since $\text{type}(e) = O(\log u(e))$. This completes the proof. □

[4] j^* is unknown, but we can enumerate each of the $(1/\varepsilon)$ possibilities of j^*.

[5] They are the leftmost edge to the right of e_M and the rightmost edge to the left of e_M that have type at most $a_k + (1/\varepsilon) - 1$.