Unsplittable Flow on a Short Path

Ilan Doron-Arad
Computer Science Department, Technion, Haifa, Israel

Fabrizio Grandoni
IDSIA, USI-SUPSI, Lugano, Switzerland

Ariel Kulik
Computer Science Department, Technion, Haifa, Israel

Abstract

In the Unsplittable Flow on a Path problem (UFP), we are given a path graph with edge capacities and a collection of tasks. Each task is characterized by a demand, a profit, and a subpath. Our goal is to select a maximum profit subset of tasks such that the total demand of the selected tasks that use each edge e is at most the capacity of e. BagUFP is the generalization of UFP where tasks are partitioned into bags, and we are allowed to select at most one task per bag. UFP admits a PTAS [Grandoni,Mömke,Wiese'22] but not an EPTAS [Wiese'17]. BagUFP is APX-hard [Spieksma'99] and the current best approximation is $O(\log n/\log\log n)$ [Grandoni,Ingala,Uniyal'15], where n is the number of tasks.

In this paper, we study the mentioned two problems when parameterized by the number m of edges in the graph, with the goal of designing faster parameterized approximation algorithms. We present a parameterized EPTAS for BagUFP, and a substantially faster parameterized EPTAS for UFP (which is an FPTAS for $m = O(1)$). We also show that a parameterized FPTAS for UFP (hence for BagUFP) does not exist, therefore our results are qualitatively tight.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Knapsack, Approximation Schemes, Parameterized Approximations

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.5

Related Version *Full Version*: https://arxiv.org/abs/2407.10138 [23]

Funding *Fabrizio Grandoni*: Partially supported by the SNSF Grant 200021_2000731/1.
Ariel Kulik: This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 852780-ERC (SUBMODULAR).

1 Introduction

In the classical Usplittable Flow on a Path problem (UFP) we are given an m-edge path graph $G = (V, E)$ with (non-negative integer) edge capacities $u : E \to \mathbb{N}$, and a collection of n tasks T. Each task i is characterized by a demand $d(i) \in \mathbb{N}$, a weight (or profit) $w(i) \in \mathbb{N}$, and a subpath $P(i)$[1]. A feasible solution consists of a subset of (selected) tasks $S \subseteq T$ such that, for each edge e, $\sum_{i \in S : e \in P(i)} d(i) \leq u(e)$. In other words, the total demand of the selected tasks using each edge e cannot exceed the capacity of e. Our goal is to compute a feasible solution OPT of maximum total profit $\text{opt} = w(\text{OPT}) := \sum_{i \in \text{OPT}} w(i)$.

UPF has several direct and undirect applications [7, 8, 11, 12, 16, 17, 18, 21, 41, 43]. For example, one might interpret G as a time interval subdivided into time slots (the edges). At each time slot we are given some amount of a considered resource, say, energy. The tasks

[1] Throughtout this paper, for a subpath P, we sometimes use P also to denote the corresponding set of edges $E(P)$: the meaning will be clear from the context.

represent jobs that we might execute, therefore gaining a profit. However each executed job will consume some amount of the shared resource during its execution, thus we might not be able to execute all the jobs (hence we need to perform a selection).

UFP is strongly NP-hard [11, 18] and it is well-studied in terms of approximation algorithms. After a long sequence of improvements [2, 3, 5, 6, 9, 11, 14, 15, 34, 37, 38, 36], a PTAS for UFP was eventually achieved by Grandoni, Mömke and Wiese [35]. We recall that a PTAS (for a maximization problem) is an algorithm parameterized by $\varepsilon > 0$, which provides a $(1-\varepsilon)$ approximation in time $|I|^{O_\varepsilon(1)}$, where $|I|$ is the input size. EPTASs and FPTASs are defined similarly, however with running times of the form $f(1/\varepsilon)\cdot|I|^{O(1)}$ and $(|I|/\varepsilon)^{O(1)}$, resp., where $f(\cdot)$ is a computable function. Wiese [49] proved that UFP, parameterized by the number of selected tasks, is $W[1]$-hard: this excludes the existence of an EPTAS for UFP by standard reductions (unless FPT=W[1] [42]).

In the above scenario there is no flexibility on the time when a job is executed. The BagUFP problem is a generalization of UFP which was introduced to allow for such flexibility. Here we are given the same input as UFP, plus a partition of the tasks into ℓ bags $\mathcal{B} = \{B_1, \dots, B_\ell\}$, $\dot\bigcup_{j=1}^{\ell} B_j = T$. A feasible solution S has to satisfy the capacity constraints as in UFP, plus the extra constraint that at most one task per bag can be selected, namely $|S \cap B_j| \leq 1$ for $j = 1, \dots, \ell$. This easily captures jobs that can be executed at different times (and even more general settings). For example, if a job can be executed within a given time window (also known as the time-windows UFP problem), it is sufficient to create a bag that contains multiple copies of the same task which differ only in the subpath $P(i)$ (with one subpath per potential valid scheduling time). BagUFP is APX-hard [47], which rules out the existence of a PTAS for it. The current best approximation ratio for BagUFP is $O(\log n/\log\log n)$ [33], slightly improving on the $O(\log n)$ approximation in [13]. A constant approximation for BagUFP is known for the cardinality version of the problem [33], i.e. when all the profits are 1. Bag constraints are frequently added to other classic optimization problems, such as makespan minimization [22, 32], knapsack [46, 44, 40], and bin packing [24, 25, 31].

1.1 Our Results and Techniques

The mentioned PTAS for UFP [35] has a very poor dependence on ε in the running time, which makes it most likely impractical. Though an improvement of the running time is certainly possible, as mentioned before an EPTAS for UFP does not exist (unless FPT = W[1]). The situation for BagUFP is even worse: here even a PTAS does not exist (unless P = NP), and currently finding a constant approximation algorithm (which might exist) is a challenging open problem.

Motivated by the above situation, it makes sense to consider parameterized approximation algorithms for UFP and BagUFP. The general goal here is to identify some integer parameter p that captures some relevant aspect of the input (or some property of the output), and try to design approximation algorithms whose running time is better than the state of the art when p is sufficiently small. In particular a parameterized PTAS (p-PTAS) is defined similarly to a PTAS, however with running time of the form $f(p)|I|^{O_\varepsilon(1)}$ for some commutable function $f(\cdot)$. Parameterized EPTAS (p-EPTAS) and parameterized FPTAS (p-FPTAS) are defined similarly, w.r.t. EPTAS and FPTAS resp. More explicitly, a p-EPTAS has a running time of the form $f(p+1/\varepsilon)\,|I|^{O(1)}$, while a p-FPTAS has a running time of the form $f(p)(|I|/\varepsilon)^{O(1)}$. For a meaningful choice of p, it makes sense to search for a p-EPTAS (or better) for UFP, and for a p-PTAS (or better) for BagUFP.

Probably the most standard parameter is the number k of tasks in the desired solution. This is also the objective function for the cardinality version of the problems (with profits equal to 1). Wiese [49] proved that UFP is $W[1]$-hard under this parametrization, which rules out a p-EPTAS. He also presented a p-PTAS for the cardinality version of UFP with parameter k (later improved by the PTAS in [35], which also works for arbitrary profits). To the best of our knowledge, the same parametrization of BagUFP was not studied in the literature.

In this paper we focus on the parameter m, namely the number of edges in G - the length of the path. This makes sense in the realistic scenarios where $n \gg m$ i.e., there are significantly more jobs than time slots. For example, such UFP instances occur in personnel scheduling [48, 4, 10, 1] where, e.g., workers are assigned to shifts within a working day ($m \approx 8$ working hours), or for an interval of days in the week ($m = 7$ days). We achieve the following main results:

1.1.1 Algorithms and Hardness for BagUFP

A simple reduction from Partition shows that (assuming $\mathrm{P} \neq \mathrm{NP}$) there is no FPTAS for BagUFP even for $m = 2$ (for $m = 1$ an FPTAS exists since the problem is equivalent to Multiple Choice Knapsack). As an obvious corollary, there is no p-FPTAS with parameter m for the same problem (see Section 4).

▶ **Theorem 1.** *Unless* $\mathrm{P} = \mathrm{NP}$*, there is no* FPTAS *for* BagUFP *even in the case* $m = 2$*.*

▶ **Corollary 2.** *Unless* $\mathrm{P} = \mathrm{NP}$*, there is no* p-FPTAS *for* BagUFP *parametrized by the path length* m*.*

Hence, qualitatively speaking, the best one can hope for is a p-EPTAS. This is precisely what we achieve (see Section 2).

▶ **Theorem 3.** *There is a* p-EPTAS *for* BagUFP *parametrized by the path length* m*. Its running time is* $2^{\left(m/\varepsilon^{1/\varepsilon}\right)^{O(1)}} \cdot |I|^{O(1)}$.

Our approach substantially differs from previous algorithmic approaches for UFP (see, e.g., [35] and references therein) which relied on concepts such as classification of items by *demands* and probabilistic arguments. We observe that the bag constraints induce a matroid (more specifically, a partition matroid with capacity 1 for each set). Therefore we consider the standard LP relaxation for a partition matroid (which has integral basic solutions), and augment it with the m linear constraints corresponding to the capacity constraints. As proved in [39], a basic optimal solution x^* to this LP (which can be computed in polynomial time for arbitrary m) has at most $2m$ fractional values (with value strictly between 0 and 1). The variables with value 1 in x^* induce a feasible BagUFP solution with profit at least the optimal LP profit minus (almost) the profit of $2m$ tasks: this is problematic if the latter tasks have a profit comparable to $\mathrm{opt} = w(\mathrm{OPT})$, where OPT is some reference optimal solution.

We can avoid the above issue as follows. Let H be the (*heavy*) tasks with profit at least $\frac{\varepsilon}{m}\mathrm{opt}$. We can guess the heavy tasks $H \cap \mathrm{OPT}$ in OPT (which are at most m/ε many), reduce the problem (i.e., remove all the tasks in the bags containing tasks from $H \cap \mathrm{OPT}$, remove tasks in H, and reduce the available capacity of every edge by the total demand of $\mathrm{OPT} \cap H$ for the specific edge), and apply the mentioned LP-rounding technique to the remaining (*light*) tasks. Now the drop of the fractional variables reduces the profit by at most $2\varepsilon \cdot \mathrm{opt}$, leading to a $1 + O(\varepsilon)$ approximation. Unfortunately, this algorithm would take time $|H|^{\Omega(m/\varepsilon)}$, which is still not compatible with a p-EPTAS.

In order to circumvent the latter issue, we exploit the notion of representative sets, which was introduced in [26, 27, 28] to deal with a class of maximization problems with a single budget constraint. In contrast, we construct a representative set in the more general regime of multiple budget constraints imposed by the unsplittable flow setting. In more detail, in p-EPTAS time, we are able to compute a (representative) subset of tasks R of size depending only on m and $1/\varepsilon$, such that there exists a nearly optimal solution S such that $S \cap H \subseteq R$. Therefore, one can restrict to R the above guessing of heavy tasks, which takes $|R|^{O(m/\varepsilon)}$ time: this is now compatible with a p-EPTAS. We remark that our techniques, combined with the representative set techniques of [26, 27, 28], can give a p-EPTAS for the more general problem of UFP with a general matroid constraint. We leave such efforts to the journal version of the paper. On the other hand, UFP with a general matroid is somewhat harder since an FPTAS is ruled out even for an instance with path of length 1 (a single budget constraint) [30], and an FPTAS exists only for laminar matroids [29].

1.1.2 Algorithms and Hardness for UFP

We start by showing that there is no p-FPTAS for UFP parameterized by m. This, together with Theorem 3, gives a tight bound for UFP in the short path point-of-view. Notice that this is not implied by Theorem 1 since UFP is a special case of BagUFP.

▶ **Theorem 4.** *Unless* FPT=W[1]*, there is no* p-FPTAS *for* UFP *parametrized by the path length m.*

Unlike previous hardness results [11, 18, 47, 49, 19] for UFP and its variant, which rely on a path of polynomial length in the input size, our lower bound requires having UFP instances with a *short* path. Namely, the number of tasks is significantly larger than the length of the path. Our starting point is to obtain a hardness result for a *multiple choice* variant of k-subset sum in which the numbers are partitioned into sets $A_1, \dots, A_k$, each set with n numbers, and the goal is to select one number from each set such their sum is exactly a given target value. We use color-coding to show that multiple-choice k-subset sum does not have an FPT-algorithm unless W[1]=FPT (which may be useful for other hardness results). Then, we reduce multiple-choice k-subset sum to UFP by constructing a UFP instance with $m = O(k)$ edges and with polynomial weights. Roughly, we interpret the edges of the path in correspondence to the k sets $A_1, \dots, A_k$. The constructed instance has a pair of tasks z_j^i, q_j^i, with complementary subpaths, for every number $j = 1, \dots, n$ in the i-th set A_i. Along with a carefully defined demand and weight functions, This UFP instance satisfies that exactly k pairs can be chosen for a sufficiently high weight if and only if the original subset sum instance has a solution. We remark that this construction utilizes the short path in a non-trivial manner. Due to space constraints, the proof is given in the full version of the paper [23].

Theorem 3 already provides a p-EPTAS for UFP. We are however able to derive a p-EPTAS with a substantially better running time (see Section 3).

▶ **Theorem 5.** *There is an* p-EPTAS *for* UFP *parameterized by the path length m, with running time $O\left(\frac{n^3}{\varepsilon} + \left(\frac{1}{\varepsilon}\right)^{O(m^2)} m^3 \log n\right)$.*

In particular, for $m \leq C \cdot \sqrt{\log_{\frac{1}{\varepsilon}} n}$, for a sufficiently small constant $C > 0$, our running time is the running time of an FPTAS. We recall that achieving an FPTAS (or even an EPTAS) for UFP in general is not possible and the previous state of the art for UFP with a constant number of edges is the PTAS for the general problem [35].

The basic idea of the algorithm is a follows. Consider all the tasks T_φ whose path is φ. Let opt_φ be the profit of some optimal solution OPT restricted to T_φ, i.e. $\text{opt}_\varphi = w(\text{OPT} \cap T_\varphi)$. Given the value of opt_φ, it is sufficient to find a minimum-demand subset of tasks $S_\varphi \subseteq T_\varphi$

with profit at least opt_φ: the union of the sets S_φ would be feasible and optimal. To achieve the target running time we use this basic idea along with rounding of the weights and a coarse guessing of the the values opt_φ. By a standard rounding argument, we can assume that the weights are in $[\frac{n}{\varepsilon}]$ while loosing a factor $1-\varepsilon$ in the approximation. This allows us to pre-compute the minimal demand subset of T_φ which attains a threshold rounded weight, for every possible threshold, using a standard dynamic program. The pre-computed subsets are used to reconstruct a solution S_φ from the value of opt_φ. Finally, we guess the values of opt_φ up to an additive error of $\approx \frac{\varepsilon}{m^2} \cdot w(\text{OPT})$. This coarse guess of the values of opt_φ allows us to enumerate over all possible guesses within the running time, while only introducing an additional $1-\varepsilon$ factor in the approximation.

1.2 Preliminaries

For every $n \in \mathbb{N}$ we use $[n] = \{1, \ldots, n\}$. We use $(G, u, T, P, d, w, \mathcal{B})$ to denote a BagUFP instance and by (G, u, T, P, d, w) to denote a UFP instance. Given and instance I of UFP or BagUFP, we let $\text{OPT}(I)$ denote some reference optimal solution, and $\text{opt}(I) = w(\text{OPT}(I))$ be its profit. We use $|I|$ to denote the encoding size of I. When I is clear from the context, we simply use OPT and opt, resp. Given a subset of tasks $S \subseteq T$, we use the standard notation $d(S) := \sum_{i \in S} d(i)$ and $w(S) := \sum_{i \in S} w(i)$.

2 A p-EPTAS for BagUFP

In this section we prove Theorem 3. For the remaining of this section, fix a instance I of BagUFP and an error parameter $0 < \varepsilon < \frac{1}{2}$. Let the set of *heavy* tasks in I be

$$H = \left\{ e \in E \mid w(e) > \frac{\varepsilon \cdot \text{opt}}{m} \right\}.$$

The remaining tasks $T \setminus H$ are *light*. Our first goal is to find the set of heavy tasks in a nearly-optimal solution. Notice that a naive enumeration takes $n^{\Omega(\frac{m}{\varepsilon})}$ time, which is far from the running time of a p-EPTAS. To avoid this issue, we compute a (small enough) representative set, which is defined as follows.

▶ **Definition 6.** *For some $R \subseteq T$, we say that R is an ε-*representative set *of I if there is a solution $S$of I such that the following holds.*
1. *$S \cap H \subseteq R$.*
2. *$w(S) \geq (1-3\varepsilon) \cdot \text{opt}$.*

Define $q(\varepsilon, m) = \left\lceil 4m \cdot \varepsilon^{-\lceil \varepsilon^{-1} \rceil} \right\rceil$ (the meaning of $q(\varepsilon, m)$ becomes clear in Section 2.2).

▶ **Lemma 7.** *There is an algorithm* RepSet *that, given a* BagUFP *instance instance $I = (G, u, T, P, d, w, \mathcal{B})$, $0 < \varepsilon < \frac{1}{2}$, and $\tilde{\text{opt}} \in [w(T)]$, in time $m^3 \cdot \varepsilon^{-2} \cdot |I|^{O(1)}$ returns $R \subseteq T$ with $|R| \leq 3 \cdot m^3 \cdot \varepsilon^{-2} \cdot q(\varepsilon, m)$. Furthermore, if $\frac{\text{opt}}{2} < \tilde{\text{opt}} \leq \text{opt}$, R is an ε-representative set of I.*

In Section 2.1 we use the representative set from Lemma 7 to design a p-EPTAS for BagUFP. Then, in Section 2.2 we prove Lemma 7.

Algorithm 1 p-EPTAS(I, ε).

```
input  : BagUFP instance $I$ and an error parameter $0 < \varepsilon < \frac{1}{2}$.
output : A $(1 - 7\varepsilon)$-approximate solution $A$ for $I$.
1  $A \leftarrow \emptyset$.
2  for $\tilde{\text{opt}} \in \left\{1, 2, \ldots, 2^{\lfloor \log_2(w(T)) \rfloor}\right\}$ do
3      Construct $R^{\tilde{\text{opt}}} \leftarrow$ RepSet$(I, \varepsilon, \tilde{\text{opt}})$.
4      for $F \subseteq R^{\tilde{\text{opt}}}$ s.t. $|F| \leq m/\varepsilon$ and $F$ is a feasible solution for $I$ do
5          Compute a basic optimal solution $\lambda^{\tilde{\text{opt}},F}$ for LP$_F^{\tilde{\text{opt}}}$.
6          Define $L_F^{\tilde{\text{opt}}} := \left\{i \in T_F^{\tilde{\text{opt}}} \mid \lambda_i^{\tilde{\text{opt}},F} = 1\right\}$ and $A_F^{\tilde{\text{opt}}} = L_F^{\tilde{\text{opt}}} \cup F$.
7          if $w\left(A_F^{\tilde{\text{opt}}}\right) > w(A)$ then
8              $A \leftarrow A_F^{\tilde{\text{opt}}}$.
9          end
10     end
11 end
12 Return $A$.
```

2.1 A Representative Set Based p-EPTAS

Given the representative set algorithm described in Lemma 7, we obtain a p-EPTAS as follows (the pseudocode is given in Algorithm 1). We consider the powers of two $\tilde{\text{opt}}$ in the domain $[w(T)]$ (i.e., all values $\tilde{\text{opt}} = 1, 2, 4, \ldots, 2^{\lfloor \log w(T) \rfloor}$). We apply the algorithm from Lemma 7 with this parameter $\tilde{\text{opt}}$ to obtain a set $R^{\tilde{opt}}$. Notice that, for $\frac{\text{opt}}{2} < \tilde{\text{opt}} \leq \text{opt}$, $R^{\tilde{\text{opt}}}$ is a representative set. Now we enumerate over all the feasible solutions $F \subseteq R^{\tilde{opt}}$ of cardinality at most m/ε. For each such F, we compute a feasible solution $A_F^{\tilde{\text{opt}}}$ (including F), and return the best such solution.

It remains to describe how $A_F^{\tilde{\text{opt}}}$ is computed. First of all, we define a reduced BagUFP instance $I_F^{\tilde{\text{opt}}} = (G, u_F, T_F^{\tilde{\text{opt}}}, P, d, w, \mathcal{B}_F)$ as follows. $\mathcal{B}_F$ is the subset of input bags not containing any task in F. The set of tasks $T_F^{\tilde{\text{opt}}}$ is given by the tasks of weight at most $\frac{2\varepsilon}{m}\tilde{\text{opt}}$ which are contained in the bags $\mathcal{B}_F$. The capacity function u_F is given by $u_F(e) := u(e) - \sum_{i \in F : e \in P(i)} d(i)$ (i.e., the residual capacity after accommodating the tasks in F). Observe that, for any feasible solution L for $I_F^{\tilde{\text{opt}}}$, $L \cup F$ is a feasible solution for the input problem. Indeed, the capacity constraints are satisfied and at most one task per bag can be selected.

Given the above instance $I_F^{\tilde{\text{opt}}}$, we considering the following LP relaxation $LP_F^{\tilde{\text{opt}}}$:

$$
\begin{aligned}
\max \quad & \sum_{i \in T_F^{\tilde{\text{opt}}}} x_i \cdot w(i) & & (LP_F^{\tilde{\text{opt}}}) \\
\text{s.t.} \quad & \sum_{i \in T_F^{\tilde{\text{opt}}} : e \in P(i)} x_i \cdot d(i) \leq u_F(e) & & \forall e \in E \\
& \sum_{i \in T_F^{\tilde{\text{opt}}} \cap B} x_i \leq 1 & & \forall B \in \mathcal{B}_F \\
& x_i \geq 0 & & \forall i \in T_F^{\tilde{\text{opt}}}
\end{aligned}
$$

We compute a basic optimal solution $\lambda^{\tilde{\text{opt}},F}$ for the above LP. Let $L^{\tilde{\text{opt}}}_F \subseteq T^{\tilde{\text{opt}}}_F$ be the tasks such that $\lambda^{\tilde{\text{opt}},F}_i = 1$. We set $A^{\tilde{\text{opt}}}_F = L^{\tilde{\text{opt}}}_F \cup F$. This concludes the description of the algorithm.

Obviously the above algorithm computes a feasible solution.

▶ **Lemma 8.** *Algorithm 1 returns a feasible solution.*

Proof. Consider a given pair $(\tilde{\text{opt}}, F)$. Obviously $L^{\tilde{\text{opt}}}_F$ is a feasible solution for the BagUFP instance $I^{\tilde{\text{opt}}}_F$. Indeed, the demand of the tasks in $L^{\tilde{\text{opt}}}_F$ whose path contains a given edge e is upper bounded by $\sum_{i\in T^{\tilde{\text{opt}}}_F : e\in P(i)} \lambda^{\tilde{\text{opt}},F}_i \cdot d(i) \leq u_F(e)$. Furthermore, for a given bag $B \in \mathcal{B}_F$, at most one variable $\lambda^{\tilde{\text{opt}},F}_i$ with $i \in B$ can be equal to 1, hence $|L^{\tilde{\text{opt}}}_F \cap B| \leq 1$. Thus, as argued before, $A^{\tilde{\text{opt}}}_F = L^{\tilde{\text{opt}}}_F \cup F$ is a feasible solution for the input BagUFP instance I. Since the returned solution A is one of the feasible solutions $A^{\tilde{\text{opt}}}_F$ (or the empty set, which is a feasible solution), A is a feasible solution. ◀

It is also not hard to upper bound the running time.

▶ **Lemma 9.** *Algorithm 1 runs in time* $\left(\frac{3\cdot m^3}{\varepsilon^2} \cdot q(\varepsilon, m)\right)^{m/\varepsilon} |I|^{O(1)}$.

Proof. Lines 3 and 5-8 can be performed in $|I|^{O(1)}$ time. Thus the overall running time is upper bounded by $|I|^{O(1)}$ multiplied by the number of possible pairs $(\tilde{\text{opt}}, F)$. There are $O(\log w(T)) = |I|^{O(1)}$ possible choices for $\tilde{\text{opt}}$. For a fixed choice of $\tilde{\text{opt}}$, one has $|R^{\tilde{\text{opt}}}| \leq \frac{3m^3}{\varepsilon^2} q(\varepsilon, m)$. Since F is a subset of $R^{\tilde{\text{opt}}}$ of cardinality at most m/ε, the number of possible choices for F (for the considered $\tilde{\text{opt}}$) is at most $2\left(\frac{3m^3}{\varepsilon^2} q(\varepsilon, m)\right)^{m/\varepsilon}$. The claim follows. ◀

It remains to bound the approximation factor of the algorithm. To this aim, we critically exploit the fact that each basic solution $\lambda^{\tilde{\text{opt}},F}$ is almost integral: more precisely, it has at most $2m$ non-integral entries. To prove that, we use a result in [39] about the sparseness of matroid polytopes with m additional linear constraints.

▶ **Lemma 10.** *Each solution $\lambda^{\tilde{\text{opt}},F}$ computed by Algorithm 1 has at most $2m$ non-integral entries.*

Proof. The proof relies on matroid theory; for more details on the subject, we refer the reader to, e.g., [45]. Consider $LP^{\tilde{\text{opt}}}_F$ for any pair $(\tilde{\text{opt}}, F)$ considered by the algorithm. Let $\tilde{LP}^{\tilde{\text{opt}}}_F$ be the LP obtained from $LP^{\tilde{\text{opt}}}_F$ by dropping the m capacity constraints $\sum_{i\in T^{\tilde{\text{opt}}}_F : e\in P(i)} x_i \cdot d(i) \leq u_F(e)$. $\tilde{LP}^{\tilde{\text{opt}}}_F$ turns out to be the standard LP for a partition matroid (in particular, in an independent set at most one task per bag can be selected, where the bags induce a partition of the tasks). In [39] it is shown that every basic solution (including an optimal one) for an LP obtained by adding m linear constraints to the standard LP for any matroid (including partition ones) has at most $2m$ non-integral entries. Hence $\lambda^{\tilde{\text{opt}},F}$ satisfies this property. ◀

▶ **Lemma 11.** *The solution A returned by Algorithm 1 satisfies $w(A) \geq (1 - 7\varepsilon)\text{opt}$.*

Proof. It is sufficient to show that some solution $C^{\tilde{\text{opt}}}_F$ has large enough profit. Consider the value of $\tilde{\text{opt}}$ such that $\frac{\text{opt}}{2} < \tilde{\text{opt}} \leq \text{opt}$. Notice that the algorithm considers exactly one such value since $1 \leq \text{opt} \leq w(T)$. We next show how to choose a convenient $F \subseteq R^{\tilde{\text{opt}}}$.

Observe that for the considered choice of $\tilde{\text{opt}}$, $R^{\tilde{\text{opt}}}$ is an ε-representative set. Let S be the solution for I guaranteed by Lemma 7 and Definition 6. Recall that $w(S) \geq (1-3\varepsilon)\text{opt}$ and $S \cap H \subseteq R^{\tilde{\text{opt}}}$. Since each $i \in S \cap H$ has $w(i) \geq \frac{\varepsilon}{m}\text{opt}$ by definition and since obviously $w(S \cap H) \leq w(S) \leq \text{opt}$, it must be the case that $|S \cap H| \leq \frac{m}{\varepsilon}$. This implies that there is an iteration of the algorithm (for the considered $\tilde{\text{opt}}$) that has $F = S \cap H$: we will focus on that iteration.

We claim that $w(A^{\tilde{\text{opt}}}_{S\cap H}) = w(S \cap H) + w(L^{\tilde{\text{opt}}}_{F}) \geq (1-7\varepsilon)\text{opt}$. Notice that each task $i \in S \setminus H$ has weight $w(i) < \frac{\varepsilon}{m}\text{opt} < \frac{2\varepsilon}{m}\tilde{\text{opt}}$. Furthermore, by construction $i \in S \setminus H$ is contained in a bag in $\mathcal{B}_{S\cap H}$. Hence $i \in T^{\tilde{\text{opt}}}_{S\cap H}$, which implies $S \setminus H \subseteq T^{\tilde{\text{opt}}}_{S\cap H}$. The feasibility of S implies that $\sum_{i \in S\setminus H : e \in P(i)} d(i) \leq u_F(e)$ for every edge e, and $|(S \setminus H) \cap B| \leq 1$ for every $B \in \mathcal{B}_{S\cap H}$. Therefore the integral solution s which has $s_i = 1$ for $i \in S \setminus H$ and $s_i = 0$ for the remaining entries is a feasible solution for $LP^{\tilde{\text{opt}}}_{S\cap H}$. Define $lp^{\tilde{\text{opt}}}_{S\cap H} := \sum_{i \in T^{\tilde{\text{opt}}}_{S\cap H}} w(i) \cdot \lambda_i^{\tilde{\text{opt}}, S\cap H}$ as the optimal LP value for $LP^{\tilde{\text{opt}}}_{S\cap H}$. The feasibility of s implies

$$w(S \setminus H) = \sum_{i \in T^{\tilde{\text{opt}}}_{S\cap H}} w(i) \cdot s_i \leq lp^{\tilde{\text{opt}}}_{S\cap H}.$$

On the other hand,

$$w(L^{\tilde{\text{opt}}}_{S\cap H}) \geq lp^{\tilde{\text{opt}}}_{S\cap H} - 2m \cdot \frac{2\varepsilon}{m}\tilde{\text{opt}} \geq lp^{\tilde{\text{opt}}}_{S\cap H} - 4\varepsilon \cdot \text{opt}.$$

In the first inequality above we used the fact that $\lambda^{\tilde{\text{opt}}, S\cap H}$ has at most $2m$ non-integral values (by Lemma 10), and that each $i \in T^{\tilde{\text{opt}}}_{S\cap H}$ has $w(i) \leq \frac{2\varepsilon}{m}\tilde{\text{opt}}$ by construction. In the second inequality above we used the assumption that $\tilde{\text{opt}} \leq \text{opt}$. Putting everything together:

$$\begin{aligned} w(A^{\tilde{\text{opt}}}_{S\cap H}) &= w(L^{\tilde{\text{opt}}}_{S\cap H}) + w(S \cap H) \geq lp^{\tilde{\text{opt}}}_{S\cap H} - 4\varepsilon \cdot \text{opt} + w(S \cap H) \\ &\geq w(S \setminus H) - 4\varepsilon \cdot \text{opt} + w(S \cap H) = w(S) - 4\varepsilon \cdot \text{opt} \geq (1-7\varepsilon)\text{opt}. \end{aligned}$$ ◀

The proof of Theorem 3 follows directly from Lemma 8, Lemma 9, and Lemma 11.

2.2 Representative Set Construction

In this section, we construct a small ε-representative set for the BagUFP instance I; this gives the proof of Lemma 7. Let $\tilde{\text{opt}} \in [w(T)]$ be a guess of the optimum value opt. Recall that in Section 2.1 we are able to find $\tilde{\text{opt}} \in \left(\frac{\text{opt}}{2}, \text{opt}\right]$ using exponential search over the domain $[w(T)]$.

We define a partition of the heavy tasks (and some tasks that are almost heavy) into *classes*, such that tasks of the same class have roughly the same weight and have the same subpath. Specifically, let $\Phi = \{P(i) \mid i \in T\}$ be the set of unique paths in the instance and define $\eta = \left\lceil \log_{1-\varepsilon}\left(\frac{\varepsilon}{2\cdot m}\right)\right\rceil$ as a parameter describing the number of classes. For all $\varphi \in \Phi$ and $r \in [\eta]$ define the *class* of φ and r as

$$\tilde{H}(\varphi, r) = \left\{ i \in T \;\middle|\; \frac{w(i)}{2 \cdot \tilde{\text{opt}}} \in \left((1-\varepsilon)^r, (1-\varepsilon)^{r-1} \right] \text{ and } P(i) = \varphi \right\}. \tag{1}$$

In simple words, a task i belongs to class $\tilde{H}(\varphi, r)$ have weight roughly $(1-\varepsilon)^r \cdot 2 \cdot \tilde{\text{opt}}$ and the subpath of i is φ. Define

$$\tilde{H} = \bigcup_{\varphi \in \Phi, r \in [\eta]} \tilde{H}(\varphi, r)$$

as the union of classes. The parameter η is carefully chosen so that the weight of every task $i \in \tilde{H}$ satisfies $w(i) \geq \frac{\varepsilon \cdot \tilde{\text{opt}}}{m}$, implying that i is roughly heavy. Since $\tilde{\text{opt}} \in \left[\frac{\text{opt}}{2}, \text{opt}\right]$, it follows that $H \subseteq \tilde{H}$ and that $\tilde{H}$ does not contain tasks with significantly smaller weight than $\frac{\varepsilon \cdot \text{opt}}{m}$ - that is the minimum weight allowed for heavy tasks.

▶ **Observation 12.** *$H \subseteq \tilde{H}$.*

Let $\mathcal{D} = \{\tilde{H}(\phi, r) \mid \phi \in \Phi, r \in [\eta]\}$ be the set of classes. We use a simple upper bound on the number of classes.

▶ **Lemma 13.** *$|\mathcal{D}| \leq 3 \cdot m^3 \cdot \varepsilon^{-2}$.*

Proof. Observe that

$$\log_{1-\varepsilon}\left(\frac{\varepsilon}{2 \cdot m}\right) \leq \frac{\ln\left(\frac{2 \cdot m}{\varepsilon}\right)}{-\ln(1-\varepsilon)} \leq \frac{2 \cdot m \cdot \varepsilon^{-1}}{\varepsilon} = 2 \cdot m \cdot \varepsilon^{-2}. \tag{2}$$

The second inequality follows from $x < -\ln(1-x), \forall x > -1, x \neq 0$, and $\ln(y) < y, \forall y > 0$. Moreover, the number of subpaths $\varphi \in \Phi$ is bounded by $|\Phi| = \binom{m}{2} \leq m^2$. Therefore, the number of classes is bounded by

$$|\mathcal{D}| \leq m^2 \cdot \left(\log_{1-\varepsilon}\left(\frac{\varepsilon}{2 \cdot m}\right) + 1\right) \leq 2 \cdot m \cdot \varepsilon^{-2} \cdot m^2 + m^2 = 2 \cdot m^3 \cdot \varepsilon^{-2} + m^2 \leq 3 \cdot m^3 \cdot \varepsilon^{-2} \tag{3}$$

The first inequality follows from (2). ◀

Algorithm 2 RepSet$(I, \varepsilon, \tilde{\text{opt}})$.

```
input  : BagUFPinstance $I$, an error parameter $0 < \varepsilon < \frac{1}{2}$, and $\tilde{\text{opt}} \in [w(T)]$.
output : An $\varepsilon$-representative set $R$ for $I$ (if $\tilde{\text{opt}} \in \left[\frac{\text{opt}}{2}, \text{opt}\right]$).
1 Initialize $R \leftarrow \emptyset$.
2 forall $\varphi \in \Phi$ and $r \in [\eta]$ do
3     Let $\mathcal{B}(\varphi, r) = \{B \in \mathcal{B} \mid B \cap \tilde{H}(\varphi, r) \neq \emptyset\}$.
4     For every $B \in \mathcal{B}(\varphi, r)$ define $i_B(\varphi, r) = \arg\min_{i \in B \cap \tilde{H}(\varphi, r)} d(i)$.
5     Sort $\mathcal{B}(\varphi, r)$ in non-decreasing order $B_1(\varphi, r), \ldots, B_\ell(\varphi, r)$
        by $d(i_B(\varphi, r))$ $\forall B \in \mathcal{B}(\varphi, r)$.
6     Define $a = \min\{q(\varepsilon, m), |\mathcal{B}(\varphi, r)|\}$.
7     Update $R \leftarrow R \cup \{i_{B_1}(\varphi, r), \ldots, i_{B_a}(\varphi, r)\}$.
8 end
9 Return $R$.
```

Our representative set construction is fairly simple. For each class $\tilde{H}(\varphi, r)$, consider the set of *active* bags $\mathcal{B}(\varphi, r)$ for $\tilde{H}(\varphi, r)$ that contain at least one task in $\tilde{H}(\varphi, r)$. For every active bag $B \in \mathcal{B}(\varphi, r)$ define the *representative* of B in the class $\tilde{H}(\varphi, r)$ as the the task from the bag B in the class $\tilde{H}(\varphi, r)$ of minimum demand (if there is more than one such task we choose one arbitrarily). We sort the active bags of the class in a non-decreasing order according to the demand of the representatives of the bags. Finally, we take the first a representatives (at most one from each bag) according to this order, where a is the minimum between the parameter $q(\varepsilon, m)$ and the number of active bags for the class. The pseudocode of the algorithm is given in Algorithm 2.

We give an outline of the proof of Lemma 7. Consider some optimal solution OPT for the instance. We partition the tasks in OPT into three sets: L, J_{k^*}, and Q such that (i) the maximum weight of a task in Q is at most ε-times the minimum weight of a task in L; (ii) L

is small: $|L| \leq \frac{q(\varepsilon,m)}{2}$; (iii) The weight of J_{k^*} is small: $w(J_{k^*}) \leq \varepsilon \cdot \text{opt}$. To prove that R is a representative set, we need to replace $H \cap \text{OPT}$ with tasks from R. As a first step, we define a mapping h from $\tilde{H} \cap \text{OPT}$ to R, where each task $i \in \tilde{H} \cap \text{OPT}$ is replaced by a task from the same class of a smaller or equal demand. For tasks $i \in \tilde{H} \cap \text{OPT}$ such that R contains a representative from the bag of i in the class of i, we simply define $h(i)$ as this representative; for other tasks, we define the mapping via a bipartite matching on the remaining tasks and representatives.

We define a solution S satisfying the conditions of Definition 6 in two steps. First, we define initial solutions S_1, S_2. The solution S_1 contains the mapping $h(i)$ of every $i \in \tilde{H} \cap \text{OPT}$ and the tasks in $L \setminus \tilde{H}$; the solution S_2 contains all tasks in Q from bags that do not contain tasks from S_1. Finally, we define $S = S_1 \cup S_2$. By the properties of L, J_{k^*}, and Q we are able to show that S is roughly an optimal solution. Specifically, by (iii) discarding J_{k^*} from the solution S does not have a significant effect on the total weight of S. Additionally,by property (i) there is a large gap between the weights in S_1 and S_2; thus, combined with property (ii) we lose only a small factor due to tasks discarded from Q, and it follows that the weight of S is $(1 - O(\varepsilon)) \cdot \text{opt}$.

Proof of Lemma 7

We start with the running time analysis of the algorithm.

▷ Claim 14. The running time of Algorithm 2 is bounded by $m^3 \cdot \varepsilon^{-2} \cdot |I|^{O(1)}$ on input I, ε, and $\tilde{\text{opt}}$. Moreover, $|R| \leq 3 \cdot m^3 \cdot \varepsilon^{-2} \cdot q(\varepsilon, m)$.

Proof. Each iteration of the **for** loop of the algorithm can be trivially computed in time $|I|^{O(1)}$. In addition, the number of iterations of the **for** loop is bounded by $3 \cdot m^3 \cdot \varepsilon^{-2}$ using Lemma 13. Therefore, the running time of the algorithm is bounded by $m^3 \cdot \varepsilon^{-2} \cdot |I|^{O(1)}$. For the second property of the lemma, recall that the number of classes is bounded by $3 \cdot m^3 \cdot \varepsilon^{-2}$ using Lemma 13. By Algorithm 2 of the algorithm, the number of tasks taken to R from each class is at most $q(\varepsilon, m)$. Therefore, $|R| \leq 3 \cdot m^3 \cdot \varepsilon^{-2} \cdot q(\varepsilon, m)$. ◁

If $\tilde{\text{opt}} \notin \left[\frac{\text{opt}}{2}, \text{opt}\right]$, the proof immediately follows from Claim 14. Thus, for the following assume that $\tilde{\text{opt}} \in \left[\frac{\text{opt}}{2}, \text{opt}\right]$. Let $\text{OPT} \subseteq T$ be an optimal solution for I. Let $w^* = \frac{\varepsilon \cdot \tilde{\text{opt}}}{m}$ be a lower bound on the minimum weight of a task in $\tilde{H}$. We partition a subset of the tasks in $\text{OPT} \setminus \tilde{H}$ with the highest weights into $N = \lceil \varepsilon^{-1} \rceil$ disjoint sets. For all $k \in [N]$ define the k-th set as

$$J_k = \left\{ i \in \text{OPT} \setminus \tilde{H} \;\middle|\; w(i) \in \left(\varepsilon^k \cdot w^*, \varepsilon^{k-1} \cdot w^*\right] \right\}. \tag{4}$$

Let $k^* = \arg\min_{k \in [N]} w(J_k)$. By (4) the sets $J_1, \ldots, J_N$ are $N \geq \varepsilon^{-1}$ disjoint sets (some of them may be empty); thus, $w(J_{k^*}) \leq \varepsilon \cdot \text{opt}$. Define

$$L \;= \left(\text{OPT} \cap \tilde{H}\right) \cup \bigcup_{k \in [k^*-1]} J_k$$

as the subset of all tasks in OPT of weight greater than $\varepsilon^{k^*-1} \cdot w^*$, and define $Q = \text{OPT} \setminus (L \cup J_{k^*})$ as the remaining tasks in OPT excluding J_{k^*}. We use the following auxiliary claim.

▷ Claim 15. $|L| \leq \frac{q(\varepsilon,m)}{2}$.

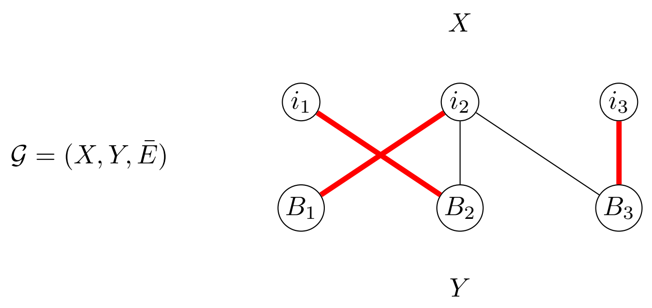

Figure 1 An illustration of the graph $\mathcal{G}$ and the maximum matching M (in red). Every edge (i, B) in the graph indicates that bag B belongs to $\mathsf{fit}(i)$; that is, the representative from B in the class of i belongs to R and the demand of this representative is at most the demand of i. Note that even though i_1 and i_2 are both connected to bag B_2, i_1 and i_2 may belong to different classes.

Proof. If $L = \emptyset$ the claim trivially follows. Otherwise,

$$|L| \leq \sum_{i \in L} \frac{w(i)}{\varepsilon^{k^*-1} \cdot w^*} = \frac{w(L)}{\varepsilon^{k^*-1} \cdot w^*} \leq \frac{\text{opt}}{\varepsilon^{k^*-1} \cdot w^*} \tag{5}$$

The first inequality holds since $w(i) \geq \varepsilon^{k^*-1} \cdot w^*$ for all $i \in L$. The second inequality follows from the fact that $L \subseteq \text{OPT}$; thus, L is a solution for I. Thus, by (5) and the definition of w^*

$$|L| \leq \frac{\text{opt}}{\varepsilon^{k^*-1} \cdot 2\tilde{\text{opt}} \cdot \frac{\varepsilon}{2 \cdot m}} \leq \frac{\text{opt}}{\varepsilon^{k^*-1} \cdot \text{opt} \cdot \frac{\varepsilon}{2 \cdot m}} = \frac{2 \cdot m}{\varepsilon^{k^*}} \leq \frac{2 \cdot m}{\varepsilon^{N}} \leq \frac{q(\varepsilon, m)}{2}.$$

The second inequality holds since we assume that $\tilde{\text{opt}} \geq \frac{\text{opt}}{2}$. ◁

Let R be the set returned by the algorithm. In the following, we show the existence of a solution S such that $S \cap H \subseteq R$ and $w(S) \geq (1 - 3 \cdot \varepsilon) \cdot \text{opt}$; this gives the statement of the lemma by Definition 6. To construct S, we first define a mapping h from $\tilde{H} \cap \text{OPT}$ to R. For a subpath $\varphi \in \Phi$ and $r \in [\eta]$, recall the set of active bags $\mathcal{B}(\varphi, r)$ and the representatives $i_B(\varphi, r)$ for all $B \in \mathcal{B}(\varphi, r)$ (see Algorithm 2).

For the simplicity of the notation, for $\varphi \in \Phi$, $r \in [\eta]$, and $i \in \tilde{H}(\varphi, r)$ let $\tilde{H}^i = \tilde{H}(\varphi, r)$ be the class to which i belongs and let $r_i = r$; moreover, for $B \in \mathcal{B}$ such that $i \in B$ define $B^i = B$ as the bag containing i. We first consider tasks i in $\text{OPT} \cap \tilde{H}$ whose bag does not have a representative in R from the class of i, i.e., $R \cap \tilde{H}^i \cap B^i = \emptyset$. Define this set of tasks as

$$X = \left\{ i \in \text{OPT} \cap \tilde{H} \;\middle|\; R \cap \tilde{H}(\varphi, r) \cap B^i = \emptyset \right\}. \tag{6}$$

The above set X contains all tasks $i \in \text{OPT} \cap \tilde{H}$ whose corresponding bag does not have a representative in R from the class of i. We define a bipartite graph, in which X is one side of the graph. The other side of the graph is

$$Y = \mathcal{B} \setminus \left\{ B \in \mathcal{B} \;\middle|\; \exists i \in L \text{ s.t. } B = B^i \right\}. \tag{7}$$

In words, Y describes all *available* bags, the collection of all bags that do not contain a task in L. Define the bipartite graph $\mathcal{G} = (X, Y, \bar{E})$ such that the set of edges is defined as follows. For some $i \in X$ let

$$\mathsf{fit}(i) = \left\{ B \in Y \;\middle|\; B \cap R \cap \tilde{H}^i \neq \emptyset \text{ and } d\left(i_B(P(i), r_i)\right) \leq d(i) \right\}. \tag{8}$$

The set $\mathsf{fit}(i)$ describes all bags that can potentially matched to i; these bags have a representative from the class $\tilde{H}^i = \tilde{H}(P(i), r_i)$ that contain i and the representative of the bag have a smaller or equal demand w.r.t. i. Now, a task i can be matched to a bag B only if $B \in \mathsf{fit}(i)$, i.e., define

$$\bar{E} = \left\{ (i, B) \in X \times Y \;\middle|\; B \in \mathsf{fit}(i) \right\}. \tag{9}$$

Let M be a maximum matching in $\mathcal{G}$. We give an illustration of the above construction in Figure 1. We show that M matches all vertices in X.

▷ Claim 16. For every $i \in X$ there is $B \in Y$ such that $(i, B) \in M$.

Proof. Assume towards a contradiction that there is $i \in X$ such that for all $B \in Y$ it holds that $(i, B) \notin M$. Let $\varphi \in \Phi$ and $r \in [\eta]$ such that $\tilde{H}^i = \tilde{H}(\varphi, r)$. Since $i \in X$, by (6) it holds that $R \cap \tilde{H}(\varphi, r) \cap B^i = \emptyset$. Intuitively, this means that the algorithm preferred other bags over B^i in the selection of representatives for class $\tilde{H}(\varphi, r)$. Therefore, by Algorithm 2 of the algorithm, there are $q(\varepsilon, m)$ distinct bags $B_1 = B_1(\varphi, r), \ldots, B_{q(\varepsilon,m)} = B_{q(\varepsilon,m)}(\varphi, r)$ such that for all $j \in [q(\varepsilon, m)]$ it holds that $i_{B_j}(\varphi, r) \in R$ and $d\left(i_{B_j}(\varphi, r)\right) \leq d(i)$. Thus, for all $j \in [q(\varepsilon, m)]$ it holds that $(i, B_j) \in \bar{E}$ by (8) and (9). In addition,

$$|M| \leq |X| \leq |L| \leq \frac{q(\varepsilon, m)}{2} < q(\varepsilon, m). \tag{10}$$

The first inequality holds since M is a matching in $\mathcal{G}$ and X is one side of a bipartition of $\mathcal{G}$. The second inequality holds since $X \subseteq L$ by (6) and the definition of L. The third inequality follows from Claim 15. The last inequality holds since $q(\varepsilon, m) \geq 2$ assuming $0 < \varepsilon < \frac{1}{2}$ and $m \geq 1$. By (10) there is $j \in [q(\varepsilon, m)]$ such that for all $t \in X$ it holds that $(t, B_j) \notin M$. In particular, $(i, B_j) \notin M$ and recall that $(i, B_j) \in \bar{E}$. Therefore, $M \cup (i, B_j)$ is a matching in $\mathcal{G}$ in contradiction that M is a maximum matching in $\mathcal{G}$. ◁

For every $i \in X$ define $M_i = B$ such that $(i, B) \in M$, i.e., M_i is the bag matched to i in M. By Claim 16 it holds that each task in X is matched and every bag is matched at most once. We define the mapping h from $\tilde{H} \cap \mathrm{OPT}$ to R. Define $h : \tilde{H} \cap \mathrm{OPT} \to R$ such that for all $i \in \tilde{H} \cap \mathrm{OPT}$:

$$h(i) = \begin{cases} i_{B^i}\left(P(i), r_i\right), & \text{if } B^i \cap R \cap \tilde{H}^i \neq \emptyset \\ i_{M_i}\left(P(i), r_i\right), & \text{else} \end{cases} \tag{11}$$

In words, a task $i \in \tilde{H} \cap \mathrm{OPT}$ is mapped to a task $h(i)$ such that if the bag of i contains a representative in R in the class of i - then $h(i)$ is this representative; otherwise, $h(i)$ is the representative of the bag M_i matched to i by the matching M. Clearly, h is well defined by Claim 16. We list immediate properties of h.

▶ **Observation 17.** *The function h satisfies the following.*

- *For every $i \in \tilde{H} \cap \mathrm{OPT}$ it holds that $d(h(i)) \leq d(i)$ and $\tilde{H}^{h(i)} = \tilde{H}^i$.*
- *For every $i, j \in \tilde{H} \cap \mathrm{OPT}$, $i \neq j$, it holds that $B^{h(i)} \neq B^{h(j)}$.*
- *For every $i \in \tilde{H} \cap \mathrm{OPT}$ and $t \in L \setminus \tilde{H}$ it holds that $B^{h(i)} \neq B^t$.*

The first property follows from the definition of the graph $\mathcal{G}$ and the definition of the bag representatives in Algorithm 2. The second and third properties hold since OPT takes at most one task from each bag and using the definition of $\mathcal{G}$. We can finally define the solution S that satisfies the conditions of Definition 6. Define

$$S_1 = \left\{ h(i) \mid i \in \tilde{H} \cap \mathrm{OPT} \right\} \cup \left(L \setminus \tilde{H} \right) \tag{12}$$

and

$$S_2 = \left\{ i \in Q \mid B^i \neq B^t \; \forall t \in S_1 \right\}. \tag{13}$$

Define $S = S_1 \cup S_2$. We show that S satisfies the conditions of Definition 6. As an immediate property of the construction we have the following.

▶ **Observation 18.** *h is a one-to-one function from $\tilde{H} \cap \text{OPT}$ to $S \cap \tilde{H}$.*

We use the above to prove the feasibility of S.

▷ Claim 19. S is a solution for I.

Proof. We show that S satisfies the bag constraints. Let $B \in \mathcal{B}$. Since OPT is a solution for I, there is at most one $i \in B \cap \text{OPT}$. We consider four cases depending on the task i.

1. If $i \in \tilde{H}$ and $R \cap \tilde{H}^i \cap B^i \neq \emptyset$. Then, $h(i) \in B$ by (11) and for all $t \in S_1 \setminus \{h(i)\}$ it holds that $t \notin B$ by Observation 17. Furthermore, for all $t \in S_2$ it holds that $t \notin B$ by (13). Thus, $|B \cap S| \leq 1$.
2. If $i \in \tilde{H}$ and $R \cap \tilde{H}^i \cap B^i = \emptyset$. Then, as $\tilde{H} \subseteq L$ it holds that $i \in L$; thus, $B \notin Y$ by (7). Therefore, by (12) we conclude that $|B \cap S_1| = 0$; thus,

   $$|B \cap S| = |B \cap S_2| \leq |B \cap Q| \leq |B \cap \text{OPT}| \leq 1.$$

   The equality holds since $|B \cap S_1| = 0$. The first inequality follows from (13). The last inequality holds since OPT is a solution.
3. If $i \in L \setminus \tilde{H}$. Then, by (12) and (13) it holds that $|B \cap S| = |B \cap i| = 1$.
4. If $i \in Q$. Then, there are two sub cases. If $i \in S_2$, by (13) for all $t \in S_1$ it holds that $B \neq B^t$; thus, as $|Q \cap B| \leq |\text{OPT} \cap B| \leq 1$ it follows that $|S \cap B| \leq 1$. Otherwise, $i \notin S_2$; then, by (13) it holds that

   $$|B \cap S| = |B \cap S_1| \leq 1.$$

   The inequality follows from Observation 17.

By the above we conclude that S satisfies all bag constraints. It remains to prove that S satisfies the capacity constraints of all edges. For $e \in E$

$$\begin{aligned}
\sum_{i \in S \text{ s.t. } e \in P(i)} d(i) &= \sum_{i \in S \cap \tilde{H} \text{ s.t. } e \in P(i)} d(i) + \sum_{i \in S \setminus \tilde{H} \text{ s.t. } e \in P(i)} d(i) \\
&= \sum_{i \in \text{OPT} \cap \tilde{H} \text{ s.t. } e \in P(i)} d(h(i)) + \sum_{i \in S \setminus \tilde{H} \text{ s.t. } e \in P(i)} d(i) \\
&\leq \sum_{i \in \text{OPT} \cap \tilde{H} \text{ s.t. } e \in P(i)} d(h(i)) + \sum_{i \in \text{OPT} \setminus \tilde{H} \text{ s.t. } e \in P(i)} d(i) \\
&\leq \sum_{i \in \text{OPT} \cap \tilde{H} \text{ s.t. } e \in P(i)} d(i) + \sum_{i \in \text{OPT} \setminus \tilde{H} \text{ s.t. } e \in P(i)} d(i) \\
&= \sum_{i \in \text{OPT} \text{ s.t. } e \in P(i)} d(i) \\
&\leq u(e).
\end{aligned}$$

The second equality holds since h is a one-to-one mapping from $\text{OPT} \cap \tilde{H}$ to $S \cap \tilde{H}$ by Observation 18. The first inequality holds since $S \setminus \tilde{H} \subseteq \text{OPT} \setminus \tilde{H}$ by (12) and (13). The second inequality holds since $d(h(i)) \leq d(i)$ for all $i \in \text{OPT} \cap \tilde{H}$ by Observation 17. The last inequality holds since OPT is a solution for I. ◁

Observe that there is a substantial gap in weight between tasks in L and tasks in Q. We use this gap in the following auxiliary claim.

▷ Claim 20. $w\left(Q \setminus S\right) \leq \varepsilon \cdot \text{opt}$.

Proof. Observe that

$$|Q \setminus S| = |Q \setminus S_2| = \left|\left\{i \in Q \mid \exists t \in S_1 \text{ s.t. } B^i = B^t\right\}\right| \leq |S_1| = |L|. \tag{14}$$

The inequality holds since Qsatisfies the bag constraints (i.e., $|Q \cap B| \leq 1$ for all $B \in \mathcal{B}$); thus, for each $t \in S_1$ there can be at most one $i \in Q$ such that $B^i = B^t$ (and only in this case i is discarded from S_2). The last equality holds since h is a one-to-one mapping from $\text{OPT} \cap \tilde{H}$ to $S \cap \tilde{H}$ by Observation 18 and since $L \setminus \tilde{H}$ belongs both to S_1 and L. Hence,

$$w(Q \setminus S) \leq |Q \setminus S| \cdot \varepsilon^{k^*} \cdot w^* \leq |L| \cdot \varepsilon^{k^*} \cdot w^* \leq \varepsilon \cdot w(L) \leq \varepsilon \cdot w(\text{OPT}) = \varepsilon \cdot \text{opt}.$$

The first inequality holds since $w(i) \leq \varepsilon^{k^*} \cdot w^*$ for all $i \in Q$. The second inequality follows from (14). The third inequality holds since $w(i) > \varepsilon^{k^*-1} \cdot w^*$ for all $i \in L$. the last inequality holds since $L \subseteq \text{OPT}$. ◁

The following claim shows that S satisfies the total weight required by Definition 6.

▷ Claim 21. $w\left(S\right) \geq (1 - 3\varepsilon) \cdot \text{opt}$.

Proof. We first give a lower bound to the weight of S_1.

$$\begin{aligned} w(S_1) &= w\left((L \setminus \tilde{H}) \cup \left\{h(i) \mid i \in \tilde{H} \cap \text{OPT}\right\}\right) \\ &= w\left(L \setminus \tilde{H}\right) + \sum_{i \in \tilde{H} \cap \text{OPT}} w(h(i)) \\ &\geq w\left(L \setminus \tilde{H}\right) + \sum_{i \in \tilde{H} \cap \text{OPT}} (1-\varepsilon) \cdot w(i) \\ &\geq (1-\varepsilon) \cdot w(L). \end{aligned} \tag{15}$$

The inequality holds since for all $i \in \text{OPT} \cap \tilde{H}$ it holds that $\tilde{H}^i = W^{h(i)}$ by Observation 17; thus, by (1) it follows that $w(h(i)) \geq (1-\varepsilon) \cdot w(i)$. For the last inequality, recall that $\tilde{H} \subseteq L$. Moreover,

$$w(S_2) = w(Q) - w(Q \setminus S) \geq w(Q) - \varepsilon \cdot \text{opt} \geq (1-\varepsilon) \cdot w(Q) - \varepsilon \cdot \text{opt}. \tag{16}$$

The first equality holds since $S_2 \subseteq Q$. The first inequality follows from Claim 20. By (15) and (16) we have

$$\begin{aligned} w(S) \quad &= w(S_1) + w(S_2) \\ &\geq (1-\varepsilon) \cdot w(L \cup Q) - \varepsilon \cdot \text{opt} \\ &= (1-\varepsilon) \cdot w(\text{OPT} \setminus J_{k^*}) - \varepsilon \cdot \text{opt} \\ &\geq (1-\varepsilon) \cdot (1-\varepsilon) \cdot \text{opt} - \varepsilon \cdot \text{opt} \\ &\geq (1-3\varepsilon) \cdot \text{opt}. \end{aligned}$$

The second inequality holds since $w(J_{k^*}) \leq \varepsilon \cdot \text{opt}$. ◁

Observe that $H \subseteq \tilde{H}$ by Observation 12. Moreover, $S \cap \tilde{H} = S_1 \cap \tilde{H} \subseteq R$ by (12) and (13). Thus, $S \cap H \subseteq R$. By Claim 19 and Claim 21, it follows that R is a representative set.

The proof follows from Claim 19, Claim 21, and Claim 14. ◀

3 A Faster p-EPTAS for UFP

In this section we prove Theorem 5. Let (G, u, T, P, d, w) be a UFP instance. For simplicity, we next assume that $1/\varepsilon$ is integer and that $n \gg 1/\varepsilon$. Recall that $\Phi = \{P(i) \mid i \in T\}$ is the set of unique paths in the instance, and for every $\varphi \in \Phi$ we use $T_\varphi = \{i \in T \mid P(i) = \varphi\}$ to denote the set of tasks with path φ. Observe that $|\Phi| \leq \frac{1}{2} \cdot m \cdot (m+1)$.

See Algorithm 3 for a pseudocode description of our approach.

Algorithm 3 p-EPTAS for UFP.

```
input      : UFP instance $I = (G, u, T, P, d, w)$ and a parameter $0 < \varepsilon < 0.1$
output     : A feasible solution APX for the instance $I$
Notations : Here $w_{\max} = \max_{i \in T} w(i)$ and $p_\varphi := \sum_{i \in T_\varphi} p(i)$ for all $\varphi \in \Phi$.
1  Define $p(i) \leftarrow \left\lfloor \frac{n \cdot w(i)}{\varepsilon \cdot w_{\max}} \right\rfloor$ for every $i \in T$.
2  For all $\varphi \in \Phi$ compute $\mathtt{DP}_\varphi$ for the Knapsack instance $(T_\varphi, d, p, \min_{e \in \varphi} u(e))$.
3  APX $\leftarrow \emptyset$.
4  for all the powers $\tilde{\text{opt}}$ of $(1+\varepsilon)$ in $\left[1, \frac{n^2}{\varepsilon}\right]$ do
5      for all non-negative integers $(X_\varphi)_{\varphi \in \Phi}$ such that $\sum_{\varphi \in \Phi} X_\varphi \leq |\Phi| \cdot \frac{1+\varepsilon}{\varepsilon}$ do
6          Set $\tilde{\text{opt}}_\varphi = X_\varphi \cdot \frac{\varepsilon}{|\Phi|} \cdot \tilde{\text{opt}}$ for all $\varphi \in \Phi$.
7          APX$' \leftarrow \bigcup_{\varphi \in \Phi} \mathtt{DP}_\varphi \left(\min\left\{p_\varphi, \lceil \tilde{\text{opt}}_\varphi \rceil\right\}\right)$.
8          if APX$'$ is a feasible solution for $I$ and $p(\text{APX}') \geq p(\text{APX})$ then
               APX $\leftarrow$ APX$'$.
9      end
10 end
11 Return APX.
```

We start to perform a standard rounding of the weights (similar to several other packing problems) so that they are positive integers in a polynomially bounded range. Let $w_{\max} = \max_{i \in T} w_i$ be the maximum weight of any task. Observe that, since w.l.o.g. each task alone induces a feasible solution, one has opt $\geq w_{\max}$. We replace each weight $w(i)$ with $p(i) := \left\lfloor \frac{n \cdot w(i)}{\varepsilon \cdot w_{\max}} \right\rfloor$. A standard calculation shows that an optimum solution OPT$'$ computed w.r.t. the modified weights p is a $(1-\varepsilon)$-approximate solution w.r.t. the original problem. Now the (rounded) weights are in the range $\left[\frac{n}{\varepsilon}\right]$. With the obvious notation, for $S \subseteq T$, we will denote $p(S) := \sum_{i \in S} p(i)$.

Now we proceed by describing the two main phases of our p-EPTAS. In the first phase we consider each path $\varphi \in \Phi$, and define a Knapsack instance $K_\varphi = (T_\varphi, d, p, \min_{e \in \varphi} u(e))$. Here T_φ is the set of items that can be placed in the knapsack, $d(i)$ and $p(i)$ are the size and profit of item $i \in T_\varphi$, resp., and $\min_{e \in \varphi} u(e)$ is the size of the knapsack. We solve this instance K_φ using the standard algorithm for Knapsack based on dynamic programming. In more detail, this algorithm defines a dynamic programming table $\mathtt{DP}_\varphi$ indexed by the possible values $p' \in [p_\varphi]$ of the profit, where $p_\varphi := \sum_{i \in T_\varphi} p(i)$. At the end of the algorithm, for each such p', $\mathtt{DP}_\varphi(p')$ contains a subset of items (in T_φ) of minimum total size (or, equivalently, demand) whose profit is at least p'[2]. Notice that T_φ satisfies $p(T_\varphi) = p_\varphi \geq p'$, hence all the

[2] In a more standard version of the algorithm $\mathtt{DP}_\varphi(p')$ would contain a minimum size solution of profit *exactly* p', or a special character if such solution does not exist. However, it is easy to adapt the algorithm to rather obtain the desired values.

table entries $\mathtt{DP}_\varphi(p')$ are well defined. We also remark that certain table entries may contain a solution of total demand larger than $\min_{e\in\varphi} u(e)$, hence such entries will never be used to construct a feasible UFP solution. Computing the dynamic tables for all $\varphi \in \Phi$ takes time

$$\sum_{\varphi\in\Phi} O(|T_\varphi| \cdot p_\varphi) \le \sum_{\varphi\in\Phi} O(|T_\varphi|) \cdot \sum_{\varphi\in\Phi} O(p_\varphi) = O(|T|) \cdot O(p(T)) \le O\left(\frac{n^3}{\varepsilon}\right).$$

We store these dynamic tables for later use.

At this point the second phase of the algorithm starts. Let $\mathrm{opt}' = p(\mathrm{OPT}')$, where OPT' is an optimal solution for the UFP instance with the rounded weights p (i.e., (G, u, T, P, d, p)). Observe that $\mathrm{opt}' \in \left[\frac{n^2}{\varepsilon}\right]$. Let $\tilde{\mathrm{opt}}$ be a power of $(1+\varepsilon)$ such that $\frac{\mathrm{opt}'}{1+\varepsilon} < \tilde{\mathrm{opt}} \le \mathrm{opt}'$. We can find this value by trying all the $O\left(\log_{1+\varepsilon} \frac{n^2}{\varepsilon}\right) = O\left(\frac{1}{\varepsilon} \cdot \log n\right)$ possibilities. Define $\mathrm{OPT}'_\varphi := \mathrm{OPT}' \cap T_\varphi$ and $\mathrm{opt}'_\varphi = p(\mathrm{OPT}'_\varphi)$. For each $\varphi \in \Phi$ we guess the largest multiple $\tilde{\mathrm{opt}}_\varphi = X_\varphi \cdot \frac{\varepsilon}{|\Phi|} \cdot \tilde{\mathrm{opt}}$ of $\frac{\varepsilon}{|\Phi|} \cdot \tilde{\mathrm{opt}}$ which is upper bounded by opt'_φ. Again by guessing we mean trying all the possible combinations. Obviously a valid guess must satisfy

$$\frac{\varepsilon}{|\Phi|} \cdot \tilde{\mathrm{opt}} \cdot \sum_{\varphi\in\Phi} X_\varphi = \sum_{\varphi\in\Phi} \tilde{\mathrm{opt}}_\varphi \le \sum_{\varphi\in\Phi} \mathrm{opt}'_\varphi = \mathrm{opt}' \le (1+\varepsilon)\tilde{\mathrm{opt}},$$

hence $\sum_{\varphi\in\Phi} X_\varphi \le Y := \lfloor \frac{1+\varepsilon}{\varepsilon} |\Phi| \rfloor$. Thus it is sufficient to generate all the ordered sequences of $|\Phi|$ non-negative integers whose sum is at most Y. As we will argue, the number of such sequences is sufficiently small.

Given a guess $\{\tilde{\mathrm{opt}}_\varphi\}_{\varphi\in\Phi}$, we compute a tentative solution $\mathrm{APX}' := \cup_{\varphi\in\Phi} \mathtt{DP}_\varphi(\min\{p_\varphi, \lceil \tilde{\mathrm{opt}}_\varphi \rceil\})$ using the pre-computed dynamic tables. Notice that, for a valid guess of $\tilde{\mathrm{opt}}_\varphi$, by integrality we also have $\lceil \tilde{\mathrm{opt}}_\varphi \rceil \le \mathrm{opt}'_\varphi$. Upper bounding with $p_\varphi \ge \mathrm{opt}'_\varphi$ guarantees that the algorithm only uses well-defined table entries. Among the solutions APX' which are feasible, we return one APX of maximum profit $p(\mathrm{APX})$. This concludes the description of the algorithm.

We can further improve the running time as follows. Let us compute and store the values $d(\mathtt{DP}_\varphi(p'))$ and $p(\mathtt{DP}_\varphi(p'))$ (this does not affect the asymptotic running time). In the for loops we only update the current value of $\mathrm{apx} := p(\mathrm{APX})$ instead of updating APX explicitly each time. Furthermore for each tentative solution APX' we only compute $p(\mathrm{APX}')$ and $\sum_{i\in\mathrm{APX}':e\in P(i)} d(i)$ for each $e \in E$. This can be done in $O(|\Phi| m)$ time and it is sufficient to check whether APX' is a feasible solution and whether $p(\mathrm{APX}') > \mathrm{apx}$. We maintain the combination of the parameters X^*_φ and $\tilde{\mathrm{opt}}^*$ that lead to the current value of apx. At the end of the process from the optimal parameters X^*_φ and $\tilde{\mathrm{opt}}^*$ we derive a corresponding solution APX of profit apx in $O(n + |\Phi|) = O(n^2)$ extra time.

▶ **Lemma 22.** *Algorithm 3 produces a feasible* UFP *solution.*

Proof. Obviously since $\mathrm{APX} = \emptyset$ is a feasible solution, and whenever we update APX, we do that with the value APX' of a feasible solution. ◀

▶ **Lemma 23.** *Algorithm 3 produces a $(1-2\varepsilon)$-approximate solution.*

Proof. Let us show that $p(\mathrm{APX}) \ge (1-\varepsilon)p(\mathrm{OPT}')$. Notice that $\mathrm{opt}' = p(\mathrm{OPT}') \in [\frac{n^2}{\varepsilon}]$, hence there is a value $\tilde{\mathrm{opt}}$ considered by the algorithm such that $\frac{1}{1+\varepsilon}\mathrm{opt}' < \tilde{\mathrm{opt}} \le \mathrm{opt}'$. Let us focus on execution of the external for loop with that value of $\tilde{\mathrm{opt}}$.

Recall that $\mathrm{opt}'_\varphi = p(\mathrm{OPT}'_\varphi) = p(\mathrm{OPT}' \cap T_\varphi)$. As already argued before, there are corresponding values $(X_\varphi)_{\varphi\in\Phi}$ considered by the algorithm such that $\tilde{\mathrm{opt}}_\varphi = X_\varphi \cdot \frac{\varepsilon}{|\Phi|} \cdot \tilde{\mathrm{opt}}$ satisfies:

$$\mathrm{opt}'_\varphi - \frac{\varepsilon}{|\Phi|}\tilde{\mathrm{opt}} \le \tilde{\mathrm{opt}}_\varphi \le \mathrm{opt}'_\varphi.$$

Let us focus on the execution of the inner for loop with these values of X_φ (hence $\tilde{\mathrm{opt}}_\varphi$). The profit of the corresponding solution APX' is at least

$$\sum_{\varphi\in\Phi} \tilde{\mathrm{opt}}_\varphi \ge \sum_{\varphi\in\Phi} \mathrm{opt}'_\varphi - \varepsilon\tilde{\mathrm{opt}} = \mathrm{opt}' - \varepsilon\tilde{\mathrm{opt}} \ge (1-\varepsilon)\mathrm{opt}'.$$

Observe that $\mathrm{OPT}'_\varphi = \mathrm{OPT}' \cap T_\varphi$ is a valid solution for the Knapsack instance K_φ with profit opt'_φ, where $p_\varphi \ge \mathrm{opt}'_\varphi \ge \lceil\tilde{\mathrm{opt}}_\varphi\rceil$, hence also a valid candidate solution for $\mathtt{DP}_\varphi\left(\min\{p_\varphi, \lceil\tilde{\mathrm{opt}}_\varphi\rceil\}\right)$. As a consequence $d(\mathtt{DP}_\varphi(\min\{p_\varphi, \lceil\tilde{\mathrm{opt}}_\varphi\rceil\}) \le d(\mathrm{OPT}'_\varphi)$. We conclude that APX' is a feasible solution. In more detail, for each $e \in E$,

$$\sum_{i\in \mathrm{APX}': e\in P(i)} d(i) = \sum_{\varphi\in\Phi: e\in\varphi} d(\mathtt{DP}_\varphi(\lceil\tilde{\mathrm{opt}}_\varphi\rceil)) \le \sum_{\varphi\in\Phi: e\in\varphi} d(\mathrm{OPT}'_\varphi) = \sum_{i\in \mathrm{OPT}': e\in P(i)} d(i) \le u(e).$$

It follows that $p(\mathrm{APX}) \ge p(\mathrm{APX}') \ge (1-\varepsilon)\mathrm{opt}'$. Using standard arguments, we conclude that

$$\begin{aligned} w(\mathrm{APX}) &\ge \frac{\varepsilon \cdot w_{\max}}{n} \cdot p(\mathrm{APX}) \\ &\ge (1-\varepsilon) \cdot \frac{\varepsilon \cdot w_{\max}}{n} p(\mathrm{OPT}') \\ &\ge (1-\varepsilon) \cdot \frac{\varepsilon \cdot w_{\max}}{n} \cdot p(\mathrm{OPT}) \\ &\ge (1-\varepsilon) \cdot \left(\frac{\varepsilon \cdot w_{\max}}{n}\left(\frac{n}{\varepsilon \cdot w_{\max}} \cdot w(\mathrm{OPT}) - n\right)\right) \\ &= (1-\varepsilon) \cdot (\mathrm{opt} - \varepsilon \cdot w_{\max}) \ge (1-\varepsilon)\cdot(1-\varepsilon)\mathrm{opt}. \end{aligned}$$ ◀

It remains to upper bound the running time. Let `iters` be the number of iterations of the inner loop in Algorithm 3, i.e. the number of possible valid combinations for $(X_\varphi)_{\varphi\in\Phi}$. The bound on the running time follows easily from the following technical lemma.

▶ **Lemma 24.** `iters` $\le \left(\frac{1+2\varepsilon}{\varepsilon} \cdot e\right)^{|\Phi|}$.

▶ **Lemma 25.** *Algorithm 3 runs in time* $O\left(\frac{n^3}{\varepsilon} + \left(\frac{1}{\varepsilon}\right)^{O(m^2)} \cdot m^3 \cdot \log n\right)$.

Proof. We already argued that the dynamic tables can be computed in total time $O(\frac{n^3}{\varepsilon})$. We also observed that the outer for loop is executed $O\left(\frac{1}{\varepsilon}\log n\right)$ times. As already discussed, lines 6-8 take $O\left(|\Phi| \cdot m\right)$ time. Thus the second phase of the algorithm can be implemented in time $O(n^2 + |\Phi| \cdot m \cdot \mathtt{iters} \cdot \frac{1}{\varepsilon} \cdot \log n)$ time. By Lemma 24, the overall running time of the algorithm is

$$O\left(\frac{n^3}{\varepsilon} + \left(\frac{1+2\varepsilon}{\varepsilon} \cdot e\right)^{|\Phi|} \cdot m \cdot |\Phi| \cdot \frac{1}{\varepsilon}\log n\right).$$

The claim follows since $|\Phi| \le \frac{1}{2} \cdot m \cdot (m+1)$. ◀

It remains to prove Lemma 24. To that aim, we need a standard bound on the binomial coefficients. Let $\mathcal{H}(x) = -x \cdot \ln(x) - (1-x) \cdot \ln(1-x)$ be the entropy function and assume $\mathcal{H}(0) = \mathcal{H}(1) = 0$.

▶ **Lemma 26** (Example 11.1.3 in [20]). *For every $n \in \mathbb{N}$ and integer $0 \le k \le n$ it holds that $\binom{n}{k} \le \exp\left(n \cdot \mathcal{H}\left(\frac{k}{n}\right)\right)$.*

Proof of Lemma 24. Recall that `iters` is equal to the possible sequences of $|\Phi|$ non-negative integers whose sume is at most $Y = \left\lfloor \frac{1+\varepsilon}{\varepsilon} \cdot |\Phi| \right\rfloor$. These sequences can be represented via a binary string as follows. Let $\varphi_1, \ldots \varphi_{|\Phi|}$ be an arbitrary ordering of Φ, and $X_i = X_{\varphi_i}$. The bit string consists of X_1 many 1s, followed by one 0, followed by X_2 many 1s and so on, ending with the $X_{|\Phi|}$ many 1s, an additional 0 and a final padding of 1s till the target length of Y is reached. In particular all valid sequences correspond to binary strings with $Y + |\Phi|$ digits and exactly $|\Phi|$ zeros. It is therefore sufficient to upper bound the number of the latter bit strings, namely $\binom{Y+|\Phi|}{|\Phi|}$. By Lemma 26 we have,

$$\begin{aligned}
\texttt{iters} &= \binom{|\Phi| + \left\lfloor |\Phi| \cdot \frac{1+\varepsilon}{\varepsilon} \right\rfloor}{|\Phi|} \\
&\le \exp\left(\left(|\Phi| + \left\lfloor |\Phi| \cdot \frac{1+\varepsilon}{\varepsilon} \right\rfloor\right) \cdot \mathcal{H}\left(\frac{|\Phi|}{|\Phi| + \left\lfloor |\Phi| \cdot \frac{1+\varepsilon}{\varepsilon} \right\rfloor}\right)\right) \\
&\le \exp\left(\left(|\Phi| \cdot \frac{1+2\varepsilon}{\varepsilon}\right) \cdot \mathcal{H}\left(\frac{|\Phi|}{|\Phi| \cdot \frac{1+2\varepsilon}{\varepsilon}}\right)\right) = \left(\exp\left(\frac{1+2\varepsilon}{\varepsilon} \cdot \mathcal{H}\left(\frac{\varepsilon}{1+2\varepsilon}\right)\right)\right)^{|\Phi|},
\end{aligned} \tag{17}$$

where the last inequality holds since $x \cdot \mathcal{H}\left(\frac{a}{x}\right)$ is increasing in x for any $a \ge 1$ and $|\Phi| + \left\lfloor |\Phi| \cdot \frac{1+\varepsilon}{\varepsilon} \right\rfloor \le |\Phi| \cdot \frac{1+2\varepsilon}{\varepsilon}$. It also holds that

$$\begin{aligned}
\frac{1+2\varepsilon}{\varepsilon} \cdot \mathcal{H}\left(\frac{\varepsilon}{1+2\varepsilon}\right) &= \frac{1+2\varepsilon}{\varepsilon} \cdot \left(-\frac{\varepsilon}{1+2\varepsilon} \cdot \ln\left(\frac{\varepsilon}{1+2\varepsilon}\right) - \left(1 - \frac{\varepsilon}{1+2\varepsilon}\right) \cdot \ln\left(1 - \frac{\varepsilon}{1+2\varepsilon}\right)\right) \\
&\le -\ln\left(\frac{\varepsilon}{1+2\varepsilon}\right) - \frac{1+2\varepsilon}{\varepsilon} \cdot \left(1 - \frac{\varepsilon}{1+2\varepsilon}\right) \cdot \left(-\frac{\varepsilon}{1+2\varepsilon}\left(1 + \frac{\varepsilon}{1+2\varepsilon}\right)\right) \\
&= \ln\left(\frac{1+2\varepsilon}{\varepsilon}\right) + \left(1 - \frac{\varepsilon}{1+2\varepsilon}\right)\left(1 + \frac{\varepsilon}{1+2\varepsilon}\right) \\
&\le \ln\left(\frac{1+2\varepsilon}{\varepsilon}\right) + 1
\end{aligned} \tag{18}$$

where the first inequality follows from $\ln(1-x) \ge -x(1+x)$ for $x \in (0.0.1)$, and the second inequality holds as $(1+x)(1-x) \le 1$ for every $x \in \mathbb{R}$. By (17) and (18) we have,

$$\texttt{iters} \le \left(\exp\left(\frac{1+2\varepsilon}{\varepsilon} \cdot \mathcal{H}\left(\frac{\varepsilon}{1+2\varepsilon}\right)\right)\right)^{|\Phi|} \le \left(\exp\left(\ln\left(\frac{1+2\varepsilon}{\varepsilon}\right) + 1\right)\right)^{|\Phi|} = \left(\frac{1+2\varepsilon}{\varepsilon} \cdot e\right)^{|\Phi|}$$

◀

We now have the tools required to complete the proof of Theorem 5.

Proof of Theorem 5. It follows directly from Lemmas 22, 23 and 25 by choosing the parameter $\varepsilon/2$ so as to have a $(1-\varepsilon)$ approximation. ◀

4 A Lower bound for BagUFP

In this section we prove Theorem 1 using a simple reduction from the partition problem.

Proof of Theorem 1

Recall that in the NP-complete Partition problem we are given a collection of n non-negative integers $A = \{a_1, \ldots, a_n\}$ in $[0, 1]$ whose sum is $2M$. Our goal is to determine whether there exists a subset of numbers whose sum is precisely M.

We show that an FPTAS for BagUFP in the considered case implies a polynomial time algorithm to solve Partition, hence the claim. We build (in polynomial time) an instance of BagUFP with 2 edges e_1 and e_2, both of capacity M. Furthermore, for each a_j, we create two tasks t^1_j and t^2_j, with demand a_j and subpath e_1 and e_2, resp. All the tasks have profit 1. The bags are given by the pairs $\{t^1_j, t^2_j\}$, $j = 1, \ldots, n$. Obviously, the input Partition instance is a YES instance iff the optimal solution to the corresponding BagUFP instance has value n, i.e. exactly one task per bag is selected (notice that a solution cannot have larger profit). Indeed, given a solution $A' \subseteq A$ for the Partition instance, a valid solution to the corresponding BagUFP instance is obtained by selecting all the tasks t^1_j with $j \in A'$ and all the tasks t^2_j with $j \notin A'$. Notice that the total demand of the tasks using e_1 and e_2 must be exactly M. Vice versa, given a BagUFP solution S of profit n, the selected tasks $S_1 \subseteq S$ of type t^1_j must have total demand exactly M, hence inducing a valid Partition solution $A' := \{j \in \{1, \ldots, n\} : t^1_j \in S_1\}$.

We run the mentioned FPTAS on the obtained BagUFP instance with parameter $\varepsilon = \frac{1}{2n}$ (hence taking polynomial time). If the optimal solution is n, the FPTAS will return a solution of profit at least $\frac{n}{1+\varepsilon} \geq n - \frac{1}{2+1/n} > n - 1$, hence a solution of profit n since the profit is an integer. Otherwise, the FPTAS will return a solution of profit at most $n - 1$. This is sufficient to discriminate between YES and NO instances of Partition. ◀

References

1 Hesham K Alfares. Survey, categorization, and comparison of recent tour scheduling literature. *Annals of Operations Research*, 127:145–175, 2004. doi:10.1023/B:ANOR.0000019088.98647.E2.

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 $2+\varepsilon$ approximation for unsplittable flow on a path. In *SODA*, pages 26–41, 2014.

4 Kenneth R Baker. Workforce allocation in cyclical scheduling problems: A survey. *Journal of the Operational Research Society*, 27(1):155–167, 1976.

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. doi:10.1145/1132516.1132617.

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 A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC '00)*, pages 735–744. ACM, 2000. doi:10.1145/335305.335410.

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

9 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. doi:10.1137/1.9781611973730.5.

10 Stephen E Bechtold, Michael J Brusco, and Michael J Showalter. A comparative evaluation of labor tour scheduling methods. *Decision Sciences*, 22(4):683–699, 1991.

11 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. doi:10.1137/120868360.

12 Gruia Călinescu, Amit Chakrabarti, Howard J. Karloff, and Yuval Rabani. An improved approximation algorithm for resource allocation. *ACM Transactions on Algorithms*, 7:48:1–48:7, 2011. doi:10.1145/2000807.2000816.

13 Venkatesan T. Chakaravarthy, Anamitra R. Choudhury, Shalmoli Gupta, Sambuddha Roy, and Yogish Sabharwal. Improved algorithms for resource allocation under varying capacity. In *ESA*, pages 222–234, 2014. doi:10.1007/978-3-662-44777-2_19.

14 A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar. Approximation algorithms for the unsplittable flow problem. *Algorithmica*, 47:53–78, 2007. doi:10.1007/S00453-006-1210-5.

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 Marek Chrobak, Gerhard J Woeginger, Kazuhisa Makino, and Haifeng Xu. Caching is hard—even in the fault model. *Algorithmica*, 63(4):781–794, 2012.

20 TM Cover and Joy A Thomas. Elements of information theory, 2006.

21 A. Darmann, U. Pferschy, and J. Schauer. Resource allocation with time intervals. *Theoretical Computer Science*, 411:4217–4234, 2010. doi:10.1016/J.TCS.2010.08.028.

22 Syamantak Das and Andreas Wiese. On minimizing the makespan with bag constraints. In *13th Workshop on Models and Algorithms for Planning and Scheduling Problems*, page 186, 2017.

23 Ilan Doron-Arad, Fabrizio Grandoni, and Ariel Kulik. Unsplittable flow on a short path. *arXiv preprint*, 2024. doi:10.48550/arXiv.2407.10138.

24 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. An aptas for bin packing with clique-graph conflicts. In *Algorithms and Data Structures: 17th International Symposium, WADS 2021, Virtual Event, August 9–11, 2021, Proceedings 17*, pages 286–299. Springer, 2021. doi:10.1007/978-3-030-83508-8_21.

25 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. An afptas for bin packing with partition matroid via a new method for lp rounding. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2023)*. Schloss-Dagstuhl – Leibniz Zentrum für Informatik, 2023.

26 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. Budgeted matroid maximization: a parameterized viewpoint. *IPEC*, 2023.

27 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. An EPTAS for budgeted matching and budgeted matroid intersection via representative sets. In *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.

28 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. An EPTAS for budgeted matroid independent set. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 69–83, 2023. doi:10.1137/1.9781611977585.CH7.

29 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. An fptas for budgeted laminar matroid independent set. *Operations Research Letters*, 51(6):632–637, 2023. `doi:10.1016/J.ORL.2023.10.005`.

30 Ilan Doron-Arad, Ariel Kulik, and Hadas Shachnai. Lower bounds for matroid optimization problems with a linear constraint. *ICALP proc.*, 2024.

31 Ilan Doron-Arad and Hadas Shachnai. Approximating bin packing with conflict graphs via maximization techniques. In Daniël Paulusma and Bernard Ries, editors, *WG 2023, Fribourg, Switzerland, June 28-30, 2023, Revised Selected Papers*, 2023.

32 Kilian Grage, Klaus Jansen, and Kim-Manuel Klein. An eptas for machine scheduling with bag-constraints. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, pages 135–144, 2019. `doi:10.1145/3323165.3323192`.

33 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. `doi:10.1007/978-3-319-28684-6_2`.

34 Fabrizio Grandoni, Tobias Mömke, and Andreas Wiese. Faster $(1+\epsilon)$-approximation for unsplittable flow on a path via resource augmentation and back. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 49:1–49:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.ESA.2021.49`.

35 Fabrizio Grandoni, Tobias Mömke, and Andreas Wiese. A PTAS for unsplittable flow on a path. In *STOC*, pages 289–302. ACM, 2022. `doi:10.1145/3519935.3519959`.

36 Fabrizio Grandoni, Tobias Mömke, and Andreas Wiese. Unsplittable flow on a path: The game! In *SODA*, pages 906–926. SIAM, 2022. `doi:10.1137/1.9781611977073.39`.

37 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. `doi:10.1137/1.9781611974782.159`.

38 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, Los Angeles, CA, USA, June 25-29, 2018*, pages 607–619, 2018. `doi:10.1145/3188745.3188894`.

39 Fabrizio Grandoni and Rico Zenklusen. Approximation schemes for multi-budgeted independence systems. In *European Symposium on Algorithms*, pages 536–548. Springer, 2010. `doi:10.1007/978-3-642-15775-2_46`.

40 Hans Kellerer, Ulrich Pferschy, David Pisinger, Hans Kellerer, Ulrich Pferschy, and David Pisinger. The multiple-choice knapsack problem. *Knapsack problems*, pages 317–347, 2004.

41 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.

42 Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008. `doi:10.1093/COMJNL/BXM048`.

43 C. A. Phillips, R. N. Uma, and J. Wein. Off-line admission control for general scheduling problems. In *Proceedings of the 11^{th} Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '00)*, pages 879–888. ACM, 2000.

44 David Pisinger. A minimal algorithm for the multiple-choice knapsack problem. *European Journal of Operational Research*, 83(2):394–410, 1995.

45 Alexander Schrijver et al. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

46 Prabhakant Sinha and Andris A Zoltners. The multiple-choice knapsack problem. *Operations Research*, 27(3):503–515, 1979. `doi:10.1287/OPRE.27.3.503`.

47 Frits C. R. Spieksma. On the approximability of an interval scheduling problem. *Journal of Scheduling*, 2:215–227, 1999.

48 Jorne Van den Bergh, Jeroen Beliën, Philippe De Bruecker, Erik Demeulemeester, and Liesje De Boeck. Personnel scheduling: A literature review. *European journal of operational research*, 226(3):367–385, 2013. doi:10.1016/J.EJOR.2012.11.029.

49 Andreas Wiese. A (1+ϵ)-approximation for unsplittable flow on a path in fixed-parameter running time. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 67:1–67:13, 2017.