A $(1+\epsilon)$-Approximation for Unsplittable Flow on a Path in Fixed-Parameter Running Time*

Andreas Wiese

Department of Industrial Engineering and Center for Mathematical Modeling, Universidad de Chile, Santiago, Chile
awiese@dii.uchile.cl

Abstract

Unsplittable Flow on a Path (UFP) is a well-studied problem. It arises in many different settings such as bandwidth allocation, scheduling, and caching. We are given a path with capacities on the edges and a set of tasks, each of them is described by a start and an end vertex and a demand. The goal is to select as many tasks as possible such that the demand of the selected tasks using each edge does not exceed the capacity of this edge. The problem admits a QPTAS and the best known polynomial time result is a $(2+\epsilon)$-approximation. As we prove in this paper, the problem is intractable for fixed-parameter algorithms since it is W[1]-hard. A PTAS seems difficult to construct. However, we show that if we combine the paradigms of approximation algorithms and fixed-parameter tractability we can break the mentioned boundaries. We show that on instances with $|OPT| = k$ we can compute a $(1+\epsilon)$-approximation in time $2^{O(k \log k)} n^{O_\epsilon(1)} \log u_{\max}$ (where $u_{\max}$ is the maximum edge capacity). To obtain this algorithm we develop new insights for UFP and enrich a recent dynamic programming framework for the problem. Our results yield a PTAS for (unweighted) UFP instances where $|OPT|$ is at most $O(\log n / \log \log n)$ and they imply that the problem does not admit an EPTAS, unless W[1] = FPT.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Combinatorial optimization, Approximation algorithms, Fixed-parameter algorithms, Unsplittable Flow on a Path

Digital Object Identifier 10.4230/LIPIcs.ICALP.2017.67

1 Introduction

The Unsplittable Flow on a Path problem is motivated by many different settings such as scheduling, bandwidth allocation, and caching. We are given an undirected path $G = (V, E)$ with a capacity $u(e) \in \mathbb{N}$ for each edge $e \in E$. Also, we are given a set of n tasks T. Each task $i \in T$ is specified by a subpath $P(i) \subseteq V$ between (and including) the start (i. e., leftmost) vertex $s(i) \in V$ and the end (i. e., rightmost) vertex $t(i) \in V$, and a demand $d(i) \in \mathbb{N}$. For instance, the tasks can be seen as jobs with start and end times that need some portion of a shared resource. The goal is to select a subset $T' \subseteq T$ of tasks of maximum total size such that for each edge e the total demand of the selected tasks using e does not exceed $u(e)$.

UFP is NP-hard [7, 14] and therefore approximation algorithms have been studied for the problem. The best known polynomial time algorithm yields a $(2+\epsilon)$-approximation [3] (improving previous results [5, 7]) and for some cases even a $(1+\epsilon)$-approximation is known [6, 17, 12]. Also, there is a QPTAS [4, 6] which makes it plausible that also a PTAS

* This work was partially supported by the Millennium Nucleus Information and Coordination in Networks ICM/FIC RC130003.

44th International Colloquium on Automata, Languages, and Programming (ICALP 2017).
Editors: Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl;
Article No. 67; pp. 67:1–67:13


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

exists. However, despite the recent progress on the problem [6, 17] the best known polynomial time result is still the mentioned $(2+\epsilon)$-approximation [3], and a PTAS seems difficult to construct. We note that all above algorithms even work in the weighted case of the problem in which each task has a profit associated with it and one wants to maximize the total profit of the selected tasks. However, as it is typical in the FPT-literature, in this paper we restrict ourselves to the unweighted case, i.e., we assume that each task yields a profit of one. No better results than the above are known for this case.

Another approach for NP-hard problems are fixed-parameter algorithms. For any instance one identifies a parameter k, e.g., the value of the optimal solution, and searches for an exact algorithm with a running time of $f(k) \cdot n^{O(1)}$ for some (typically exponential) function f. A problem is called fixed-parameter tractable (FPT) if it admits such an algorithm. We refer the reader to the recent textbook by Cygan et al. [13] for an introduction to FPT algorithms. For UFP, throughout this paper our parameter will be the size of the optimal solution. Unfortunately, as we show in this paper, it is unlikely that UFP is FPT since the problem is W[1]-hard.

1.1 Our Contribution

In this paper we show that if we combine the paradigms of approximation and fixed-parameter algorithms then we can break the mentioned barriers of $2+\epsilon$ and W[1]-hardness for UFP. We present an algorithm with a running time of $2^{O(k \log k)} n^{O_\epsilon(1)} \log u_{\max}$ that computes a $(1+\epsilon)$-approximation for any instance with $|OPT| = k$, i.e., the computed solution contains at least $k/(1+\epsilon)$ tasks, where $u_{\max} = \max_e u(e)$. Hence, we obtain a PTAS for (unweighted) UFP for instances where $|OPT| \leq O(\log n / \log \log n)$.

We first consider the special case where the number of different task demands in the input is bounded by a parameter k'. We show that then there exists an optimal solution that has a special structure. We can guess this structure in FPT-time (i.e., the number of possibilities is bounded by a function $f(k, k')$) and based on this we can construct the solution deterministically.

Then, we generalize this result to the setting where the tasks have arbitrary demands and the edge capacities are in a bounded range. There, we show that if we have k tasks with relatively small demand (of at most a $1/k$-fraction of the capacities of the edges they are using) then they form a feasible solution and we are done. Otherwise, in FPT-time we can guess which of these tasks are contained in the optimal solution and then focus on the remaining, relatively large tasks. For those we use a result from [6] that shows that there is a $(1+\epsilon)$-approximative solution in which (essentially) each edge has some slack that equals the minimum size of a large task. Thus, there is still a near-optimal solution if we round up the task demands so that they have only $f(k)$ many different demands. On the resulting instance, we apply our FPT-algorithm from above.

To obtain an algorithm for the general case with arbitrary task demands and edge capacities we use the machinery that was introduced in [17] in order to turn a PTAS for UFP with resource augmentation (i.e., where the edge capacities are increased by a factor $1+\epsilon$) to a PTAS without resource augmentation. In order to apply it in our setting, we need several new ideas.

First, we prove that we can identify at most k vertices of the input path such that each input task uses one of them (for instances in which no k such vertices exist we can find a solution with k tasks using a greedy algorithm). These vertices divide the path into $k+1$ *segments*. Our algorithm proceeds in phases and in each phase we process some set of tasks. These tasks are divided into tiny and non-tiny tasks. A crucial difficulty is to

estimate how much capacity should be given to each of these groups on each edge. We cannot afford to guess this for each edge separately. However, we show that there exists a $(1+\epsilon)$-approximative solution in which this allocation has a structure that we can guess in FPT-time. For each segment S we can essentially argue that (i) either it is not used by non-tiny tasks and in this case we can give the whole edge capacity to the tiny tasks (ii) or all tiny tasks use the same capacity on each edge of S which we can guess. Then, for the remaining decisions for the tiny tasks we call the FPT-algorithm for the case of a bounded range of edge capacities. For the non-tiny tasks we know that each segment is used by at most $O_\epsilon(1)$ of them and we can guess them step by step in polynomial time.

Finally, we prove that UFP is W[1]-hard (if parametrized by the size of the optimal solution) which makes it unlikely that there is a fixed-parameter algorithm for it that computes an optimal solution, instead of an approximation. Also, this implies that UFP does not admit an EPTAS (i.e., an $(1+\epsilon)$-approximation algorithm with a running time of $f(\epsilon)\cdot n^{O(1)}$ for some function f), unless W[1] = FPT.

We hope that our new techniques yield progress for eventually finding a PTAS for UFP. For instance, many algorithms for UFP are based on a recursive decomposition of the problem, embedded into a DP [3, 6, 17]. Using our new algorithm we can now stop such a recursion once the optimal solution of a considered subproblem has a size of at most $O(\log n/\log\log n)$. Also, for the setting of bounded edge capacities we proved that there exist optimal solutions with a special structure (inherited from the case of few different task demands). This insight might be useful beyond our result. Note that even for uniform edge capacities no PTAS is known for UFP.

We would like to point out that in the literature there exists the notion of an FPT-approximation scheme (FPT-AS) which is a $(1+\epsilon)$-approximation algorithm with a running time of $f(\epsilon,k)\cdot n^{O(1)}$ for some suitable function f, while our algorithm has a running time of $2^{O(k\log k)}n^{O_\epsilon(1)}\log u_{\max}$ and thus ϵ appears in the exponent of n. For UFP, we cannot hope for an FPT-AS since otherwise we could choose e.g., $\epsilon := 1/(2k)$ and obtain an FPT-algorithm for UFP, thus contradicting that the problem is W[1]-hard. There are many FPT-ASs known in the literature, see [18] and references therein.

We note that due to space constraints many proofs and details are omitted in this extended abstract.

1.2 Other related work

If all input tasks of a UFP instance have (relatively) small demand compared to the capacities of the edges they use, Chekuri et al. [12] proved that there is a $(1+\epsilon)$-approximation via LP-rounding. This unifies (and improves) previously known results for the special cases of uniform edge capacities [8] and the no-bottleneck-assumption (NBA) [9] which requires that $\max_{i\in T} d(i) \leq \min_{e\in E} u(e)$. Under the latter assumption, tasks with relatively large demands can be handled via dynamic programming, and thus $O(1)$- and $(2+\epsilon)$-approximation algorithms were known for these cases [8, 9, 12] and later such algorithms were also found for the general case of the problem [7, 3]. Another line of research on UFP is to find good LP-relaxations for the problem.

The natural LP-relaxation suffers from an integrality gap of $\Omega(n)$ [9] but with additional constraints Chekuri et al. [11] reduced it to $O(\log^2 n)$ (which was later improved to $O(\log n)$ by the same authors [10]). Anagnostopoulos et al. [2] found a compact LP for the cardinality case of UFP with constant integrality gap and an extended formulation with a constant gap for the weighted case. Grandoni et al. [16] prove the currently best integrality gap for an LP-relaxation without additional variables of $O(\log n/\log\log n)$.

1.3 Preliminaries and Notation

In a UFP instance, for each edge $e \in E$, let $T_e \subseteq T$ be the subset of tasks i using edge e, i. e., with $e \in P(i)$. For every set of tasks T' we define $d(T') := \sum_{i \in T'} d(i)$. The goal of (unweighted) UFP is to select set of tasks T' with maximum cardinality $|T'|$ such that $d(T' \cap T_e) \leq u(e)$ for each edge e. For a given instance of UFP we denote by OPT an optimal solution. Without loss of generality, we may assume that $|V| = 2n$, that each vertex is either the start-vertex or the end-vertex of exactly one input task, and that each task alone yields a feasible solution [4]. Throughout this paper, we use the notation $O_\epsilon(f(n))$ for functions that are in $O(f(n))$ if ϵ is a constant. In particular, $O_\epsilon(1)$ represents a value that depends only on ϵ.

2 Bounded task demands or edge capacities

In this section we first present an algorithm for the special case that the number of different task demands in the input instance is bounded by a parameter k'. Afterwards we will use it as a subroutine for the case where the range of edge capacities is bounded by a parameter k'' (without a bound on the task demands).

2.1 Bounded number of task demands

Suppose we are given an instance with at most k' different task demands where $|OPT| = k$. We present an algorithm with a running time of $f(k, k') \cdot n^{O(1)}$ that computes an optimal solution.

We first guess some properties of OPT. We can assume w.l.o.g. that OPT does not contain any task i such that there is a task $i' \in T \setminus OPT$ with $d(i) = d(i')$ and $P(i') \subseteq P(i)$ (otherwise we could replace i by i'). Assume that $OPT = \{i(1), ..., i(k)\}$ such that $s(i(\ell))$ lies on the left of $s(i(\ell'))$ if and only if $\ell < \ell'$. We use color-coding (see [1]) to split the input tasks into k pair-wise disjoint groups $T^1, ..., T^k$ such that for each ℓ the group T^ℓ contains $i(\ell)$. Note that in [1] the authors present a version of the color-coding method that does not require randomization.

▶ **Lemma 1** (implied by [1]). *By increasing the running time by a factor $2^{O(k)} \log n$ we can assume that the input tasks are colored with k colors $\{1, ..., k\}$ such that for each $\ell \in [k]$ the task $i(\ell)$ is colored with color ℓ.*

Next, we guess for each $\ell \in \{1, ..., k\}$ the demand of the task $i(\ell)$. Since for each task there are k' possibilities, the total number of guesses is $(k')^k$. We remove from T^ℓ all tasks whose demand does not equal the demand that we guessed for $i(\ell)$. Furthermore, we remove from T^ℓ each task i such that there is another task $i' \in T^\ell$ with $i \neq i'$ and $P(i') \subseteq P(i)$. Note that by our assumption about OPT above this does not remove the task $i(\ell)$. Denote by $\bar{T}^\ell$ the resulting set for each $\ell \in [k]$.

In the next lemma we define an (optimal) solution OPT' with k tasks. It will turn out that the information guessed so far is sufficient to construct OPT'. We say that a task $i \in T$ is *compatible* with a set of tasks T' if $T' \cup \{i\}$ is a feasible solution.

▶ **Lemma 2.** *There is a solution $OPT' = \{i'(1), ..., i'(k)\}$ (with $|OPT'| = k$) that satisfies for each $\ell \in \{1, ..., k\}$ that the task $i'(\ell)$ is the task in $\bar{T}^\ell$ with the leftmost start vertex that is compatible with the tasks $i'(1), ..., i'(\ell - 1)$.*

Proof. We prove the lemma by transforming $OPT =: OPT_0$ step by step into OPT'. For each $\ell \in \{1, ..., k\}$ let OPT_ℓ be the solution obtained after ℓ steps. We will ensure that each OPT_ℓ is feasible and contains k tasks. Let $i'(1)$ be the task in $\bar{T}^1$ with the leftmost start vertex. If $i'(1) \in OPT_0$ then we define $OPT_1 := OPT$. If $i'(1) \notin OPT_0$ then we replace $i(1)$ by $i'(1)$ and we define $OPT_1 := OPT \setminus \{i(1)\} \cup \{i'(1)\}$. We claim that OPT_1 is feasible. To this end, let us first consider all edges on the left of $s(i(1))$. By definition of $i(1)$, there is no task in OPT starting on the left of $s(i(1))$. By assumption, the task $i'(1)$ alone yields a feasible solution. Thus, for each edge e on the left of $s(i(1))$ we have that $d(OPT_1 \cap T_e) \leq u(e)$. By construction of the set $\bar{T}^1$ we know that $P(i(1)) \not\subseteq P(i'(1))$ and thus $t(i'(1))$ lies on the left of $t(i(1))$. Since $d(i(1)) = d(i'(1))$ and OPT is feasible we have that $d(OPT_1 \cap T_e) \leq u(e')$ for each edge e' on the right of $s(i(1))$.

Assume by induction that we constructed a feasible solution $OPT_\ell = \{i'(1), ..., i'(\ell), i(\ell + 1), ..., i(k)\}$ such that for each $\ell' \in \{1, ..., \ell\}$ the task $i'(\ell')$ is the task in $\bar{T}^{\ell}$ with the leftmost start vertex that is compatible with the tasks $i'(1), ..., i'(\ell' - 1)$ and that OPT_ℓ contains k tasks. Let $i'(\ell + 1)$ be the task in $\bar{T}^{\ell+1}$ with the leftmost start vertex that is compatible with the tasks $i'(1), ..., i'(\ell)$. Define $OPT_{\ell+1} := OPT_\ell \setminus \{i(\ell + 1)\} \cup \{i'(\ell + 1)\}$. Clearly, $OPT_{\ell+1}$ contains k tasks. We claim that $OPT_{\ell+1}$ is feasible. Let e be an edge on the left of $s(i(\ell + 1))$. Then e is not used by the tasks $i(\ell + 1), ..., i(k)$. By definition, $i'(\ell + 1)$ is compatible with the tasks $i'(1), ..., i'(\ell')$ and thus $d(OPT_{\ell+1} \cap T_e) \leq u(e)$. Let e' be an edge on the right of $s(i(\ell + 1))$. Again, by construction of the set $\bar{T}^{\ell+1}$ we know that $t(i'(\ell + 1))$ lies on the left of $t(i(\ell + 1))$. Thus, if e' is used by $i'(\ell + 1)$ then it is also used by $i(\ell + 1)$. Since by induction OPT_ℓ is feasible, this implies that also $OPT_{\ell+1}$ is feasible. ◀

Note that the start vertices of $i'(1), ..., i'(k)$ are not necessarily ordered, i.e., it could be that $s(i'(\ell))$ lies on the right of $s(i'(\ell + 1))$. Nevertheless, due to Lemma 2 we can use now the following algorithm to find a solution of size k. We define $i'(1)$ to be the task in $\bar{T}^1$ with the leftmost start vertex. Then, for each $\ell \in \{2, ..., k\}$ we inductively define $i'(\ell)$ to be the task in $\bar{T}^{\ell}$ with the leftmost start vertex that is compatible with the tasks $i'(1), ..., i'(\ell - 1)$. This yields the solution OPT' due to Lemma 2.

▶ **Theorem 3.** *Suppose we are given an UFP instance with k' different task demands in the input. Then there is an algorithm that computes a solution of size k in time $(k \cdot k')^k n^{O(1)}$ if such a solution exists.*

2.2 FPT-range of edge capacities

We give now a $(1 + \epsilon)$-approximation algorithm with a running time of $f(k, k'') \cdot n^{O_\epsilon(1)}$ for the case that the edge capacities differ by some parameter k''. Here, we allow arbitrary task demands in the input and thus lift the assumption from the previous section. Formally, our algorithm outputs a solution of size at least $k/(1 + \epsilon)$ or asserts that there is no solution of size k.

Let $u_{\min} = \min_{e \in E} u(e)$ and $u_{\max} = \max_{e \in E} u(e)$. We assume that $u_{\max} \leq k'' \cdot u_{\min}$ where k'' is a parameter. For each task i we define its bottleneck capacity $b(i) := \min_{e \in P(i)} u(e)$. We define a task i to be *large* if $d(i) \geq b(i)/k$ and small otherwise. The next lemma shows that if there are at least k small tasks then any k of them will form a feasible solution (and hence we are done). It holds even for arbitrary edge capacities.

▶ **Lemma 4.** *Any set of at most k small tasks forms a feasible solution.*

Proof. Let T' be a set of k small tasks. We want to prove that T' is a feasible solution. Let e be an edge. We have that $d(T' \cap T_e) = \sum_{i \in T' \cap T_e} d(i) < \sum_{i \in T' \cap T_e} b(i)/k \leq \frac{1}{k} \sum_{i \in T' \cap T_e} u(e) \leq u(e)$. ◀

If there are at least k small tasks in the input then we output k of them and we are done. Otherwise, denote by OPT_S the small tasks from OPT. We guess in time 2^{k-1} the set OPT_S, select all these tasks for our final solution, and discard all other small tasks. We focus on the large tasks now. Denote by OPT_L the set of large tasks in OPT.

We borrow an idea from [6, Lemma 2.6] to achieve the following: we sacrifice a factor of $1+O(\epsilon)$ in the objective and remove some tasks from OPT_L such that if an edge e is used by at least $1/\epsilon$ tasks in OPT_L we remove at least one task from $T_e \cap OPT_L$.

▶ **Lemma 5** ([6]). *There is a set $\overline{OPT}_L \subseteq OPT_L$ with $|\overline{OPT}_L| \leq O(\epsilon) \cdot |OPT_L|$ such that for each edge e with $|T_e \cap OPT_L| \geq 1/\epsilon$ we have that $|T_e \cap \overline{OPT}_L| \geq 1$.*

We define $OPT' := OPT \setminus \overline{OPT}_L$ with $\overline{OPT}_L$ being defined as in Lemma 5. Assume for a moment that each edge is used by at least one task in $\overline{OPT}_L$. Then we know that $d(T_e \cap OPT') \leq u(e) - \min_{i \in OPT_L} d(i)$. Since all tasks in OPT_L are large we have that $\min_{i \in OPT_L} d(i) \geq \frac{1}{k} \cdot u_{\min}$. On the other hand, $|OPT'| \leq k$. Thus, OPT' remains feasible if we increase the demand of each large task to the next higher integral multiple of $\frac{1}{k^2} u_{\min}$. Since $d(i) \leq u_{\max}$ for each task $i \in T$, this yields an instance with only $\frac{u_{\max}}{\frac{1}{k^2} \cdot u_{\min}} \leq k^2 \cdot k''$ different demands. We can then apply the exact FPT-algorithm from Section 2.1 on the resulting instance.

This procedure fails if there are edges not used by tasks in $\overline{OPT}_L$. Denote those edges by E_f. However, such edges are used by only few tasks in OPT_L, at most $1/\epsilon$ many. Thus, we can employ a dynamic program (DP) that guesses those edges step by step and guess their corresponding tasks. Any two consecutive edges e_L, e_R in E_f yield a subproblem for which (like above) we can increase the demands of the tasks whose path lies strictly between e_L and e_R and then invoke the exact FPT-algorithm from Section 2.1 as a subroutine.

▶ **Theorem 6.** *There is a $(1+\epsilon)$-approximation algorithm with a running time of $(k \cdot k'')^{O(k)} n^{O(1/\epsilon)}$ for UFP-instances with $|OPT| = k$ in which the edge capacities lie within a factor k''.*

3 General case

In this section we present our main result. For any $\epsilon > 0$ and any $k \in \mathbb{N}$ we present an algorithm with a running time of $2^{O(k \log k)} \cdot n^{O_\epsilon(1)}$ that computes a solution consisting of at least $k/(1+\epsilon)$ tasks on any instance with $|OPT| = k$ or asserts that there is no solution of size k. We will assume for the moment that the input numbers are bounded by a polynomial in the input size and later explain how to lift this assumption. Our argumentation consists of the following steps:

- In Section 3.1 we use techniques from [17] in order to gain some slack (i.e., unused capacity) on the edges while losing only a factor of $1+\epsilon$ in the objective. Then we classify edges into types and supertypes according to their respective amount of slack. We do a similar classification into types/supertypes for the tasks.
- Afterwards in Section 3.2, we identify a set $\bar{V}$ of k vertices such that each input task uses at least one of them. They split the input path into $k+1$ segments for which we establish some structural properties.
- We present the main algorithm in Section 3.3, described as a (possibly exponential time) recursion. It processes the tasks in phases with one phase for each supertype.
- Finally, we embed our recursive algorithm into a dynamic program with the claimed running time and lift the assumption that the input numbers are polynomially bounded.

3.1 Classification of tasks and edges

In this subsection, we apply the machinery presented in [17] in order to classify edges via how much unused capacity (i.e., *slack*) they have in some near-optimal solution. Also, we group tasks into tiny, medium, and huge tasks. Several times we apply some standard shifting arguments to ensure that we lose at most a factor $1 + O(\epsilon)$ in the process.

▶ **Lemma 7** ([17]). *Let $\epsilon > 0$ be a constant. Given a UFP instance with optimal solution OPT, there exists a feasible solution OPT' with $|OPT'| \geq (1 - O(\epsilon)) \cdot |OPT|$ such that for each edge e there is a value $\delta_e \geq 0$ satisfying the following conditions:*

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

In our reasoning, we will aim at computing a solution with nearly as many tasks as OPT'. Like in [17] we group the edges and the input tasks according to the amount of slack (i.e., the δ_e-values) that they have/that the edges on their respective paths have.

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

▶ **Definition 8.** A task $i \in T$ of type j is *huge* if $d(i) \geq \epsilon^2 \cdot \delta^{(j)}$.

Next, we group the tasks into supertypes. Each supertype consists of $1/\epsilon - 1$ (usual) types. We remove the tasks of all types $a + \ell/\epsilon - 1$ with $\ell \in \mathbb{N}$ for some offset $a \in \{0, ..., 1/\epsilon - 1\}$ and define the tasks supertypes $\mathcal{T}^{(\ell)} := \bigcup_{\ell'=a+\ell/\epsilon}^{a+\ell/\epsilon+1/\epsilon-2} T^{(\ell')}$, one for each $\ell \in \mathbb{Z}$. For a task i we say that i is of *supertype* ℓ if $i \in \mathcal{T}^{(\ell)}$ and we write stype$(i) = \ell$. Similarly, we define for the edges the supertypes $\mathcal{E}^{(\ell)} := \bigcup_{\ell'=a+\ell/\epsilon}^{a+\ell/\epsilon+1/\epsilon-2} E^{(\ell')}$ (for the same offset a as above) and write stype$(e) := \ell$ if $e \in \mathcal{E}^{(\ell)}$. This implies that edges of supertype ℓ have slacks in the range $[\left(\frac{1}{\epsilon^2}\right)^{a+\ell/\epsilon}, \left(\frac{1}{\epsilon^2}\right)^{a+\ell/\epsilon+1/\epsilon-2}] =: [s^{(\ell)}_{\min}, s^{(\ell)}_{\max}]$. The following proposition follows from a simple shifting argument.

▶ **Proposition 9.** *There exists an offset $a \in \{0, ..., 1/\epsilon - 1\}$ such that by reducing the number of tasks in OPT' by a factor $1 + O(\epsilon)$ we can assume that $OPT' \subseteq \bigcup_{\ell \in \mathbb{N}} \mathcal{T}^{(\ell)}$.*

Since we removed the tasks of all types $a + \ell/\epsilon - 1$ with $\ell \in \mathbb{N}$ we can guarantee that all non-huge tasks of a supertype $\mathcal{T}^{(\ell)}$ fit into the slack of each edge of supertype $\ell + 1$ or larger. Let $OPT'_{NH} \subseteq OPT'$ denote the tasks in OPT' that are not huge.

▶ **Lemma 10.** *Let $\mathcal{T}^{(\ell)}$ be a supertype and let $e \in \mathcal{E}^{(\ell)}$. Then $d(T_e \cap OPT'_{NH} \cap \mathcal{T}^{(\ell)}) \leq 10 \cdot \frac{1}{\epsilon^3} \cdot (\frac{1}{\epsilon^2})^{a+\ell/\epsilon+1/\epsilon-2} := d^{(\ell)}_{\max}$. Moreover, for each edge $e' \in \mathcal{E}^{(\ell')}$ with $\ell' \geq \ell + 1$ we have that $d(T_e \cap OPT'_{NH} \cap \mathcal{T}^{(\ell)}) \leq d^{(\ell)}_{\max} \leq 10\epsilon \cdot s^{(\ell+1)}_{\min} \leq 10\epsilon \cdot \delta_{e'}$.*

We split the non-huge tasks into tiny and medium tasks. Let $\mu_1, \mu_2 > 0$ with $\mu_1 < \mu_2$ be two constants to be defined later. We say that a non-huge task $i \in \mathcal{T}^{(\ell)}$ is *tiny* if $d(i) \leq \mu_1 \cdot s^{(\ell)}_{\min}$ and it is *medium* if $d(i) \geq \mu_2 \cdot s^{(\ell)}_{\min}$. Note that there are some tasks that are neither tiny nor medium, i.e., a task i with $\mu_1 \cdot s^{(\ell)}_{\min} < d(i) < \mu_2 \cdot s^{(\ell)}_{\min}$. We will neglect such tasks. Due to the following lemma, we can find values for μ_1, μ_2 such that this is justified. Also, there is a large gap between these two values which we will exploit later.

▶ **Lemma 11.** *For each $\epsilon > 0$ we can find a set of $1/\epsilon$ pairs $(\mu_1^{(1)}, \mu_2^{(1)}), ..., (\mu_1^{(1/\epsilon)}, \mu_2^{(1/\epsilon)})$ such that for one pair $(\mu_1^{(r)}, \mu_2^{(r)})$ it holds that $\mu_1 \leq \frac{\epsilon}{\alpha_\epsilon}$ with $\alpha_\epsilon := \frac{1}{\epsilon} \cdot \left(\frac{2}{\epsilon^6} + \frac{10\epsilon}{\mu_2} \cdot (1/\epsilon)^{1/\epsilon}\right)$ and the set $OPT' \cap \bigcup_\ell \{i \in \mathcal{T}^{(\ell)} | \mu_1^{(r)} \cdot s_{\min}^{(\ell)} < d(i) < \mu_2^{(r)} \cdot s_{\min}^{(\ell)}\}$ contains at most $\epsilon \cdot |OPT'|$ tasks.*

We assume that we guess the correct pair $(\mu_1^{(r)}, \mu_2^{(r)})$ and define the sets of tiny and medium tasks according to it.

3.2 Structure via segments

Next, we show that we can compute a set of at most k vertices such that the path of each input task uses one of them (otherwise we can directly find a solution with k tasks).

▶ **Lemma 12.** *In polynomial time we can identify (i) a set $\bar{V} \subseteq V$ of at most k vertices such that for each task $i \in T$ there is a vertex $v \in \bar{V}$ such that $v \in P(i)$ or (ii) a set of k tasks that form a feasible solution.*

Proof. Let i be the task with leftmost end vertex $t(i)$. We define $\bar{T} := \{i\}$ and $\bar{V} := \{t(i)\}$. We remove all tasks using $t(i)$ from the input. Note that all remaining tasks start and end on the right of $t(i)$. We iterate this process $k-1$ more times: among the remaining tasks we identify the task i' with left most end vertex $t(i')$ and we add $t(i')$ to $\bar{T}$, we add $t(i')$ to $\bar{V}$, and we remove all tasks using $t(i')$. At the end, we have that $|\bar{T}| = |\bar{V}|$ and by construction, no two tasks in $\bar{T}$ share a vertex (and thus also no edge) and each input task uses one vertex in $\bar{V}$. Hence, if $|\bar{T}| \geq k$ then we found a feasible solution with k tasks. Otherwise, the set $\bar{V}$ is the set satisfying the claim of the lemma. ◀

The vertices in $\bar{V}$ divide the path into a set of at most $k+1$ *segments* $\mathcal{S}$, i.e., any two vertices $v, v' \in \bar{V}$ such that there is no vertex of $\bar{V}$ between v and v' induce a segment $S \subseteq E$ which contains all edges between v and v'. Additionally, $\mathcal{S}$ contains a segment containing all edges between the leftmost vertex of G and the leftmost vertex in $\bar{V}$ and a segment between the rightmost vertex in $\bar{V}$ and the rightmost vertex in G. For each supertype $\mathcal{T}^{(\ell)}$ we can bound the number of huge tasks starting or ending within a segment.

▶ **Lemma 13.** *Let $\mathcal{T}^{(\ell)}$ be a supertype and let S be a segment. Then there can be at most $\frac{2}{\epsilon^6}$ huge tasks in $\mathcal{T}^{(\ell)} \cap OPT'$ that use an edge of S.*

A core problem for our algorithm is that we do not know how to allocate the edge capacities between the tiny, the medium, and the huge tasks. To this end, we prove the following lemma that will later allow us to essentially guess this allocation in FPT-time.

▶ **Lemma 14.** *By reducing the number of tasks in OPT' by at most a factor $1 + O(\epsilon)$ we can assume that for each segment $S \in \mathcal{S}$ and each supertype ℓ one of the following holds:*

- *there is no huge or medium task of supertype ℓ using any edge of S or*
- *at most α_ϵ tiny tasks of supertype ℓ start or end in S. In this case, the total demand of all tiny tasks of type j starting or ending in S is bounded by $\alpha_\epsilon \cdot \mu_1 \cdot s_{\min}^{(\ell)} \leq \epsilon \cdot s_{\min}^{(\ell)}$.*

Proof. Consider a segment S and assume that there is a huge or medium task using some edge e of S. By Lemma 13 there can be at most $\frac{2}{\epsilon^6}$ such huge tasks. Moreover, there can be at most $\frac{d_{\max}^{(\ell)}}{\mu_2 \cdot s_{\min}^{(\ell)}} \leq \frac{10\epsilon \cdot s_{\min}^{(\ell+1)}}{\mu_2 \cdot s_{\min}^{(\ell)}} \leq \frac{10\epsilon}{\mu_2} \cdot (1/\epsilon)^{1/\epsilon}$ such medium tasks and hence at most $\epsilon \cdot \alpha_\epsilon$ medium or huge tasks in total. If there are more than α_ϵ tiny tasks starting or ending in S then we remove all medium and huge tasks using an edge of S. We do this operation with

all segments S. We charge the cost of the removed huge and medium tasks to the tiny tasks. Let n' be the number of removed tasks. Then $OPT' \geq \frac{1}{2\epsilon}n'$ and thus $n' \leq 2\epsilon \cdot OPT'$. ◀

If for a segment S and a supertype ℓ the first case of Lemma 14 applies then we say that the pair (S, ℓ) is *tiny*, otherwise we say that (S, ℓ) is *huge*. As we show in the next lemma, in time FPT-time we can guess which task supertypes appear in the optimal solution.

▶ **Lemma 15.** *In time* $(\log n)^{O(k)} \leq n \cdot 2^{O(k)}$ *we can guess the set* $L = \{\ell \mid OPT \cap \mathcal{T}^{(\ell)} \neq \emptyset\}$.

Proof. Each supertype ℓ arising in the optimal solution is an integer between -1 and $\log_{1/\epsilon^2} \max_{e \in E} u(e)$. Since the input numbers are polynomially bounded this yields at most $O(\log n)$ many supertypes. For each of the k tasks in OPT' there are $O(\log n)$ options for its supertype. Thus, in time $O(\log n)^k$ we can guess all supertypes arising in OPT'. ◀

Next, we use color-coding [1] in order to guess the correct supertype of each tiny task from OPT'. More precisely, we use it in order to obtain sets $\bar{\mathcal{T}}^{(\ell)}$ for $\ell \in \mathbb{N}$ such that each tiny task $i \in OPT'$ of supertype ℓ is contained in the set $\bar{\mathcal{T}}^{(\ell)}$ (but the set $\bar{\mathcal{T}}^{(\ell)}$ possibly contains more tasks). Note that then we know the set $T \setminus \bigcup_\ell \bar{\mathcal{T}}^{(\ell)}$ which contains all medium and huge tasks in OPT'.

▶ **Lemma 16** ([1])**.** *By increasing the running time by a factor* $2^{O(k)} \cdot \log n$ *we can assume that we are given sets* $\bar{\mathcal{T}}^{(\ell)}$, $\ell \in \mathbb{N}$, *such that each tiny task* $i \in OPT'$ *of supertype* ℓ *is contained in the set* $\bar{\mathcal{T}}^{(\ell)}$.

3.3 Recursive algorithm

Denote by OPT'_T, OPT'_M, and OPT'_H the tiny, medium and huge tasks in OPT', respectively. We describe now a recursive algorithm that constructs a solution with $|OPT'|$ many tasks. We will show later how to embed it into a dynamic program that runs in FPT-time. Our algorithm proceeds in phases, each phase corresponds to one supertype ℓ. Let ℓ be the supertype of the first phase. We assume that stype$(e) \geq \ell$ for each edge e (otherwise we can reduce the instance to a set of smaller instances in which this holds).

First, in time 2^{k+1} we guess for each segment S whether (S, ℓ) is huge or tiny. For each edge $e \in S$ we allocate a certain amount of capacity $u_\ell(e)$ for the tiny tasks of supertype ℓ. A special case arises for the supertype ℓ containing the type $j = -1$. There are no tiny tasks of this supertype and we define $u_\ell(e) := 0$. Otherwise, if (S, ℓ) is huge then this means that the tiny tasks of supertype ℓ starting or ending in S have very little total capacity, at most $\epsilon \cdot s^{(\ell)}_{\min}$. However, there might be more tiny tasks that use the edges of S but do not start or end in S. Denote by x their total capacity and note that $x \leq d^{(\ell)}_{\max}$. We assign the same amount of capacity to the tiny tasks in $\bar{\mathcal{T}}^{(\ell)}$ on each edge $e \in S$. We guess the value $\bar{x} := \min\left\{d^{(\ell)}_{\max}, \left(\left\lceil x/(s^{(\ell)}_{\min} \cdot \epsilon)\right\rceil + 1\right) \cdot \epsilon \cdot s^{(\ell)}_{\min}\right\}$ and we define $u_\ell(e) := \bar{x}$ for each edge $e \in S$. There are only $O_\epsilon(1)$ many options for $\bar{x}$. Since there are at most $k+1$ segments there are only $2^{O_\epsilon(k)}$ many guesses for the huge pairs (S, ℓ).

▶ **Lemma 17.** *Let* e *be an edge of a segment* S *such that* (S, ℓ) *is huge. Then* $d(OPT'_T \cap T_e \cap \bar{\mathcal{T}}^{(\ell)}) \leq u_\ell(e)$.

Now assume that (S, ℓ) is tiny. We do not know the supertype of each edge $e \in S$. However, we know that for each edge $e \in S$ of supertype ℓ there is no huge or medium task of supertype ℓ that uses e. For each edge $e \in S$ of supertype $\ell + 1$ or larger we know that the tiny tasks of supertype ℓ use at most $d^{(\ell)}_{\max}$ units of its capacity. Therefore,

we can give to the tiny tasks of supertype ℓ the capacity of each edge $e \in S$ that is not used by the previously guessed huge tasks, up to a maximum of $d^{(\ell)}_{\max}$. We define $u_\ell(e) := \min\{u(e) - d(OPT'_{H,\geq \ell-1} \cap T_e) - (1-\epsilon)s^{(\ell)}_{\min}, d^{(\ell)}_{\max}\}$ for each edge $e \in S$ where $OPT'_{H,\geq \ell-1}$ is the set of huge tasks $i \in OPT'$ that satisfy $d(i) \geq \epsilon^2 \cdot s^{(\ell-1)}_{\min}$.

▶ **Lemma 18.** *Let ℓ be a supertype. Let e be an edge of a segment S such that (S,ℓ) is tiny. Then $d(OPT'_T \cap T_e \cap \bar{\mathcal{T}}^{(\ell)}) \leq u_\ell(e)$.*

The following lemma implies that after we assigned $u_\ell(e)$ units of capacity to the tiny tasks the remaining capacity is sufficient for the huge and medium tasks in OPT' (in particular the not yet selected ones of supertype ℓ or larger). This holds even if we assign the capacity of the tiny tasks in this manner for all supertypes $\ell' \leq \ell$.

▶ **Lemma 19.** *For each edge e of a segment S we have that $\sum_{\ell':\ell'\leq\ell} u_{\ell'}(e) + d(OPT'_H \cap T_e) + d(OPT'_M \cap T_e) \leq u(e) - \frac{1}{2} \cdot s^{(\ell)}_{\min}$ where $\ell := \text{stype}(e)$.*

For each edge e we have that $u_\ell(e) \in [\epsilon \cdot s^{(\ell)}_{\min}, d^{(\ell)}_{\max}]$, independent on whether e is in a huge or in a tiny segment. Thus, the $u_\ell(e)$ values are in a constant range. We call our FPT-algorithm for this case (see Theorem 6) with the input consisting of $\bar{\mathcal{T}}^{(\ell)}$ and the edge capacities u_ℓ. Due to Lemmas 17 and 18 it will output a solution consisting of at least $|OPT'_T \cap \bar{\mathcal{T}}^{(\ell)}|$ tasks.

Next, we want to guess the huge and medium tasks of supertype ℓ in OPT' and split the path into subpaths such that each subpath E' has the property that $\text{stype}(e') \geq \ell+1$ for each edge $e' \in E'$. First, we guess which segments S have the property that for some edge $e \in S$ we have that $\text{stype}(e) = \ell$. We can do this in time 2^{k+1}. Denote by $\mathcal{S}'$ the resulting set of segments. We process the segments in $\mathcal{S}'$ from left to right, starting with the leftmost such segment $S \in \mathcal{S}'$. We guess the leftmost and the rightmost edge in S of supertype ℓ, denote them by e_L and e_R, respectively. Then we guess the at most $O_\epsilon(1)$ medium and huge tasks of supertype ℓ that use e_L or e_R. We recurse on the subpath consisting all edges on the left of e_L. There, each edge is of supertype at least $\ell+1$. On the subpath on the right of e_R we continue splitting the remaining path into subpaths. To this end, we take the next segment $S' \in \mathcal{S}'$, guess its leftmost and rightmost edges e'_L and e'_R of supertype ℓ, and guess the $O_\epsilon(1)$ medium and huge tasks of supertype ℓ using one of them. We recurse on the subpath between e_R and e'_L, knowing that each of its edges is of supertype at least $\ell+1$. Also, there cannot be any task whose path lies strictly between e_L and e_R since each input task has to use some vertex in $\bar{V}$. We proceed with splitting the remaining segments in $\mathcal{S}'$ on the right of S'. To this end, it suffices to know e'_R and the $O_\epsilon(1)$ medium and huge tasks using it, rather than also e_L, e_R, and e'_L and the medium and huge tasks using those (apart from those that use also e'_R).

We can embed our whole recursive algorithm into a dynamic program whose running time is FPT. Here we use ideas from [17], in particular for using the slack on the edges in order to be able to "forget" some of the previously taken decisions. Crucial here is that in order to define the $u_\ell(e)$-values it is not necessary to remember all previously guessed tasks and all values $u_{\ell'}(e)$ for all $\ell' < \ell$, but only the tasks in $OPT'_{H,\geq\ell-1}$ that use the leftmost or the rightmost edge of the subpath of the respective subproblem. One can show that those can be only $O_\epsilon(1)$ many. Thus, each arising subproblem can be described by a supertype ℓ, a subpath E' of E, and the $O_\epsilon(1)$ tasks in $OPT'_{H,\geq\ell-1}$ using the leftmost or the rightmost edge of E'. This bounds the number subproblems and thus the number of DP-cells by $n^{O_\epsilon(1)} \cdot \log u_{\max}$. With an additional color coding step and some slight extensions to the above routine one can remove the assumption that the input values are polynomially bounded.

▶ **Theorem 20.** *There exists a $(1+\epsilon)$-approximation algorithm with a running time of $2^{O(k \log k)} n^{O_\epsilon(1)} \log u_{\max}$ for UFP-instances with $|OPT| = k$.*

▶ **Corollary 21.** *There is a PTAS for UFP instances that have satisfy the property that $|OPT| \leq O(\log n / \log \log n)$.*

4 W[1]-hardness

In this section we prove that UFP is W[1]-hard if the parameter k represents the number of tasks in the optimal solution.

▶ **Theorem 22.** *The unweighted Unsplittable Flow on a Path problem is W[1]-hard when parametrized by the number of tasks in the optimal solution.*

We give a reduction from the k-subset sum problem which is W[1]-hard [15]. Given a set of n values $A = \{a_1, ..., a_n\}$, a target value B and an integer k, the goal is to choose exactly k values from A that sum up to exactly B. Suppose we are given an instance of k-subset sum. First, we claim that we can assume w.l.o.g. the following properties for it.

▶ **Lemma 23.** *W.l.o.g. we can assume that there are values $\epsilon_1, ..., \epsilon_n$, not necessarily positive, such that $a_i = B/k + \epsilon_i$ for each $i \in [n]$ and that $\sum_{i=1}^{n} |\epsilon_i| < B/k$.*

We construct an instance of UFP that admits a solution with $2k$ tasks if and only if the given k-subset sum is a yes-instance. Our UFP instance has a path with $n+2$ vertices $v_0, v_1, ..., v_{n+1}$. Denote the leftmost and the rightmost edge by e_L and e_R, respectively. We define $u(e_L) = u(e_R) = B$. For all other edges e we define $u(e) := B + k \cdot \max_i |\epsilon_i|$. Assume that the values in S are ordered such that $a_1 \geq a_2 \geq ... \geq a_n$. Let $j \in [n]$. We introduce two tasks $i(j), i'(j)$ with $s(i(j)) := v_0$, $t(i(j)) := v_i$, $d(i(j)) := a_j$ $s(i'(j)) := v_i$, $t(i'(j)) = v_{n+1}$, and $d(i'(j)) := 2B/k - a_j$. See Figure 1 for a sketch.

In order to get some intuition about the constructed instance, we observe the following.

▶ **Lemma 24.** *In the constructed instance there can be no solution with more than $2k$ tasks.*

In the next lemma we show that we can construct a solution with $2k$ tasks if the given k-subset sum instance is a yes-instance: for a given set $J \subseteq [n]$ of k indices such that $\sum_{j \in J} a_j = B$ we select the tasks $i(j)$ and $i'(j)$ for each $j \in J$. One can easily verify that this yields a feasible solution.

▶ **Lemma 25.** *If the given k-subset sum instance is a yes-instance, then the constructed UFP instance has a solution with $2k$ tasks.*

Conversely, we show that if the UFP instance has a solution with $2k$ tasks then the k-subset sum instance is a yes-instance. Suppose we are given such a solution for the UFP instance. First, we establish that for each $j \in [n]$ the solution selects either both $i(j)$ and $i'(j)$ or none of these two tasks.

▶ **Lemma 26.** *Given a solution T' to the UFP instance with $2k$ tasks. Then there is a solution T'' with $2k$ tasks such that for each $j \in [n]$ we have that either $\{i(j), i'(j)\} \subseteq T''$ or $\{i(j), i'(j)\} \cap T'' = \emptyset$.*

Proof. Let j be an index such that neither $\{i(j), i'(j)\} \subseteq T'$ nor $\{i(j), i'(j)\} \cap T' = \emptyset$. First assume that $i(j) \in T'$ but $i'(j) \notin T'$. Then by construction the edge $\{v_j, v_{j+1}\}$ is used by at most $k-1$ tasks. Let j' be the smallest index greater than j such that the edge $\{v_{j'}, v_{j'+1}\}$

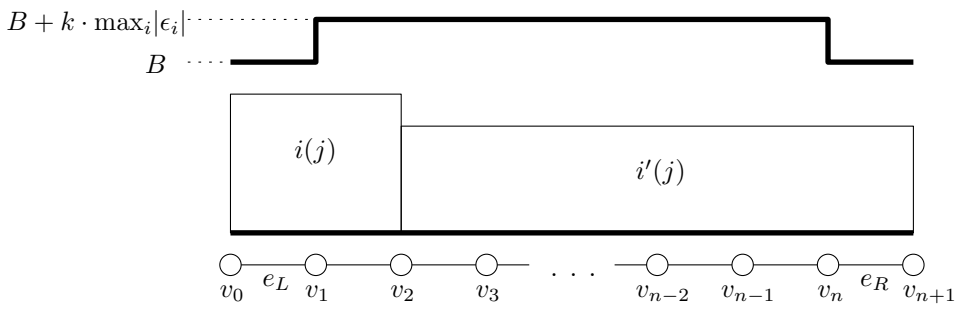

Figure 1 Sketch of the reduction used in order to prove Theorem 22. The sketch shows the tasks $i(j)$ and $i'(j)$ for only one index j.

is used by k tasks. Such an index must exist since e_R is used by k tasks from T'. Since the edge $\{v_{j'-1}, v_{j'}\}$ is used by only $k-1$ tasks this implies that $i'(j') \in T'$ but $i(j') \notin T'$. We define $\tilde{T} := T' \cup \{i'(j)\} \setminus \{i'(j')\}$. We claim that $\tilde{T}$ is feasible. The task $i'(j)$ does not use e_L and thus $\tilde{T}$ does not violate the capacity bound of e_L. Furthermore, $s(i'(j))$ lies on the left of $s(i'(j'))$ and thus $a_j \geq a_{j'}$. Hence, $d(i'(j)) = 2B/k - a_j \leq 2B/k - a_{j'} = d(i'(j'))$. Hence, $\tilde{T}$ does not violate the capacity bound of e_R. Each edge e with $e_L \neq e \neq e_R$ is used by at most k tasks. Hence, we do not violate its capacity bound (same calculation as in the proof of Lemma 25). The case that $i(j) \notin T'$ but $i'(j) \in T'$ can be handled with a similar argumentation. We repeat this process until we cannot find another index j that violates the property of the lemma. Denote by T'' the resulting set. ◀

Suppose we are given a solution T' to the UFP instance with $2k$ tasks that satisfies Lemma 26. Let J' be the set of indices j such that $i(j) \in T'$. We show in the next two lemmas that J' is a solution to the k-subset sum instance. Lemma 27 follows from our assumption that each value a_i almost equals B/k (see Lemma 23) and the fact that the edges e_L and e_R have capacity B each.

▶ **Lemma 27.** *The set T' contains exactly k tasks using e_L and exactly k tasks using e_R. Furthermore, we have that $|J'| = k$.*

▶ **Lemma 28.** *We have that $\sum_{j \in J'} a_j = B$.*

Proof. Let $T'_L \subseteq T'$ and $T'_R \subseteq T'$ denote the set of tasks in T' using e_L and e_R, respectively. Then $B = u(e_L) \geq \sum_{i \in T'_L} d(i) = \sum_{j \in J'} a_j$. On the other hand, due to Lemma 26 we have that $B = u(e_R) \geq \sum_{i \in T'_R} d(i) = \sum_{j \in J'} 2B/k - a_j$ and hence $\sum_{j \in J'} a_j \geq (\sum_{j \in J'} 2B/k) - B = B$. Therefore $\sum_{j \in J'} a_j = B$. ◀

Hence, we proved that the constructed UFP instance has a solution with $2k$ tasks if and only if the k-subset sum instance is a yes-instance. This completes the proof of Theorem 22.

▶ **Corollary 29.** *There is no EPTAS for the unweighted Unsplittable Flow on a Path problem, unless* W[1] = FPT.

Acknowledgments. The author would like to thank Tobias Mömke and Hang Zhou for helpful discussions on the topic and many comments on an earlier draft, and the anonymous referees for many helpful suggestions and comments.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, July 1995.

2 Aris Anagnostopoulos, Fabrizio Grandoni, Stefano Leonardi, and Andreas Wiese. Constant integrality gap LP formulations of unsplittable flow on a path. In *International Conference on Integer Programming and Combinatorial Optimization (IPCO 2013)*, 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. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 26–41. SIAM, 2014.

4 Nikhil Bansal, Amit Chakrabarti, Amir Epstein, and Baruch Schieber. A quasi-PTAS for unsplittable flow on line graphs. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC 2006)*, pages 721–729. ACM, 2006.

5 Nikhil Bansal, Zachary Friggstad, Rohit Khandekar, and Mohammad R. Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. In *Proceedings of the 20th Ann. ACM-SIAM Symposium on Discrete Algorithms (SODA 2009)*, pages 702–709, 2009.

6 Jatin Batra, Naveen Garg, Amit Kumar, Tobias Mömke, and Andreas Wiese. New approximation schemes for unsplittable flow on a path. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*, pages 47–58, 2015. doi:10.1137/1.9781611973730.5.

7 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.

8 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.

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

10 Chandra Chekuri, Alina Ene, and Nitish Korula. Unsplittable flow in paths and trees and column-restricted packing integer programs. Unpublished. Available at http://cs-people.bu.edu/aene/papers/ufp-full.pdf.

11 Chandra Chekuri, Alina Ene, and Nitish Korula. Unsplittable flow in paths and trees and column-restricted packing integer programs. In *APPROX-RANDOM 2009*, pages 42–55, 2009.

12 Chandra Chekuri, Marcelo Mydlarz, and F. Bruce Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Transactions on Algorithms*, 3, 2007.

13 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015.

14 Andreas Darmann, Ulrich Pferschy, and Joachim Schauer. Resource allocation with time intervals. *Theoretical Computer Science*, 411:4217–4234, 2010.

15 Michael R. Fellows and Neal Koblitz. Fixed-parameter complexity and cryptography. In *International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pages 121–131. Springer, 1993.

16 Fabrizio Grandoni, Salvatore Ingala, and Sumedha Uniyal. Improved approximation algorithms for unsplittable flow on a path with time windows. In *International Workshop on Approximation and Online Algorithms (WAOA 2015)*, pages 13–24. Springer, 2015.

17 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 *Proc. of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, 2017. To appear.

18 Dániel Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008.