Faster $(1+\epsilon)$-approximation for Unsplittable Flow on a Path via resource augmentation and back

Fabrizio Grandoni
IDSIA, USI-SUPSI, Switzerland

Tobias Mömke
Department of Computer Science, University of Augsburg, Germany

Andreas Wiese
Department of Industrial Engineering, Universidad de Chile, Chile

Abstract

Unsplittable flow on a path (UFP) is an important and well-studied problem. We are given a path with capacities on its edges, and a set of tasks where for each task we are given a demand, a subpath, and a weight. The goal is to select the set of tasks of maximum total weight whose total demands do not exceed the capacity on any edge. UFP admits an $(1+\epsilon)$-approximation with a running time of $n^{O_\epsilon(\text{poly}(\log n))}$, i.e., a QPTAS [Bansal et al., STOC 2006; Batra et al., SODA 2015] and it is considered an important open problem to construct a PTAS. To this end, in a series of papers polynomial time approximation algorithms have been developed, which culminated in a $(5/3+\epsilon)$-approximation [Grandoni et al., STOC 2018] and very recently an approximation ratio of $(1+\frac{1}{e+1}+\varepsilon) < 1.269$ [Grandoni et al., 2020]. In this paper, we address the search for a PTAS from a different angle: we present a faster $(1+\epsilon)$-approximation with a running time of only $n^{O_\epsilon(\log\log n)}$. We first give such a result in the relaxed setting of resource augmentation and then transform it to an algorithm *without* resource augmentation. For this, we present a framework which transforms algorithms for (a slight generalization of) UFP under resource augmentation in a black-box manner into algorithms for UFP *without* resource augmentation, with only negligible loss.

2012 ACM Subject Classification Theory of computation → Dynamic programming; Theory of computation → Packing and covering problems; Theory of computation → Rounding techniques

Keywords and phrases Approximation Algorithms Unsplittable Flow Dynamic Programming

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.50

Funding *Fabrizio Grandoni*: Partially supported by the SNSF Excellence Grant 200020B_182865/1.
Tobias Mömke: Partially supported by the DFG Grant 439522729 (Heisenberg-Grant), the ERC Advanced Investigators Grant 695614 (POWVER), and by DFG Grant 389792660 as part of TRR 248 (CPEC).
Andreas Wiese: Partially supported by FONDECYT Regular grants 1170223 and 1200173.

1 Introduction

The unsplittable flow on a path problem (UFP) is a natural packing problem which has received a lot of attention, e.g., [3, 17, 15, 6, 7, 4, 8, 9]. We are given a path $G = (V, E)$ with a capacity $u(e) \in \mathbb{N}_0$ for each edge $e \in E$ and a set of tasks T. For each task $i \in T$ we are given a sub-path $P(i)$ of E, a demand $d(i) \in \mathbb{N}_0$, and a weight (or profit) $w(i) \in \mathbb{N}_0$. For any set of tasks $T' \subseteq T$, we define $d(T') := \sum_{i \in T'} d(i)$ and $w(T') := \sum_{i \in T'} w(i)$ and for each $e \in E$ we define $T_e \subseteq T$ to be the set of all tasks $i \in T$ using e, i.e., such that $e \in P(i)$. Similarly we sometimes say that i is contained, contains, or intersects a subpath P if $P(i)$ does. The goal is to select a set of tasks $T' \subseteq T$ of maximum total weight $w(T')$ such that T' obeys the edge capacities, i.e., $d(T' \cap T_e) \leq u(e)$ for each edge $e \in E$. We denote by n the size of the input, and hence in particular $|T| \leq n$ and $|E| \leq n$.

UFP is a generalization of Knapsack (i.e., if $|E| = 1$) and it is motivated by various applications. For example, the path G can represent a network with a chain of communication links in which we seek to select the most profitable set of possible transmissions that obey the given edge capacities. Also, the edges in E can correspond to discrete time slots, each task models a job that we might want to execute, and the edge capacities model the available amount of a resource shared by the jobs like energy or machines. Also, there is a connection between UFP and general caching, i.e., where pages can have different (possible non-unit) sizes and different costs for being evicted [12].

There is a $(1+\epsilon)$-approximation algorithm known for UFP with running time of $n^{O_\epsilon(\log n)}$ [7], i.e., a QPTAS (improving an earlier QPTAS in [5]), and it has been a long-standing open problem to construct a PTAS. Towards this goal, polynomial time approximation algorithms for UFP have been developed and the best known approximation ratio has been gradually improved from $O(\log n)$ [6] to $7+\epsilon$ [8], $2+\epsilon$ [3], and $5/3+\epsilon$ [17], while the currently best known ratio ratio is $1 + \frac{1}{1+e} + \epsilon < 1.269$ [15].

1.1 Our Contribution

In this paper, we contribute to the search for a PTAS for UFP from a different angle: we improve the *running time* of the known QPTAS for UFP [7] and present a $(1+\epsilon)$-approximation with a running time of only $n^{O_\epsilon(\log \log n)}$. Hence, our result is in the same spirit as similar improvements for the Maximum Independent Set of Rectangles problem in [13] and precedence constrained scheduling for unit-size jobs in [18].

We first present our result for the case of resource augmentation, i.e., when we allow ourselves to increase the edge capacities by a factor of $1+\delta$ for some constant $\delta > 0$ while the compared optimal solution OPT does not have this privilege. Let $u_{\min}$ and $u_{\max}$ denote the minimum and maximum edge capacities, respectively. In that setting, it is easy to show that we can assume that the edge capacities are in a constant range, namely $u_{max} = O_{\varepsilon,\delta}(u_{min})$. We classify each task $i \in T$ to be *small* or *large*, depending on whether i uses a relatively small or a relatively large fraction of the available capacity on the edges of $P(i)$. Since we have a constant range of edge capacities, one can show easily that each edge e can be used by only a constant number of large tasks in OPT.

The high level strategy in the known QPTASs [5, 7] is to take the middle edge e^*, guess the large tasks using e^*, split the small tasks using e^* into $O(\log n)$ or $(\log n)^{O(1)}$ groups, and guess an *under-estimating capacity profile* for each group with $O_\epsilon(1)$ uniform steps. In more detail, in the latter step one guesses for each edge $e \in E$ approximately how much capacity from $u(e)$ is used in OPT by the tasks crossing e^* in each group. For each group one argues that one loses only a factor of $1+\epsilon$ in the profit by underestimating the true

capacity. Then one recurses on the subpaths of E on the left of e^* and on the right of e^* which yields a recursion depth of $O(\log n)$.

Instead, when we consider the small tasks using e^*, we use only *one over*-estimating profile with *non*-uniform step size for *all* tasks together with only $O_{\varepsilon,\delta}(\log\log n)$ steps. Each step is a power of $1+\delta$ in $[\frac{\delta}{\log n}u_{\min}, u_{\max}]$. This yields $O_{\varepsilon,\delta}(\log\log n)$ steps since the profile is monotone on the left and on the right of e^* and $u_{\max} = O_{\varepsilon,\delta}(u_{\min})$. Also, our justification for the error is very different. On some edges e the error is at most $\frac{\delta}{\log n}u_{\min}$ which accumulates to at most $O(\delta)u_{\min} \leq O(\delta)u(e)$ during the recursion. On the other edges e the error is at most a δ-fraction of the total demand used on e by tasks crossing e^*; over the recursion this can add up to at most $\delta u(e)$ since in OPT edge e is used by tasks with a total demand of at most $u(e)$.

When we recurse we employ another novelty: we consider the tasks crossing at least one of *two* specially defined edges e_1^*, e_2^*, and recurse in *three* subpaths induced by them: e_1^* is like before the middle edge of current subpath. The other edge e_2^* is chosen such that half of previously guessed steps are on the left and the other half on the right (like in the QPTAS in [7]). This ensures that *at the same time* the recursion depth is $O(\log n)$ and each recursive call is described by a subpath of E and only $O_{\varepsilon,\delta}(\log\log n)$ steps from previously guessed profiles. Thus, we can embed this recursion into a dynamic program (DP) with DP-table of size $n^{O_{\varepsilon,\delta}}(\log\log n)$.

Then, we present a new framework using which one can transform algorithms for UFP under resource augmentation (like ours) in a black-box manner into algorithms for UFP *without* resource augmentation, while losing only a factor of $1+\epsilon$ in the approximation ratio. To this end, we define a slight generalization of UFP that we denote by *Bonus-UFP*. The key difference to UFP is that in addition to profit from normal tasks, one receives a bonus for subpaths which do not completely contain the path of any selected task. Also, its instances are required to have a simpler structure than general UFP instances, yielding similar properties as obtained via resource augmentation. For example, on each edge one is allowed to select only a constant number of large tasks (independently of the actual edge capacities!), so they can be easily guessed in time $n^{O_\epsilon(1)}$ for each edge. Also, the capacity allocated for small tasks in OPT is intuitively in a constant range (similarly as above) and the notion of resource augmentation is defined such that this capacity for the small tasks is increased by a factor $1+\delta$.

Our framework directly transforms any algorithm for Bonus-UFP under resource augmentation to an algorithm for UFP *without* resource augmentation, while increasing the approximation ratio only by a factor $1+\epsilon$. The transformation uses the slack-lemma [16] which was employed in previous algorithms for UFP [16, 15, 17] in order to gain free capacity on the edges within quite complicated dynamic programs. With our framework one does not need to apply the slack-lemma and construct this technical machinery "by hand" anymore, but it is sufficient to design an algorithm for Bonus-UFP under resource augmentation and then the framework does the transformation automatically.

▶ **Theorem 1** (informal). *Given an α-approximation algorithm for Bonus-UFP under resource augmentation with a running time of $T(n)$, we can construct a $(1+\epsilon)\alpha$-approximation algorithm for UFP (without resource augmentation) with a running time of $T(n)n^{O_\epsilon(1)}$.*

We hope that this facilitates future research on UFP for (eventually) finding a PTAS. In particular, we believe that if one constructs an algorithms for UFP under resource augmentation then it is very likely that it can be adjusted to an algorithm for Bonus-UFP under resource augmentation. For example, we demonstrate that this can be easily done

with our algorithm for UFP under resource augmentation above, which yields the following theorem.

▶ **Theorem 2.** *For any $\epsilon > 0$ there is a $(1+\epsilon)$-approximation algorithm for UFP with a running time of $n^{O_\epsilon(\log\log n)}$.*

1.2 Other related work

There are some special cases of UFP for which PTASs are known, for example when there are $O(1)$ edges such that each input task uses one of them [16], each input task can be selected an unbounded number of times [16], or when the profit of each task is proportional to its demand [7]. Also, there is an FPT-$(1+\epsilon)$-algorithm known for the unweighted case of UFP where the fixed parameter is the cardinality of the optimal solution [19].

Variations of UFP have been studied like bagUFP where the input tasks are partitioned into bags, and we are allowed to select at most one task from each bag [14]. Also, since the natural LP-formulation of UFP suffers from an integrality gap of $\Omega(n)$ [9], stronger LP-formulations have been investigated [10, 2]. Furthermore, unsplittable flow has been studied on trees, where the best known results are a $O(\log^2 n)$-approximation [10]. This was generalized to a $O(k \cdot \log n)$-approximation [1] for submodular objectives where k denotes the pathwidth of the given tree (bounded by $O(\log n)$).

2 Faster approximation scheme for UFP with resource augmentation

We first provide a $(1+\epsilon)$-approximation algorithm for UFP with a running time of $n^{O_{\epsilon,\delta}(\log\log n)}$ for the simplified setting of *δ-resource augmentation* where we permit the algorithm to compute a solution S where for each edge $e \in E$ we have that $\sum_{i \in S \cap T_e} d(i) \leq (1+\delta)u(e)$ and we require the optimal solution OPT to respect the original capacities $u(\cdot)$, i.e., $\sum_{i \in \mathrm{OPT} \cap T_e} d(i) \leq u(e)$ for each $e \in E$. For a UFP instance with a path $G = (V, E)$ and edge capacities $u(\cdot)$, we define $u_{\min} := \min_{e \in E} u(e)$ and $u_{\max} := \max_{e \in E} u(e)$. First, we use the resource augmentation to reduce the general case to the setting of a constant range of (polynomially bounded) edge capacities where $u_{\max} \leq u_{\min}/\delta^{(2/\epsilon)}$.

▶ **Lemma 3.** *Assume that for any constants $\epsilon > 0$, $\delta > 0$, there is an α-approximation algorithm under δ-resource augmentation for UFP with a running time of $T(n)$ for instances such that $u_{\min} = n/\delta$ and $u_{\max} \leq u_{\min}/\delta^{2/\epsilon}$. Then there is an $(\alpha+\epsilon)$-approximation algorithm for UFP under 4δ-resource augmentation with a running time of $T(n)(n \log u_{\max})^{O_{\epsilon,\delta}(1)}$.*

Due to Lemma 3, in the following we assume that we are given an instance of UFP such that $n/\delta \leq u(e) \leq n/(\eta\delta)$ for each $e \in E$ and some $\epsilon, \delta > 0$, with $\eta := \delta^{2/\epsilon}$, and that we are in the setting of δ-resource augmentation. By contracting edges suitably we can assume w.l.o.g. that each edge $e \in E$ is the first or the last edge of the path $P(i)$ of some input task $i \in T$.

2.1 Recursive Algorithm

We describe our algorithm first as a recursive algorithm and then embed it into a dynamic program which will have a running time of $n^{O_{\epsilon,\delta}(\log\log n)}$.

Let $e^* \in E$ be an edge in the middle of E, i.e., such that at most $\lfloor |E|/2 \rfloor$ edges are on the left of e^* and at most $\lfloor |E|/2 \rfloor$ edges are on the right of e^*. We would like to guess the tasks in $\mathrm{OPT} \cap T_{e^*}$. Since there are too many possibilities for this, we first guess approximately how much capacity the tasks in $\mathrm{OPT} \cap T_{e^*}$ use on each edge $e \in E$. For any set of tasks $S \subseteq T$,

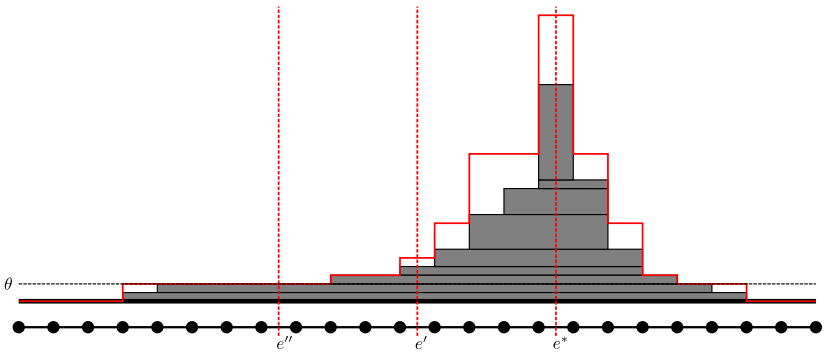

Figure 1 Overestimating profile $\mathrm{mp}(u_{T_{e^*}\cap \mathrm{OPT}})$ (the red line). The edge e' halves the number of steps and e'' the length of the path on the left-hand side of e^*.

we define $u_S(e) := \sum_{i\in S\cap T_e} d(i)$ for each $e \in E$. Observe that $u_{T_{e^*}\cap \mathrm{OPT}}$ is non-decreasing on the left of e^* and non-increasing on the right of e^*, since all tasks in $T_{e^*} \cap \mathrm{OPT}$ use e^*. We will guess an *over*-estimating profile $\mathrm{mp}_{T_{e^*}\cap \mathrm{OPT}} \colon E \to \mathbb{N}$ of $u_{T_{e^*}\cap \mathrm{OPT}}(e)$ defined below, with the properties that

- for each edge e it holds that $\mathrm{mp}_{T_{e^*}\cap \mathrm{OPT}}(e) \geq u_{T_{e^*}\cap \mathrm{OPT}}(e)$,
- $\mathrm{mp}_{T_{e^*}\cap \mathrm{OPT}}$ is a step-function with only $O_{\epsilon,\delta}(\log\log n)$ steps (i.e., E can be partitioned into $O_{\epsilon,\delta}(\log\log n)$ subpaths and $\mathrm{mp}_{T_{e^*}\cap \mathrm{OPT}}$ is constant on each subpath), and
- $\mathrm{mp}_{T_{e^*}\cap \mathrm{OPT}}(e)$ and $u_{T_{e^*}\cap \mathrm{OPT}}(e)$ do not differ too much on each edge $e \in E$.

Formally, let $D := \log n$; our overall algorithm will intuitively be a recursion with D levels. For any set of tasks $S \subseteq T$ we define the *minimal profile* $\mathrm{mp}_S \colon E \to \mathbb{N}$ of S as follows. Let $\theta := \lfloor u_{\min} \cdot \delta/D \rfloor$. If for an edge $e \in E$ it holds that $d(S \cap T_e) = 0$ then we define $\mathrm{mp}_S(e) := 0$. Otherwise, we define $\mathrm{mp}_S(e) := \lfloor \theta(1+\delta)^{k_e} \rfloor$ where k_e is the smallest integers such that $\theta(1+\delta)^{k_e} \geq d(S \cap T_e)$ (see Figure 1). It turns out that $\mathrm{mp}_S(\cdot)$ is a step-function with only $O_{\delta,\epsilon}(\log\log n)$ steps, assuming that there is an edge $\bar{e}$ that is used by all tasks in S (like the edge e^* above).

▶ **Lemma 4.** *There is a value $\Gamma = O_{\delta,\epsilon}(\log D) = O_{\delta,\epsilon}(\log\log n)$ such that for every edge $\bar{e} \in E$ and every set $S \subseteq T_{\bar{e}}$ with $d(S \cap T_e) \leq u(e)$ for each $e \in E$, we have that $\mathrm{mp}_S(\cdot)$ is a step-function with at most Γ steps.*

Proof. Recall that by Lemma 3 $u_{\min} = n/\delta$ and $u_{\max} \leq n/(\delta\eta)$. We therefore search for an upper bound on the smallest value k such that $\lfloor \theta(1+\delta)^k \rfloor \geq n/(\delta\eta)$. We have that

$$k \in O(\log_{1+\delta}\Big(\frac{n}{\delta\eta\theta}\Big)) = O\left(\log_{1+\delta}\Big(\frac{nD}{\eta\delta^2 u_{\min}}\Big)\right)$$
$$= O\Big(\log_{1+\delta}\Big(\frac{D}{\eta\delta^2}\Big)\Big) = O_{\delta,\epsilon}(\log D) = O_{\delta,\epsilon}(\log\log n).$$

The second claim follows due to monotonicity of the profile formed by the tasks $\mathrm{OPT}_S \cap T_e$. ◀

For any set S, the profile $\mathrm{mp}_S(\cdot)$ overestimates the true demand of the tasks S on each edge e. Therefore, we define the error of this estimation by $\mathrm{err}(\mathrm{mp}_S, e) := \mathrm{mp}_S(e) - u_S(e)$ for each edge $e \in E$. Intuitively, in our algorithm we will guess profiles mp_S in each of the $D = \log n$ recursion levels, such that at the end each edge e is in the support of at most $2D$ of these profiles. A key insight is the following lemma which implies that for each edge

e, the sum of the errors of these $2D$ profiles is bounded by the extra space of $\delta u(e)$ due to the resource augmentation. Intuitively, if $\mathrm{mp}_S(e) = \theta$ on some edge e, then the error is at most $\theta = u_{\min} \cdot \delta/D$ and for $2D$ profiles the errors of this type can accumulate to at most $2D \cdot \theta \leq 2\delta u_{\min} \leq 2\delta u(e)$. On the other hand, if $\mathrm{mp}_S(e) > \theta$ then the error is at most $\delta \cdot d(S \cap T_e)$, and this can accumulate to at most $\delta \cdot u(e)$ for all profiles together if these profiles correspond to disjoint sets of tasks $S_1, S_2, \ldots, S_{D'}$ that use at most $u(e)$ units of capacity on e altogether, for any D'.

▶ **Lemma 5.** *Let $S_1, S_2, \ldots, S_{D'} \subseteq T$ be disjoint sets of tasks with $D' \in O(D)$. If $\sum_j d(S_j \cap T_e) \leq u(e)$ for an edge e, then $\sum_{j=1}^{D'} err(\mathrm{mp}_{S_j \cap T_e}, e) \leq O(\delta) \cdot u(e)$.*

Proof. For each index j and edge e such that $\mathrm{mp}_{S_j \cap T_e}(e) = \lfloor \theta(1+\delta)^{k_j} \rfloor$ for some k_j, the error $\mathrm{err}(\mathrm{mp}_{S_j \cap T_e}, e)$ is bounded from above by $\lfloor \theta \rfloor$ if $k_j = 0$ and by $\lfloor \theta \cdot (1+\delta)^{k_j} \rfloor - \lceil \theta \cdot (1+\delta)^{k_j - 1} \rceil \leq \theta(1+\delta)^{k_j - 1} \cdot \delta \leq \delta d(S_j)$ if $k_j > 0$. (Observe that the rounding is valid because the demands are integers.) Thus $\mathrm{err}(\mathrm{mp}_{S_j \cap T_e}, e) \leq \max\{\theta, \delta d(S_j)\}$. Summing up over all errors we obtain

$$\sum_{j=1}^{D'} \mathrm{err}(\mathrm{mp}_{S_j \cap T_e}, e) \leq D' \cdot \theta + \sum_j \delta d(S_j \cap T_e) \leq \tfrac{D' u_{\min} \delta}{D} + \delta u(e) \in O(\delta) u(e).$$

◀

As mentioned above, we guess $\mathrm{mp}_{T_{e^*} \cap \mathrm{OPT}}$, which can be done in time $n^{O_{\delta,\epsilon}(\log \log n)}$ due to Lemma 4. Then, we compute the essentially most profitable set of tasks $T' \subseteq T_{e^*}$ that fits into $\mathrm{mp}_{T_{e^*} \cap \mathrm{OPT}}$, i.e., such that $d(T' \cap T_e) \leq \mathrm{mp}_{T_{e^*} \cap \mathrm{OPT}}(e)$ on each edge $e \in E$. This can be done using a PTAS in [16] for *rooted* UFP instances in which all input tasks share a common edge (like the edge e^* in our case).

▶ **Theorem 6** ([16])**.** *There is a PTAS for instances of UFP in which there is an edge that is used by every input task (rooted UFP). The same holds if there exist $O(1)$ edges such that each task uses at least one of them.*

We recurse on the subpaths on the left and on the right of e^*. Let us consider the left subpath, i.e., let E_L denote the path from the left-most edge of E up to e^*, not including e^*. (The recursion to the right is analogous.) We subdivide E_L into three parts determined by two edges e', e'' in E_L (or two parts if these edges coincide). We choose the first edge $e' \in E_L$ on the left of e^* such that the number of steps in the profile $\bar{u} := \mathrm{mp}_{\mathrm{OPT} \cap T_{e^*}}$ within E_L is halved (a similar trick was used in [7]). Formally, let $\Gamma = O_{\delta,\epsilon}(\log \log n)$ such that $\mathrm{mp}_{\mathrm{OPT} \cap T_{e^*}}$ is a step-function with Γ steps. We choose e' such that $\mathrm{mp}_{\mathrm{OPT} \cap T_{e^*}}$ restricted to the subpath of E_L on the left of e' is a step-function with at most $\lceil \Gamma/2 \rceil$ steps, and the same is true for $\mathrm{mp}_{\mathrm{OPT} \cap T_{e^*}}$ restricted to the subpath of E_L on the right of e'. The second edge e'' lies in the middle of E_L such that on the left of e'' there are at most $\lfloor |E_L|/2 \rfloor$ edges of E_L and the same is true on the right of e''. We may assume without loss of generality that e' lies on the left of e''.

We guess the profiles $\bar{u}' := \mathrm{mp}_{(\mathrm{OPT} \cap T_{e'}) \setminus T_{e*}}$ and $\bar{u}'' := \mathrm{mp}_{(\mathrm{OPT} \cap T_{e''}) \setminus (T_{e*} \cup T_{e'})}$ and apply Theorem 6 to compute essentially the most profitable subset of tasks that fit into $\bar{u}'$ (and $\bar{u}''$), among the input tasks $i \in T_{e'}$ with $P(i) \subseteq E_L$ (among the input tasks $i \in T_{e''} \setminus T_{e'}$ with $P(i) \subseteq E_L$). Then we recurse on each of the up to three components of $E_L \setminus \{e', e''\}$, where the parameter is the respective subpath and the profile $\bar{u} + \bar{u}' + \bar{u}''$ restricted to that subpath. Due to the choice of e'', the depth of our recursion is bounded by $D = \log n$. In particular, at the end each edge e is in the support of at most $O(D)$ guessed profiles. By Lemma 5, the total error of these profiles is at most $O(\delta)u(e)$ which is compensated by the resource augmentation. Due to the choice of e', one can show that in each recursive call the profile

of the parameter is a step function with only $4\Gamma = O_{\delta,\epsilon}(\log\log n)$ steps: each recursive call "inherits" only half of the steps that were given to its parent subproblem, and additionally up to 2Γ new steps due to the guessed profiles in the parent subproblem. This allows us to embed this recursion into a dynamic program (DP) with only $n^{O_{\delta,\epsilon}(\log\log n)}$ DP-cells as follows, and hence we obtain an overall running time of $n^{O_{\delta,\epsilon}(\log\log n)}$.

2.2 Dynamic program

We define the mentioned DP formally. In our DP table, we have a cell $(P, \hat{u})$ for each subpath $P \subseteq E$ and each function $\hat{u} : P \to \mathbb{N}_0$ such that $0 \leq \hat{u}(f) \leq n/(\delta\eta)$ for each edge $f \in P$ and $\hat{u}$ is a step-function with at most 4Γ steps, i.e., one can partition P into 4Γ subpaths such that $\hat{u}$ is constant on each one of them. The intuitive meaning of $\hat{u}$ is that for each edge $e \in P$, $\hat{u}(e)$ units of capacity have already been assigned to some tasks. The goal is to compute a set of tasks with maximum total weight and with $P(i) \subseteq P$ for each selected task i, such that on each edge $e \in P$ the selected tasks use a total capacity of at most $u(e) - \hat{u}(e)$.

▶ **Lemma 7.** *There are at most $n^{O_{\delta,\epsilon}(\log\log n)}$ many different DP cells.*

Proof. There are $O(n^2)$ possible choices for P. To determine the number of possible profiles $\hat{u}$, recall that $\hat{u}$ has at most $4\Gamma = O_{\delta,\epsilon}(\log\log n)$ many steps. There are $\binom{O(n)}{O_{\delta,\epsilon}(\log\log n)} \in n^{O_{\delta,\epsilon}(\log\log n)}$ possibilities to partition P into at most $O_{\delta,\epsilon}(\log\log n)$ subpaths. For each subpath, there are at most $n/(\delta\eta)$ possibilities for the value of $\hat{u}$ on this subpath, i.e., $(n/(\delta\eta))^{O_\delta(\log\log n)}$ possible combinations overall. Recall that $\eta = \delta^{2/\varepsilon}$. Multiplying these two numbers gives the total number of profiles. ◀

Given a DP cell $\mathrm{DP}(P, \hat{u})$. We identify two edges $e', e'' \in P$ similarly as above, i.e., we select $e' \in P$ such that if we restrict $\hat{u}$ to the edges of P on the left of e' or on the right of e', then $\hat{u}$ is a step-function with at most 2Γ steps. Also, we select e'' such that at most $\lfloor |P|/2 \rfloor$ edges of P lie on the left of e'', and at most $\lfloor |P|/2 \rfloor$ edges of P lie on the right of e''. Let P_1, P_2, P_3 denote the subpaths of P induced by e' and e'', i.e., the components of $P \setminus \{e', e''\}$.

Let $\mathcal{T}$ be the set of all pairs $(\hat{u}', \hat{u}'')$ such that $\hat{u}' : P \to \mathbb{N}_0$ and $\hat{u}'' : P \to \mathbb{N}_0$ are step-functions with at most Γ steps each, and on each edge $f \in P$ they use at most $(1+\delta)u(f)$ units of capacity, i.e., $\hat{u}(f) + \hat{u}'(f) + \hat{u}''(f) \leq (1+\delta)u(f)$. We apply the algorithm due to Theorem 6 to the instance whose input tasks contain all tasks in $i \in T_{e'}$ with $P(i) \subseteq P$ and with edge capacities given by $\hat{u}'$. Similarly, we apply this algorithm to the instance whose input tasks contain all tasks in $i \in T_{e''} \setminus T_{e'}$ with $P(i) \subseteq P$ and with edge capacities given by $\hat{u}''$. Denote by $\mathrm{alg}(\hat{u}')$ and $\mathrm{alg}(\hat{u}'')$ the respective solutions. With the pair $(\hat{u}', \hat{u}'')$ we associate the solution given by $\mathrm{alg}(\hat{u}') \cup \mathrm{alg}(\hat{u}'')$ and the solutions stored in the cells $DP(P_j, (\hat{u} + \hat{u}' + \hat{u}'')_{|P_j})$ for $j \in \{1, 2, 3\}$. We store in $\mathrm{DP}(P, \hat{u})$ the weight of the most profitable solution for all pairs $(\hat{u}', \hat{u}'') \in \mathcal{T}$, i.e.,

$$\mathrm{DP}(P, \hat{u}) := \max_{t=(\hat{u}', \hat{u}'') \in \mathcal{T}} ((\mathrm{alg}(\hat{u}')) + w(\mathrm{alg}(\hat{u}'')) + \textstyle\sum_{j=1}^{3} \mathrm{DP}(P_j, (\hat{u} + \hat{u}' + \hat{u}'')_{|P_j})).$$

Observe that $(\hat{u} + \hat{u}' + \hat{u}'')_{|P_j}$ has at most 4Γ steps as required for valid DP cells: the profile $\hat{u}$ restricted to P_j has at most 2Γ steps due to the halving and both $\hat{u}'$ and $\hat{u}''$ introduce at most Γ new steps each.

At the end we output $\mathrm{DP}(E, u_0)$ where u_0 is the profile with $u_0(e) = 0$ for all $e \in E$. By standard memoization we can also output the solution associated with $\mathrm{DP}(E, u_0)$.

We need to show that the DP computes a near-optimal solution. It is clear that we output a feasible solution since in the computation for each cell, the set $\mathcal{T}$ contains only pairs $(\hat{u}', \hat{u}'')$

such that $\hat{u}(f) + \hat{u}'(f) + \hat{u}''(f) \leq (1+\delta)u(f)$ for each edge $f \in P$. Intuitively, Lemma 5 allows us to argue that in each step we can guess for $\hat{u}'$ and $\hat{u}''$ the profiles $\text{mp}_{\text{OPT}' \cap T_{e'}}$ and $\text{mp}_{(\text{OPT}' \cap T_{e''}) \setminus T_{e'}}$, respectively, where OPT' denotes the tasks $i \in \text{OPT}$ with $P(i) \subseteq P$, without violating the capacities of the edges.

▶ **Lemma 8.** *The dynamic program computes a solution* ALG *which is feasible with $O(\delta)$-resource augmentation and $w(\text{ALG}) \geq (1-\epsilon)w(\text{OPT})$.*

Proof. Consider a DP-cell $(P, \hat{u})$ for which the DP defined the edges e', e'' when calculating its solution. Consider the profiles $\text{mp}_{\text{OPT} \cap \{i \in T_{e'} \mid P(i) \subseteq P\}}$ and $\text{mp}_{\text{OPT} \cap \{i \in T_{e''} \setminus T_{e'} \mid P(i) \subseteq P\}}$. Observe that all tasks $\{i \in \text{OPT} \cap (T_{e'} \cup T_{e''}) \mid P(i) \subseteq P\}$ fit into the combined profile and all remaining tasks from OPT within P will be considered in subproblems. If the two profiles are a feasible choice, then the DP obtains a profit of at least $(1-\epsilon)w(\{i \in \text{OPT} \cap (T_{e'} \cup T_{e''}) \mid P(i) \subseteq P\})$ from the tasks in $T_{e'} \cup T_{e''}$ when choosing these profiles. Therefore, it suffices to proof the claim that if in each recursive call up to some recursion level ℓ the DP chooses exactly these profiles, then in each subsequent recursive call of level $\ell + 1$ the corresponding profiles will be a feasible choice again. Inductively, we can conclude then that the DP obtains a profit of at least $(1-\epsilon)\text{opt}$.

To prove the mentioned claim, intuitively, due to the halving of the paths, the recursion depth is bounded from above by $O(\log n)$ and therefore the total error does not exceed the available amount of resource augmentation. Formally, we inductively maintain the invariant that for a DP cell $(P, \hat{u})$, if the profile $\hat{u}$ is composed of $2k$ profiles (from previous DP cells), then the length of P is bounded from above by $2n/2^k$. Recall that we assumed that $|E| \leq O(n)$. For each edge, we therefore have that at most $O(\log n)$ profiles overlap and by Lemma 5, the total error of is at most $O(\delta)u(e)$ for each edge e, i.e., $O(\delta)$-resource augmentation is sufficient. ◀

Together with Lemma 3 we conclude that we obtain a $(1+\epsilon)$-approximation algorithm for UFP with δ-resource augmentation.

3 Bonus-UFP

In this section we define formally the Bonus-UFP problem (BUFP) and formalize how an algorithm with resource augmentation for Bonus-UFP yields an algorithm for (ordinary) UFP *without* resource augmentation.

3.1 Problem definition

Like in (ordinary) UFP, in the Bonus-UFP problem we are given as input a path $G = (V, E)$ and a set of tasks T where each task $i \in T$ has a demand $d(i) \in \mathbb{N}$, a sub-path $P(i)$ of E, and a weight $w(i) \in \mathbb{N}_0$. The profit of a computed solution with task set $T' \subseteq T$ stems as usual from $w(T')$ and, additionally, from some *bonus profit* that we obtain from subpaths $E' \subseteq E$ (i.e., E' forms the edges of the respective subpath) such that no task $i \in T'$ satisfies that $P(i) \subseteq E'$ (but it might be that $P(i) \cap E' \neq \emptyset$). The amount of bonus of such a subpath E' depends on the tasks $i \in T'$ with $P(i) \cap E' \neq \emptyset$. The reader may imagine that we get more bonus from E' if fewer tasks from T' intersect E'. In particular, the computed solution consists of T' and of the subpaths E' from which the respective bonus profit is collected. Additionally, the input tasks are divided into large and small tasks, and on any edge we are allowed to select only $O(1)$ large tasks (in particular, for each edge e the large tasks in $OPT \cap T_e$ can be guessed easily in time $n^{O(1)}$). Also, for technical reasons, the selected

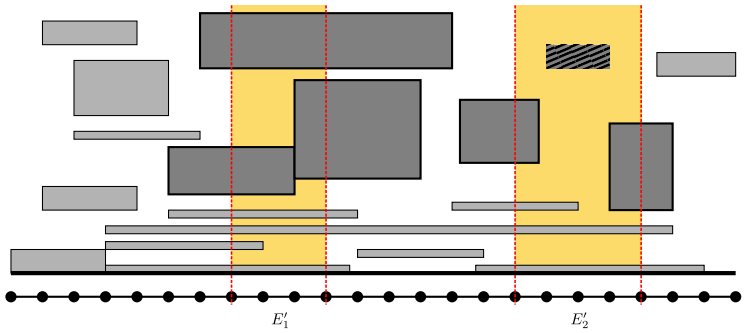

Figure 2 A Bonus-UFP instance with two bonus intervals E' and E''. The hatched task within E'' is not allowed. The bonus depends on the intervals and the tasks depicted in dark gray – the large tasks that intersect with an interval.

small tasks are allowed to use at most $2b$ units of capacity of each edge $e \in E$ for some given value b (see Figure 2).

Formally, in the input we are given a constant $\tau \in \mathbb{N}$ and values $\mu \in (0,1)$, $b > 0$ which partition the tasks T into a set of *large tasks* $T_L = \{i \in T : d(i) > \mu b\}$ and *small tasks* $T_S = \{i \in T : d(i) \leq \mu b\}$. For each pair of a subpath $E' \subseteq E$ and a set $L'_{int} \subseteq T_L$ such that

- the path of each task $i \in L'_{int}$ uses the leftmost or the rightmost edge of E' and
- $|L'_{int} \cap T_e| \leq \tau$ for each edge $e \in E'$

we are given a possible *bonus* $bn(E', L'_{int}) \geq 0$. As described above, we obtain the bonus $bn(E', L'_{int})$ if no selected task (large or small) is contained in E' and the tasks in L'_{int} are exactly the selected large tasks that use some edge of E' (the amount of bonus does not depend on the selected small tasks i with $P(i) \cap E' \neq \emptyset$). A feasible solution to an instance of BUFP consists of a set of tasks $R \subseteq T_L \cup T_S$ and a collection of node-disjoint subpaths $E_1, \dots E_q$ of G such that the following holds:

1. R is a feasible UFP solution, i.e., for every $e \in E$ it holds that $d(R \cap T_e) \leq u(e)$,
2. the small tasks in R use at most a capacity of $2b$ on each edge $e \in E$, i.e., $d(R \cap T_S \cap T_e) \leq 2b$,
3. each edge $e \in E$ is used by at most τ tasks in $R \cap T_L$, i.e., $|R \cap T_L \cap T_e| \leq \tau$, and
4. for no $i \in R$ the path $P(i)$ is contained in any E_j (but possibly $P(i) \cap E_j \neq \emptyset$).

The profit of the solution is $w(R)$ plus the bonus for each subpath E_j, where the latter depends on the large tasks in R that use E_j, i.e., the total profit is $w(R) + \sum_{j=1}^{q} bn(E_j, L_{int,j})$ where $L_{int,j} = \{i \in R \cap T_L | P(i) \cap E_j \neq \emptyset\}$.

We say that a solution to the above problem is feasible *under δ-resource augmentation* if we relax the first two conditions to

1.' for every $e \in E$ it holds that $d(R \cap T_e) \leq u(e) + \delta b$,
2.' for every $e \in E$ it holds that $d(R \cap T_S \cap T_e) \leq 2b + \delta b$.

So intuitively this increases the capacity for the small tasks by at least a factor $1 + O(\delta)$. Also, note that Property 3 ensures that each edge e is used by at most $\tau = O(1)$ large tasks, and hence we can guess the large tasks for each edge in time $n^{O(\tau)} = n^{O(1)}$.

For non-negative values $\alpha, \beta, \gamma \leq 1$ we say that a tri-criteria (α, β, γ)*-approximation algorithm for BUFP with δ-resource augmentation* is an algorithm that computes a solution that is feasible under δ-resource augmentation of profit at least $\alpha \mathrm{opt}_S + \beta \mathrm{opt}_L + \gamma \mathrm{opt}_B$, assuming that there exists an (optimal) solution R^* with bonuses $\left\{bn(E^*_j, L^*_{int,j})\right\}_j$ such that

$\text{opt}_S = w(R^* \cap T_S)$, $\text{opt}_L = w(R^* \cap T_L)$, and $\text{opt}_B = \sum_{j=1}^{q} bn(E_j^*, L_{int,j}^*)$. Then our main result states that if $\gamma = 1$ we can translate such an algorithm to an algorithm for normal UFP whose approximation ratio is $1/\min\{\alpha, \beta\}$. We will prove the following theorem in Section 3.2.

▶ **Theorem 9** (Black-box reduction). *Assume that there is a $(\alpha, \beta, 1)$-approximation algorithm for BUFP with δ-resource augmentation with a running time of $T_{\tau,\delta}(n, u_{\max})$. Then there is a $\frac{1+\epsilon}{\min\{\alpha,\beta\}}$-approximation algorithm for UFP (without resource augmentation) with a running time of $T_{O_\epsilon(1),O_\epsilon(1)}(n, u_{max}) \cdot (n \cdot \log u_{\max})^{O_\epsilon(1)}$.*

3.2 Black-box reduction

In this section we explain the key ideas for the black-box reduction due to Theorem 9 (omitting details due to space constraints). We start with a lemma indicating that there are near-optimal solutions in which each edge has a certain amount of slack. This slack is a constant fraction of the capacity used by small tasks according to a suitable definition of small and large tasks. To avoid confusion, we refer to large and small tasks as in definition of BUFP as bonus-large and bonus-small, respectively.

In the lemma, we assign a level $\ell(e)$ to each edge e. We say that a task i is of level ℓ if $P(i)$ includes an edge e with $\ell(e) = \ell$ and no edge e' with $\ell(e') < \ell$; let $T^{(\ell)} \subseteq T$ denote all tasks of level ℓ. Intuitively, each edge e of level ℓ has an amount of slack $\varepsilon^4 a(\ell)$ for some value $a(\ell)$ corresponding to level ℓ. We define small and large tasks such that on each edge e of level ℓ, the small tasks of level ℓ use a total capacity of at most $2b(\ell)$ with $b(\ell) = O_\epsilon(a(\ell))$ and each edge e of level ℓ is used by at most $\tau = O_\epsilon(1)$ large tasks i with $d(i) = \Omega(a(\ell))$. Formally, for an offset $h' \in \{0, ..., 1/\epsilon - 1\}$ and a value $h = \Theta(\frac{1}{\varepsilon} \ln \frac{7}{\varepsilon^2})$ defined in the next lemma, we set $a(\ell) := (1+\varepsilon)^{h'+\ell h \cdot (1+\frac{1}{\varepsilon})-\frac{h}{\varepsilon}}$ and $b(\ell) = a(\ell) \cdot (1+\varepsilon)^{\frac{h}{\varepsilon}}$. Let opt denote the weight of the optimal solution to the given instance.

▶ **Lemma 10** (Slack Lemma). *Let $\epsilon > 0$. There are two constants $\mu_1, \mu_2 \in (0, \varepsilon^4)$ with $\mu_1 < \mu_2/(1+\varepsilon)^{1/\varepsilon^3}$, values $h = \Theta(\frac{1}{\varepsilon} \ln \frac{7}{\varepsilon^2})$ and $\tau = O_\varepsilon(1)$, an offset h' contained in a set of size $O_\epsilon(1)$ that can be computed in polynomial time, a near-optimal solution* OPT *with* $w(\text{OPT}) \geq (1 - O(\epsilon))\text{opt}$*, and a level $\ell(e)$ for each edge $e \in E$ with the following properties. We define*

- $T_L := \{i \in T : d(i) \geq \mu_2 \cdot a(\ell(e))$ *for some edge* $e \in P(i)\}$ *(large tasks)*
- $T_S := \{i \in T : d(i) < \mu_1 \cdot b(\ell(e))$ *for every edge* $e \in P(i)\}$ *(small tasks)*
- $OPT_L = OPT \cap T_L$ *and* $OPT_S^{(\ell)} := \text{OPT} \cap T_S \cap T^{(\ell)}$ *for each* ℓ.

Then, for each edge e of level $\ell(e) = \ell$, it holds that:

- $d(T_e \cap OPT_L) + d(T_e \cap OPT_S^{(\ell)}) \leq u(e) - \varepsilon^4 a(\ell)$,
- $d(T_e \cap OPT_S^{(\ell)}) \leq 2b(\ell)$,
- *e is used by at most τ tasks $i \in \text{OPT}_L \cap T^{(\ell)}$ with $d(i) \geq \mu_2 \cdot a(\ell)$.*

Our goal is to define an algorithm BB that solves UFP given an oracle OR for BUFP with δ-resource augmentation. We will use the notation G^B, $T^B, T_S^B, T_L^B, u^B, \tau^B$ for the input path, input tasks, edge capacities, and threshold τ of the constructed instances of BUFP, while we use G, T, u to denote the corresponding values of the given instance of UFP.

Intuitively, BB solves a subproblem of BUFP for each maximally long path G^B in which each edge is of level at least ℓ for some given value $\ell \in \{0, \ldots, \ell_{max}\}$ (where $\ell_{\max}$ denotes the maximum level due to Lemma 10), and the bonus subpaths E_j selected in an optimal solution will correspond to maximal subpaths of G^B consisting of edges of level at least $\ell + 1$. With this interpretation, the amount of received bonus on such a subpath E_j is calculated

via subproblems of BUFP defined on this subpath E_j, level $\ell+1$, and any possible set of $O_\epsilon(1)$ selected large tasks of level ℓ that use at least one edge of E_j but which are not contained in E_j. The parameters of the calls to OR for this a subproblem (i.e., path G^B and level ℓ) will be intuitively as follows. We set $b=b(\ell)$, $\mu=\mu_1$, and $a=a(\ell)$. We place in T^B the tasks in T contained in G^B whose demand is at most $\mu_1 b(\ell)$ or at least $\mu_2 a(\ell)$. This yields that $T_L^B=\{i\in T^B | d(i)\geq \mu_2 a(\ell)\}$ and $T_S^B=\{i\in T^B | d(i)\leq \mu_1 b(\ell)\}$ according to the definition of BUFP. We remark that either $\mu_2 a(\ell)>\mu_1 b(\ell)$ or $\mu_1 b(\ell)<1$ (in which case $T_L^B=\emptyset$); hence T_L^B and T_S^B are distinct. This way we will enforce that $T_L^B\subseteq T_L$ and $T_S^B\subseteq T_S$ for the sets T_L, T_S due to Lemma 10. We will set τ^B to be the respective value τ due to Lemma 10.

Note that in this recursive call there are $O_\epsilon(1)$ (previously selected) large tasks that use some but possibly not all edges of E_j. However, in the definition of Bonus-UFP, we have a global upper bound of τ^B for the number of allowed large tasks using an edge. To this end, we guess some further large tasks whose paths are contained in E_j, profiles for some of the small tasks, and a partition of E_j into $O_\epsilon(1)$ subpaths, such that we split this problem into subproblems in which each edge can be used by the same number $\tau^B\leq\tau$ of large tasks, so that we can call OR on the resulting subproblem. We denote by opt_S and opt_L the profit due to small and large tasks, resp., in the solution OPT from Lemma 10.

▶ **Theorem 11.** *Given a $T_{\tau,\delta}(n,u_{max})$ time $(\alpha,\beta,1)$-approximation algorithm OR for BUFP with δ-resource augmentation, for every given constant $\delta>0$. Then, for every constant $\varepsilon>0$, there exists a $T_{O_\epsilon(1),O_\epsilon(1)}(n,u_{max})\cdot(n\log u_{max})^{O_\epsilon(1)}$ time algorithm BB for UFP that computes a solution of profit at least $(1-\epsilon)\alpha\mathrm{opt}_S+(1-\epsilon)\beta\mathrm{opt}_L$.*

Theorem 9 then follows from Lemma 10 and Theorem 11, where we set $\alpha=1-\varepsilon$ and $\beta=1$.

The reader might wonder why we proved the above theorem for general α and β. The reason is that it turns out that tasks $i\in OPT_S=OPT\cap T_S$ from Lemma 10 satisfy $d(i)\leq\epsilon^2 u(e)$ for each $e\in P(i)$. Therefore, we can use LP-rounding techniques as in [11] to compute an alternative UFP solution with profit at least $(1-O(\epsilon))\mathrm{opt}_S$. Hence any algorithm for BUFP as in Theorem 11 implies a $(\frac{1+\beta-\alpha}{\beta}+O(\varepsilon))$-approximation for UFP. Recent work showed how to obtain (α,β)-approximation algorithms for UFP in the above sense with $\beta=1$ and α equal to $\frac{1}{3}$ [17]. This result becomes substantially easier to prove with resource augmentation in the BUFP setting. So we hope that Theorem 11 as stated can be a handy tool for future work along the same line.

4 Algorithm for Bonus-UFP under resource augmentation

We describe now how to adjust our algorithm for ordinary UFP with resource augmentation from Section 2 to an algorithm for Bonus-UFP with resource augmentation. Intuitively, there are two extra issues than one needs to address. First, the edges e' and e'' that partition the considered subpath P might happen to fall within the interval of some bonus. This has to be taken into account in the definition of the subproblems which are solved recursively. Second, we need to keep track of the large tasks that use each edge so that the threshold τ is not exceeded. Therefore it is convenient to *guess* them explicitly, and use the over-estimated profiles for the small tasks only.

▶ **Theorem 12.** *There is a $(1-\epsilon,1,1)$-approximation algorithm for Bonus-UFP with δ-resource augmentation with a running time of $n^{O_{\varepsilon,\delta}(\log\log n)}$.*

Theorems 9 and 12 together yield Theorem 2. It remains a challenging open problem to construct a PTAS for (Bonus-)UFP. One key bottleneck in our approach is that already in

the first iteration we need to guess a profile with up to $\Omega(\log \log n)$ many steps, and it is not clear how to do this in polynomial time.

In the remaining section, we prove Theorem 12. Again, we start with normalizing the instance. Since we have an additive resource augmentation of δb, we can use half of it in order to ensure $u_{\min} \geq \delta b/2$. Thus $2b \leq 4u_{\min}/\delta$, i.e., $u_{\mathrm{OPT}_S}(e)$ together with half of the resource augmentation has a constant capacity profile. Furthermore, analogous to Lemma 3, we can assume that each task $i \in T$ has an integer demand $d(i) \in \mathbb{N}$ and $b \in O_\delta(n)$. Unlike before, we do not lose a profit of $\epsilon w(\mathrm{OPT}_L)$.

▶ **Lemma 13.** *For arbitrary $\delta > 0$, suppose there is an (α, β, γ)-approximation algorithm for BUFP instances I with $(1+\delta)$ resource augmentation such that $u_{\min} = \delta b$, $b \in O_\delta(n)$, and $d(i) \in \mathbb{N}$ for all $i \in T$. Then there is an $((1-\epsilon)\alpha, \beta, \gamma)$-approximation algorithm for BUFP with $(1+4\delta)$ resource augmentation.*

Due to Lemma 13, in the following we assume that we are given an instance of BUFP with $(1+\delta)$ resource augmentation such that $u_{\min} = \delta b$, $2b = n/\eta$ for some $\eta \in O_\delta(1)$, and $d(i) \in \mathbb{N}$ for all $i \in T$.

We adapt the strategy used for UFP with resource augmentation and we use the same notation. Again we overestimate profiles based on the value $\theta := \lfloor u_{\min} \cdot \delta / D \rfloor$, now with $u_{\min} = \delta b$ and $D = \log n$. Lemma 4 is still valid for BUFP, if we restrict the set S to small tasks.

▶ **Lemma 14.** *There is a value $\Gamma = O_{\delta,\epsilon}(\log D) = O_{\delta,\epsilon}(\log \log n)$ such that for every edge $\bar{e} \in E$ and every set $S \subseteq T_S \cap T_{\bar{e}}$ with $d(S \cap T_e) \leq 2b$ for each $e \in E$, we have that $mp_S(\cdot)$ is a step-function with at most Γ steps.*

Proof. Since the demand $d(S)$ is bounded from above by $2b$, it is sufficient to consider the profile that for each edge $e \in E$ has the capacity $\min\{2b, u(e)\}$. With the modification, we can apply Lemma 4. ◀

Furthermore, Lemma 5 does not change for BUFP, if we restrict it to small tasks, i.e., $S_j \subseteq T_S$ for all j.

For ease of notation, we assume that for each edge f and each set $L \subseteq T_L \cap T_e$ with $d(L) \leq u(f)$, the BUFP instance to be solved has is a (dummy) bonus interval $(\{f\}, L)$. If the bonus interval is not part of the original instance, it has zero bonus, i.e., $\mathrm{bn}(\{f\}, L) = 0$. By duplicating edges, we can assume without loss of generality that there is no task i with $P(i) = f$.

4.1 Dynamic program for BUFP

In our DP table, we have a cell $(P, \hat{u}, L)$ for each subpath $P \subseteq E$, each function $\hat{u} : P \to \mathbb{N}_0$, and each set of large tasks $L \subseteq T_L$ with the following properties. We require $0 \leq \hat{u}(f) \leq n/\eta$ for each edge $f \in P$ and $\hat{u}$ is a step-function with at most 8Γ steps, i.e., one can partition P into 8Γ subpaths such that $\hat{u}$ is constant on each of them. A set L is valid, if each task in L uses the leftmost or the rightmost edge of P. Note that $|L| \leq 2\tau$ since there can only be at most τ large tasks crossing each of the two boundaries of P.

The intuitive meaning of $\hat{u}$ and L is that for each edge $e \in P$, $\hat{u}(e)$ units of capacity have already been assigned to some small tasks and the tasks L have already been selected. For a set of tasks T', we define $u_{T'} \colon E \to \mathbb{N}_0$ to be the profile with $u_T(e) = d(T' \cap T_e)$ for each $e \in E$. The goal is to compute a set of tasks with maximum total weight and with $P(i) \subseteq P$ for each selected task i, such that on each edge $e \in P$ the selected tasks use a total capacity of at most $u(e) - \hat{u}(e) - u_L(e)$.

▶ **Lemma 15.** *There are at most $n^{O_{\delta,\epsilon,\tau}(\log\log n)}$ many different DP cells.*

Proof. There are $O(n^2)$ possible choices for P. To determine the number of possible profiles $\hat{u}$, recall that $\hat{u}$ has at most $8\Gamma = O_{\delta,\epsilon}(\log\log n)$ many steps and L has a size of at most 2τ.

There are $\binom{O(n)}{O_{\delta,\epsilon,\tau}(\log\log n)} \in n^{O_{\delta,\epsilon,\tau}(\log\log n)}$ possibilities to partition P into at most $O_{\delta,\epsilon,\tau}(\log\log n)$ subpaths. For each subpath, there are at most n/η possibilities for the value of $\hat{u}$ on this subpath and $\binom{n}{2\tau} \leq n^{2\tau}$ possibilities for selecting L, i.e., $(n/\eta)^{O_{\delta,\epsilon}(\log\log n)} \cdot n^{2\tau}$ possible combinations overall. Multiplying these two numbers gives the total number of profiles. ◀

Suppose that we are given a DP cell $\mathrm{DP}(P,\hat{u},L)$. We identify two edges $e',e'' \in P$. We select $e' \in P$ such that if we restrict $\hat{u}$ to the edges of P on the left of e' or on the right of e', then $\hat{u}$ is a step-function with at most 4Γ steps. Also, we select e'' such that at most $\lfloor |P|/2 \rfloor$ edges of P lie on the left of e'', and at most $\lfloor |P|/2 \rfloor$ edges of P lie on the right of e''. Let P_1,P_2,P_3 denote the subpaths of P induced by $E_{j'}$ and $E_{j''}$, i.e., the components of $P \setminus (E_{j'} \cup E_{j''})$.

Let $\mathcal{T}$ be the set of all tuples $(p_1,p_2,\hat{u}',\hat{u}'')$ with $p_1 := (E_{j'}, L_{\mathrm{int},j'})$ and $p_2 := (E_{j''}, L_{\mathrm{int},j''})$ specified as follows. The pair $(E_{j'}, L_{\mathrm{int},j'})$ is a bonus interval with $e' \in E_{j'}$ and $(E_{j''}, L_{\mathrm{int},j''})$ is a bonus interval with $e'' \in E_{j''}$ such that $E_{j'} \cap E_{j''} = \emptyset$ or $p_1 = p_2$. A task i which overlaps with both $E_{j'}$ and $E_{j''}$ (i.e., $P(i) \cap E_{j'} \neq \emptyset$ and $P(i) \cap E_{j''} \neq \emptyset$) is either in both $L_{\mathrm{int},j'}$ and $L_{\mathrm{int},j''}$ or in none.

Each of the remaining two entries of the tuple $\hat{u}' \colon P \to \mathbb{N}_0$ and $\hat{u}'' \colon P \to \mathbb{N}_0$ is composed of two step-functions with at most Γ steps each, and on each edge $f \in P$, $\hat{u}'(f) + \hat{u}''(f) \leq (2+\delta)b$. Furthermore, the capacity of all step-functions use at most $u(f) + \delta b$ units of capacity, i.e., $\hat{u}(f) + \hat{u}'(f) + \hat{u}''(f) + u_L(f) + u_{L_{\mathrm{int},j'}}(f) + u_{L_{\mathrm{int},j''}}(f) \leq u(f) + \delta b$. Let $e'_\ell, e'_r, e''_\ell, e''_r$ be the left-most and right-most edge of $E_{j'}$ and $E_{j''}$, respectively. We apply the algorithm due to Theorem 6 to the instance whose input tasks contain all tasks in $i \in T_{e'_\ell} \cup T_{e'_r}$ with $P(i) \subseteq P$ and $P(i) \not\subseteq E_{j'}$, with edge capacities given by $\hat{u}'$. Similarly, we apply this algorithm to the instance whose input tasks contain all tasks in $i \in T_{e''_\ell} \cup T_{e''_r} \setminus (T_{e'_\ell} \cup T_{e'_r})$ with $P(i) \subseteq P$ and $P(i) \not\subseteq E_{j''}$, with edge capacities given by $\hat{u}''$. Denote by $\mathrm{alg}(\hat{u}')$ and $\mathrm{alg}(\hat{u}'')$ the respective solutions. With the tuple $(p_1,p_2,\hat{u}',\hat{u}'')$ we associate the solution given by $\mathrm{alg}(\hat{u}') \cup \mathrm{alg}(\hat{u}'')$, the bonuses $\mathrm{bn}(p_1), \mathrm{bn}(p_2)$, the profit from large tasks $w(L_{\mathrm{int},j'} \cup L_{\mathrm{int},j''} \setminus L)$, and the solutions stored in the cells $DP(P_j, (\hat{u} + \hat{u}' + \hat{u}'')_{|P_j}, L')$ for $j \in \{1,2,3\}$ and L' the tasks from $L \cup L_{\mathrm{int},j'} \cup L_{\mathrm{int},j''}$ crossing P_j. We store in $\mathrm{DP}(P,\hat{u},L)$ the weight of the most profitable solution for all tuples $(p_1,p_2,\hat{u}',\hat{u}'') \in \mathcal{T}$, i.e.,

$$\mathrm{DP}(P,\hat{u},L) := \max_{t=(p_1,p_2,\hat{u}'_S,\hat{u}''_S)\in\mathcal{T}} \big(w(\mathrm{alg}(\hat{u}')) + w(\mathrm{alg}(\hat{u}'')) + \sum_{p\in\{p_1,p_2\}} \mathrm{bn}(p) + w(L_{\mathrm{int},j'} \cup L_{\mathrm{int},j''} \setminus L) + \sum_{j=1}^{3} \mathrm{DP}(P_j, (\hat{u} + \hat{u}' + \hat{u}'')_{|P_j}) \big),$$

where $\hat{u}'$ and $\hat{u}''$ are derived from t as described before. Observe that if $p_1 = p_2$, we collect only one bonus. We have to ensure that $\hat{u}^{(j)} := (\hat{u} + \hat{u}' + \hat{u}'')_{|P_j}$ is a valid profile. Due to the halving, restricted to P_j the profile $\hat{u}$ has at most 4τ steps and each of the other two profiles has at most 2Γ steps from small tasks, which results in 8Γ steps in total. Independently, we always have a total number of at most 4τ steps from large tasks

At the end we output $\mathrm{DP}(E,u_0)$ where u_0 is the profile with $u_0(e) = 0$ for all $e \in E$. By standard memoization we can also output the solution associated with $\mathrm{DP}(E,u_0)$.

We need to show that the DP computes a near-optimal solution. It is clear that we always output a feasible solution since in the computation for each cell, the set $\mathcal{T}$ contains

only tuples $(\hat{u}'_S, \hat{u}''_S, p_1, p_2)$ such that $\hat{u}(f) + \hat{u}'(f) + \hat{u}''(f) + u_{L \cup L_{\text{int},j'} \cup L_{\text{int},j''}} \leq u(f) + \delta b$ for each edge $f \in P$. Intuitively, Lemma 5 allows us to argue that in each step we can guess for $\hat{u}'$ and $\hat{u}''$, the pairs p_1 and p_2, the profiles $\text{mp}_{\text{OPT}' \cap T_{e'}}$ and $\text{mp}_{(\text{OPT}' \cap T_{e''}) \setminus T_{e'}}$, respectively, where OPT' denotes the tasks $i \in \text{OPT}$ with $P(i) \subseteq P$, without violating the capacities of the edges. Let OPT_B be the set of bonus pairs of an optimal solution and $w(\text{OPT}_B)$ the sum of bonuses.

▶ **Lemma 16.** *The dynamic program computes a solution* ALG *which is feasible with $O(\delta)$-resource augmentation and $w(\text{ALG}) \geq (1-\epsilon)w(\text{OPT}_S) + w(\text{OPT}_L) + w(\text{OPT}_B)$.*

Proof. Initially, the DP chooses the middle edge e (i.e., $e = e' = e''$) and a bonus interval containing e. One of the choices is the bonus pair $p = (E_j, L_{\text{int},j})$ such that $p \in \text{OPT}_B$ and $e \in E_j$. Since OPT is feasible, it has no tasks contained in E_j. Let e_ℓ and e_r be the left-most and right-most edge of E_j. Then one of the options for the profile is to choose $\hat{u} := \text{mp}_{\text{OPT} \cap T_{e_\ell}} + \text{mp}_{\text{OPT} \cap T_{e_r}}$.

All tasks $\text{OPT} \cap (T_{e_\ell} \cup T_{e_r})$ fit into the profile $\hat{u}$. Since the DP approximates the profit from these tasks, the value of the DP cell without the values from the subproblems is at least $(1-\epsilon)w(\text{OPT}_S \cap T_e) + w(\text{OPT}_L \cap (T_{e_\ell} \cup T_{e_r})) + \text{bn}(p)$. The instance is split into the left hand side of E_j and the right hand side of E_j. Let $(P, \hat{u}, L)$ be a sub-problem reached (recursively) by the described choice of profile. Each edge e' chosen subsequently within P has the property that the DP can choose the bonus pair $p_1 := (E_{j'}, L_{\text{int},j'})$ (where as before we assume that e'_ℓ and e'_r are the leftmost and rightmost edges of $E_{j'}$, respectively) from OPT_B containing e' and a feasible choice of $\hat{u}'$ is $\text{mp}_{\text{OPT}_S \cap \{i \in T_{e'_\ell} \cup T_{e'_r} \mid P(i) \subseteq P\}}$. Each edge e'' chosen subsequently within P has the property that the DP can choose the bonus pair $p_2 := (E_{j''}, L_{\text{int},j''})$ (where as before we assume that e''_ℓ and e''_r are the leftmost and rightmost edges of $E_{j''}$, respectively) from OPT_B containing e'' and a feasible choice of $\hat{u}'$ is $\text{mp}_{\text{OPT}_S \cap \{i \in T_{e''_\ell} \cup T_{e''_r} \setminus (T_{e'_\ell} \cup T_{e'_r}) \mid P(i) \subseteq P\}}$. Observe that all tasks $\{i \in \text{OPT}_S \cap (T_{e'_\ell} \cup T_{e'_r} \cup T_{e''_\ell} \cup T_{e''_r}) \mid P(i) \subset P\}$ fit into the combined profiles, we select all tasks specified in p_1 and p_2, and all remaining tasks from OPT within P will be available in sub-problems. The DP collects a profit of at least $(1-\epsilon)w(\{i \in \text{OPT}_S \cap (T_{e'_\ell} \cup T_{e''_r} \cup T_{e''_\ell} \cup T_{e''_r}) \mid P(i) \subseteq P\}) + w(\{i \in \text{OPT}_L \cap (T_{e'_\ell} \cup T_{e''_r} \cup T_{e''_\ell} \cup T_{e''_r}) \setminus L \mid P(i) \subseteq P\}) + \text{bn}(p_1) + \text{bn}(p_2)$. We inductively conclude that if all choices as described above are feasible, the DP obtains a profit of at least $(1-\epsilon)w(\text{OPT}_S) + w(\text{OPT}_L) + w(\text{OPT}_B)$.

To show feasibility, we have to argue that none of the described choices exceeds the overall capacity of $u(f) + \delta b$ for an edge $f \in E$. The argument is analogous to the proof of Lemma 8. ◀

Together with Lemma 13 we conclude that we obtain a $(1-\epsilon, 1, 1)$-approximation algorithm for BUFP with δ-resource augmentation.

References

1 Anna Adamaszek, Parinya Chalermsook, Alina Ene, and Andreas Wiese. Submodular unsplittable flow on trees. *Math. Program.*, 172(1-2):565–589, 2018. doi:10.1007/s10107-017-1218-4.

2 Aris Anagnostopoulos, Fabrizio Grandoni, Stefano Leonardi, and Andreas Wiese. Constant integrality gap LP formulations of unsplittable flow on a path. In *IPCO*, pages 25–36, 2013. doi:10.1007/978-3-642-36694-9_3.

3 Aris Anagnostopoulos, Fabrizio Grandoni, Stefano Leonardi, and Andreas Wiese. A mazing 2+ϵ approximation for unsplittable flow on a path. *ACM Trans. Algorithms*, 14(4):55:1–55:23, 2018. doi:10.1145/3242769.

4 Y. Azar and O. Regev. Combinatorial algorithms for the unsplittable flow problem. *Algorithmica*, 44(1):49–66, 2006. doi:10.1007/s00453-005-1172-z.

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.

7 Jatin Batra, Naveen Garg, Amit Kumar, Tobias Mömke, and Andreas Wiese. New approximation schemes for unsplittable flow on a path. In *SODA*, pages 47–58, 2015. URL: http://epubs.siam.org/doi/abs/10.1137/1.9781611973730.5, doi:10.1137/1.9781611973730.5.

8 Paul Bonsma, Jens Schulz, and Andreas Wiese. A constant-factor approximation algorithm for unsplittable flow on paths. *SIAM Journal on Computing*, 43:767–799, 2014.

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

10 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.

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

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

13 Julia Chuzhoy and Alina Ene. On approximating maximum independent set of rectangles. In *IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS 2016)*, pages 820–829. IEEE, 2016.

14 Fabrizio Grandoni, Salvatore Ingala, and Sumedha Uniyal. Improved approximation algorithms for unsplittable flow on a path with time windows. In *WAOA*, pages 13–24, 2015.

15 Fabrizio Grandoni, Tobias Mömke, and Andreas Wiese. Unsplittable flow on a path: The game! 2020. unpublished.

16 Fabrizio Grandoni, Tobias Mömke, Andreas Wiese, and Hang Zhou. To augment or not to augment: Solving unsplittable flow on a path by creating slack. In *SODA*, pages 2411–2422, 2017.

17 Fabrizio Grandoni, Tobias Mömke, Andreas Wiese, and Hang Zhou. A $(5/3 + \epsilon)$-approximation for unsplittable flow on a path: placing small tasks into boxes. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 607–619, 2018. doi:10.1145/3188745.3188894.

18 Shi Li. Towards PTAS for precedence constrained scheduling via combinatorial algorithms. In *SODA*, pages 2991–3010. SIAM, 2021.

19 Andreas Wiese. A (1+epsilon)-approximation for unsplittable flow on a path in fixed-parameter running time. In *ICALP 2017*, volume 80 of *LIPIcs*, pages 67:1–67:13, 2017. doi:10.4230/LIPIcs.ICALP.2017.67.