On the Approximability of Unsplittable Flow on a Path with Time Windows

Alexander Armbruster[1][0009−0004−6826−398X], Fabrizio Grandoni[2][0000−0002−9676−4931], Edin Husić[2][0000−0002−6708−5112], Antoine Tinguely[2][0009−0000−7321−5457], and Andreas Wiese[1][0000−0003−3705−016X]

[1] Technical University of Munich, Munich, Germany
{alexander.armbruster,andreas.wiese}@tum.de
[2] USI-SUPSI, IDSIA, Lugano, Switzerland
{fabrizio,antoine.tinguely}@idsia.ch, edinehusic@gmail.com

Abstract. In the Time-Windows Unsplittable Flow on a Path problem (twUFP) we are given a resource whose available amount changes over a given time interval (modeled as the edge-capacities of a given path G) and a collection of tasks. Each task is characterized by a demand (of the considered resource), a profit, an integral processing time, and a time window. Our goal is to compute a maximum profit subset of tasks and schedule them non-preemptively within their respective time windows, such that the total demand of the tasks using each edge e is at most the capacity of e.

We prove that twUFP is APX-hard which contrasts the setting of the problem without time windows, i.e., Unsplittable Flow on a Path (UFP), for which a PTAS was recently discovered [Grandoni, Mömke, Wiese, STOC 2022]. Then, we present a quasi-polynomial-time $2+\varepsilon$ approximation for twUFP under resource augmentation. Our approximation ratio improves to $1+\varepsilon$ if all tasks' time windows are identical. Our APX-hardness holds also for this special case and, hence, rules out such a PTAS (and even a QPTAS, unless $\mathsf{NP} \subseteq \mathrm{DTIME}(n^{\mathrm{poly}(\log n)})$) *without* resource augmentation.

Keywords: Approximation algorithm · APX-hardness · Quasi-polynomial algorithm · Scheduling · Unsplittable Flow on a Path.

1 Introduction

In the well-studied Unsplittable Flow on a Path problem (UFP), we are given a path graph $G = (V, E)$ with m edges, a capacity $u(e) \in \mathbb{N}$ for each edge $e \in E$, and a collection T of n tasks. Each task $i \in T$ is characterized by a *weight* (or *profit*) $w(i) \in \mathbb{N}$, a *demand* $d(i) \in \mathbb{N}$, and a subpath $P(i) \subseteq E$ of G.[3] A feasible solution consists of a subset of tasks $S \subseteq T$ such that $\sum_{i \in S: e \in P(i)} d(i) \leq u(e)$ for each $e \in E$. In other words, the total demand of the tasks in S whose subpath contains e does not exceed the capacity of e. Our goal is to compute a feasible

[3] Given a subpath P of G, we will sometimes use P also to denote the corresponding set of edges $E(P)$. The meaning will be clear from the context.

solution OPT of maximum profit $w(\mathsf{OPT}) \coloneqq \sum_{i \in \mathsf{OPT}} w(i)$. One can naturally interpret G as a time interval subdivided into time slots (the edges). At each time slot, a given amount of a considered resource (e.g., energy) is available. Each task i corresponds to a job that we can execute (or not) in a fixed time interval: if i is executed, it consumes a fixed amount of the considered resource during its entire execution and generates a profit of $w(i)$. UFP is strongly NP-hard [8,11], and a lot of attention was devoted to the design of approximation algorithms for it [1,2,3,6,8,14,16,17,18], culminating in a recent PTAS for the problem [15].

In UFP, we have no flexibility for the time interval during which each selected task i is executed. In practice, it makes sense to consider scenarios where i has a given length (or processing time) and a *time window* during which it needs to be executed. In this setting, for each selected task i we need to specify a starting time for i such that i is processed completely within its time window. This leads to the Time-Windows UFP problem (twUFP). Here, we are given the same input as in UFP, with the difference that for each task i, instead of a subpath $P(i)$ we are given a *length* (or *processing time*) $p(i) \in \{1, \dots, m\}$, and a subpath $\mathrm{tw}(i) \subseteq E$ with at least $p(i)$ edges (the *time window* of i). A *scheduling* of i is a subpath $P(i) \subseteq \mathrm{tw}(i)$ containing precisely $p(i)$ edges. A feasible solution for the given instance is a pair $(S, P(\cdot))$ such that for each $i \in S$ the path $P(i)$ is a feasible scheduling of i and $\sum_{i \in S : e \in P(i)} d(i) \leq u(e)$ for each $e \in E$. Our goal is to maximize $w(S)$, like in UFP. Observe that UFP is the special case of twUFP where for each task $i \in T$ the time window $\mathrm{tw}(i)$ contains exactly $p(i)$ edges; hence, twUFP is strongly NP-hard. The best known approximation algorithm for it is a $O(\log n / \log \log n)$-approximation [13], improving on a prior result in [9] (both results hold for a more general problem, bagUFP, which is defined later). However, no result in the literature excludes the existence of a much better approximation ratio for twUFP, including possibly a PTAS. In this paper, we make progress on a better understanding of the approximability of twUFP.

1.1 Our Results and Techniques

Our first main result is that twUFP does *not* admit a PTAS, which in particular implies that it is strictly harder than UFP (unless $\mathsf{P} = \mathsf{NP}$). We show that this already holds when the time window of each task spans all the edges of G, i.e., if $\mathrm{tw}(i) = E$ for each $i \in T$; we denote this special case of the problem by Spanning UFP (spanUFP). Our result even holds in the cardinality setting, i.e., when all tasks have unit weight, and for polynomially bounded input data.

Theorem 1. spanUFP *(thus also* twUFP*) is* APX*-hard and does not admit a (polynomial-time)* $\frac{2755}{2754}$*-approximation algorithm (unless* $\mathsf{P} = \mathsf{NP}$*), even in the cardinality case and if demands and the number of edges is polynomially bounded in* n*.*

The proof of Theorem 1 follows from a reduction from the Maximum 3-Dimensional Matching (3-DM) problem and the bound in [10]. In our reduction,

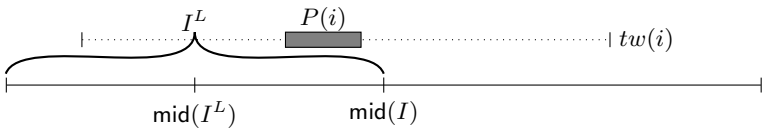

Fig. 1: The interval I and its subdivision together with a task i.

we model 3-DM instances as instances of SPANUFP in which the capacity profile models bins, reminiscent of the 2-Dimensional Vector Bin Packing (2-VBP) problem, and adapt the construction in [23] (which refutes the existence of an asymptotic PTAS for 2-VBP).

Our second main contribution is a constant factor approximation algorithm for TWUFP with resource augmentation which runs in quasi-polynomial time. More specifically, for any constant $\varepsilon > 0$, we compute a $(2+\varepsilon)$-approximate solution in time $2^{O_\varepsilon(\mathrm{poly}(\log(nm)))}$ which violates the edge capacities at most by a factor $1 + O(\varepsilon)$. As usual in the setting of resource augmentation, the approximation ratio is computed with respect to an optimal solution which is *not* allowed to violate the edge capacities.

Theorem 2. *For any $\varepsilon > 0$, there is a $(2+\varepsilon)$-approximation for* TWUFP *under $(1 + O(\varepsilon))$-resource augmentation with running time $O\big((mn)^{O(\log^2 n \log^5 m/\varepsilon^4)} \cdot \log U\big)$, where $U := \max_{e\in E} u(e)$ is the maximum capacity of the instance.*

For the special case of SPANUFP, the approximation ratio of our algorithm improves to $1 + \varepsilon$, i.e., we achieve a quasi-PTAS (QPTAS) under resource augmentation for SPANUFP. In contrast (unless $\mathsf{NP} \subseteq \mathrm{DTIME}(n^{\mathrm{poly}(\log n)})$), a QPTAS for SPANUFP is impossible *without* resource augmentation since SPANUFP is APX-hard.

Theorem 3. *For any $\varepsilon > 0$ there is a QPTAS for* SPANUFP *under $(1+O(\varepsilon))$-resource augmentation.*

We leave as an interesting open problem to find a polynomial-time (or even quasi-polynomial-time) constant factor approximation for TWUFP *without* resource augmentation.

We describe now the key ideas in our $(2+\epsilon)$-approximation for TWUFP. The previous QPTASes for UFP [2,6] are based on the following approach (written in slightly different terminology here). We interpret the path E as an interval I, i.e., with one subinterval of unit length for each edge $e \in E$. We consider its midpoint which we denote by $\mathsf{mid}(I)$ and all input tasks i whose path $P(i)$ contains $\mathsf{mid}(I)$. These tasks are partitioned into polylogarithmically many groups such that within each group, all tasks have roughly the same weight and demand (up to a factor of $1 + \epsilon$). For each group, we guess an estimate for the amount of capacity they use in the optimal solution on each edge in E. Given this, we compute the most profitable set of tasks for which these edge capacities are sufficient. Then, the remaining problem splits into two *independent* subproblems: one for all input tasks whose paths lie on the left of $\mathsf{mid}(I)$ and one for all input

tasks whose paths lie on the right of $\mathsf{mid}(I)$. Therefore, we can easily recurse on these two subproblems and solve them independently.

Unfortunately, this approach does not extend to TWUFP. A natural adaption for TWUFP would be to consider all input tasks i for which the path $P(i)$ *in the optimal solution* contains the midpoint of I which we denote by $\mathsf{mid}(I)$. However, then the remaining problem does *not* split nicely into two independent subproblems: there can be a task i whose time window $\mathrm{tw}(i)$ contains $\mathsf{mid}(I)$ and which we could schedule entirely on the left of $\mathsf{mid}(I)$ or entirely on the right of $\mathsf{mid}(I)$ (see Figure 1). Thus, i appears in the input of the left *and* the right subproblem, even though we are allowed to select i only once. Hence, these subproblems are no longer independent. Even more, this issue for i can arise again in each level of the recursion, which yields many interconnected subproblems.

Therefore, we use a different approach to obtain a $(2+\epsilon)$-approximation for TWUFP. We consider all input tasks i whose *time window* contains $\mathsf{mid}(I)$. By losing a factor of 2 on the weight of these tasks in the optimal solution, we can sacrifice either all of them that are scheduled on the left of $\mathsf{mid}(I)$ or all of them that are scheduled on the right of $\mathsf{mid}(I)$. We guess which case applies and we assume it in the following to be the latter case w.l.o.g. For the tasks i for which $P(i)$ contains $\mathsf{mid}(I)$ in the optimal solution we guess an estimate for the amount of capacity they use on the edges, similar to the QPTASs for UFP. Then we recurse on the subproblems corresponding to the left and the right of $\mathsf{mid}(I)$. For the right subproblem, we do not allow to select tasks whose time window uses $\mathsf{mid}(I)$. Therefore, our two subproblems are now independent! Let us consider the left subproblem, denote its corresponding interval by I^L and its midpoint by $\mathsf{mid}(I^L)$. If we continued natively in the same fashion, we would lose another factor of 2 on the weight of the tasks whose time windows contain $\mathsf{mid}(I)$ and $\mathsf{mid}(I^L)$, which we cannot afford. Instead, we employ a crucial new idea. We partition the mentioned tasks into polylogarithmically many groups such that within each group, all tasks have the same demand (using resource augmentation) and roughly the same weight (up to a factor of $1+\epsilon$). Since the time window of each such task i contains $\mathsf{mid}(I)$ and $\mathsf{mid}(I^L)$, we can schedule it freely within the interval $[\mathsf{mid}(I^L), \mathsf{mid}(I)]$. Therefore, for each group, we consider its tasks that are scheduled entirely during $[\mathsf{mid}(I^L), \mathsf{mid}(I)]$ in the optimal solution and round their processing times via linear grouping [21,22] to $O(1/\epsilon)$ different values. For each resulting combination of demand, weight, and rounded processing time, we guess for the right subproblem of I^L (i.e., the "left-right subproblem") how many tasks it schedules and give it these tasks as part of the input. To the left subproblem of I^L (i.e., the "left-left subproblem") we give the additional constraint that it needs to leave the corresponding number of tasks from each group unassigned. An important aspect is that the processing times of these tasks are *not* rounded in the left-left subproblem, but they *are* rounded in the left-right subproblem. This is a crucial difference of our approach in comparison to other rounding methods from the literature. Thanks to this technique, we avoid to lose another factor of 2 on the mentioned tasks. We apply it recursively for $O(\log m)$ levels and obtain an approximation ratio of $2+\epsilon$ overall.

1.2 Related Work

The BAGUFP problem is a generalization of UFP where we are given the same input as in UFP plus a partition of the tasks into subsets, the *bags*, and we are allowed to select at most one task per bag. This also generalizes TWUFP by creating bags that model each possible way to schedule a task within its time window. It was already known that BAGUFP is APX-hard [20] (which is now also implied by our result). The current best approximation ratio for BAGUFP is $O(\log n/\log\log n)$ [13] (and 17 under the no-bottleneck assumption [9]). A constant factor approximation algorithm for BAGUFP (hence for TWUFP) is known for the cardinality case of the problem [13].

For the special case of TWUFP when all the demands and capacities are 1 (also known as the non-preemptive throughput maximization problem), the best-known approximation factor for this problem is 2 [4,5,7]. Notice that we give a $(2+\varepsilon)$-approximation for a much more general problem, however, in quasi-polynomial time and using resource augmentation. In the cardinality case of the problem, there is an algorithm with an approximation ratio of $e/(e-1)+\varepsilon$ [12] (also for bags instead of time windows), which was later slightly improved in [19].

2 A $(2+\varepsilon)$-approximation for TWUFP

In this section, we present a $(2+\varepsilon)$-approximation algorithm with a quasi-polynomial running time for TWUFP under $(1+\varepsilon)$-resource augmentation, for any constant $\varepsilon > 0$. W.l.o.g. assume $1/\varepsilon \in \mathbb{N}$. Formally, let $(\mathsf{OPT}, P^*(\cdot))$ denote an optimal solution for the given instance. We compute a solution $(S, P(\cdot))$ consisting of a set of tasks S and for each task $i \in S$ a subpath $P(i) \subseteq \mathrm{tw}(i)$ of length $p(i)$ such that $w(\mathsf{OPT}) \leq (2+\varepsilon)w(S)$ and $\sum_{i\in S: e\in P(i)} d(i) \leq (1+\varepsilon)u(e)$ for every $e \in E$.

First, we use resource augmentation in order to round the edge capacities and task demands. We also round the tasks' weights.

Lemma 1. *Let $\alpha \geq 1$. Assume that there is an α-approximation algorithm running in time $T(n,\varepsilon)$ for the special case of* TWUFP *in which*

- *for each $e \in E$ we have that $u(e) \in [1, (n/\varepsilon)^{1/\varepsilon})$*
- *for each task $i \in T$ we have that*
  - *$d(i) \in [1, (n/\varepsilon)^{1/\varepsilon})$ and $d(i)$ is a power of $1+\varepsilon$, and*
  - *$w(i) \in [1, n/\varepsilon]$ and $w(i)$ is a power of $1+\varepsilon$.*

Then there is a $(1+5\varepsilon)\alpha$-approximation algorithm for TWUFP *under $(1+4\varepsilon)$-resource augmentation with a running time of $O(\mathrm{poly}(n)\frac{\log U}{\log n}T(n,\varepsilon))$, where $U = \max_{e\in E} u(e)$.*

In the following, we will present a $(2+\varepsilon)$-approximation algorithm for the special case of TWUFP defined in Lemma 1; this yields a $(2+O(\varepsilon))$-approximation for arbitrary instances of TWUFP under $(1+O(\varepsilon))$-resource augmentation.

We first define a hierarchical decomposition of E. Assume w.l.o.g. that $m = |E|$ is a power of 2. We define a (laminar) family of subpaths $\mathcal{I}$ which intuitively

form a tree. We will refer them also as *intervals*. There is one interval $I_r \in \mathcal{I}$ such that $I_r = E$ which intuitively forms the root of the tree. For each $I \in \mathcal{I}$, we will ensure that $|I|$ is a power of 2. Consider an interval $I \in \mathcal{I}$ with $|I| \geq 2$ and let $\mathsf{mid}(I)$ denote its middle vertex. Given I, we define recursively that there is an interval $I^L \in \mathcal{I}$ induced by the edges of I to the left of $\mathsf{mid}(I)$, and an interval $I^R \in \mathcal{I}$ induced by the edges of I to the right of $\mathsf{mid}(I)$. We say that I^L and I^R are the two *children* of I. We apply this definition recursively. As a result, for each edge $e \in E$ there is an interval $\{e\} \in \mathcal{I}$; such intervals form the leaves of our tree. We say that an interval $I \in \mathcal{I}$ is of *level* ℓ for some $\ell \in \mathbb{N}_0$ if $|I| = 2^{\log m - \ell}$, e.g., the interval I_r is of level 0. Note that we have $1 + \log m$ levels (for which there is an interval in $\mathcal{I}$).

We group the tasks into a polylogarithmic number of groups. For each task $i \in T$ we define that it is of *level* $\ell(i)$ if $\mathrm{tw}(i)$ is contained in an interval $I \in \mathcal{I}$ of level $\ell(i)$ but not in an interval $I' \in \mathcal{I}$ of level $\ell(i)+1$. We define $T_\ell := \{i \in T : \ell(i) = \ell\}$ for each integer ℓ. Also, for each combination of values w, d such that w represents a weight and d a demand, we define the set $T_{w,d} = \{i \in T : w(i) = w \wedge d(i) = d\}$. Note that there are only a polylogarithmic number of combinations for these pairs w, d for which $T_{w,d} \neq \emptyset$ by Lemma 1.

Recall that $(\mathsf{OPT}, P^*(\cdot))$ denotes an optimal solution. Consider a task $i \in \mathsf{OPT}$ of some level $\ell(i)$. Let I be the (unique) interval of level $\ell(i)$ that contains $\mathrm{tw}(i)$ and let I^L and I^R denote the children of I. Observe that $P^*(i)$ must satisfy one of the following three conditions:

1. $P^*(i)$ is contained in I^L, or
2. $\mathsf{mid}(I)$ is in the interior of $P^*(i)$, i.e., $\mathsf{mid}(I)$ is a vertex of $P^*(i)$ but it is neither the leftmost nor the rightmost vertex of $P^*(i)$, or
3. $P^*(i)$ is contained in I^R.

Let $\mathsf{OPT}^{(L)}, \mathsf{OPT}^{(M)}$, and $\mathsf{OPT}^{(R)}$ denote the tasks in OPT for which case 1, 2, and 3 applies, respectively. In particular, $w(\mathsf{OPT}) = w(\mathsf{OPT}^{(M)}) + w(\mathsf{OPT}^{(L)}) + w(\mathsf{OPT}^{(R)})$ and thus $w(\mathsf{OPT}^{(L)}) \leq \frac{1}{2}w(\mathsf{OPT})$ or $w(\mathsf{OPT}^{(R)}) \leq \frac{1}{2}w(\mathsf{OPT})$. The first step of our algorithm is to guess which of these cases applies. Assume w.l.o.g. that $w(\mathsf{OPT}^{(R)}) \leq \frac{1}{2}w(\mathsf{OPT})$. In the following, we will essentially aim to select only tasks from $\mathsf{OPT}^{(L)} \cup \mathsf{OPT}^{(M)}$. In this step we lose a factor of (up to) 2 on the obtained profit.

Formally, we say that a solution $(S, P(\cdot))$ is *left constrained* if the following holds for each task $i \in S$: For I being the unique interval of level $\ell(i)$ containing $\mathrm{tw}(i)$, we require that $P(i)$ is not contained in the right child of I, i.e., in the subinterval of I containing all edges of I on the right of $\mathsf{mid}(I)$. We will later define artificial tasks (not contained in T) for which this is not required. Our algorithm will return a left constrained solution by construction and we will compare its profit with the left constrained solution $\mathsf{OPT}^{(L)} \cup \mathsf{OPT}^{(M)}$.

Our algorithm is recursive. We start by describing the root subproblem in the next subsection; it is simpler than a generic subproblem, however it is convenient to introduce certain notions that will be needed also in the general case. In the subsequent subsection, we will describe a generic subproblem.

2.1 Root subproblem

The root subproblem corresponds to the interval $I_r = E$ (of level 0). Let $(\overline{\mathsf{OPT}}_r, P^*(\cdot))$ be an optimal left constrained solution. Let $\overline{\mathsf{OPT}}_r^{(M)}$ be all the tasks $i \in \overline{\mathsf{OPT}}_r$ such that $\mathsf{mid}(I_r)$ is in the interior of $P^*(i)$. Consider a group $T_{w,d}$. We want to guess a set of *boxes* that delimit space that we reserve for tasks in $T_{w,d} \cap \overline{\mathsf{OPT}}_r^{(M)}$. Formally, a box b is characterized by a path $\mathrm{pb}(b) \subseteq E$, a height $h(b) > 0$, a demand $d(b) > 0$, and a weight $w(b) > 0$. We will consider only boxes b for which $w(b)$ is some task weight, $d(b)$ is some task demand, and $h(b)$ is some integer multiple of $d(b)$ upper bounded by $n \cdot d(b)$. In the following, we will assign certain sets of tasks to some boxes such that, intuitively, these tasks are stacked on top of each other inside the box and each of them has a weight of $w(b)$, a demand of $d(b)$, and their total demand is at most $h(b)$, i.e., they are at most $h(b)/d(b)$ many. Furthermore, it is possible to schedule each such task within the path of the box.

More explicitly, for any set of tasks S, using the notation $d(S) \coloneqq \sum_{i \in S} d(i)$ and $w(S) \coloneqq \sum_{i \in S} w(i)$, we say that S *fits inside a box* b if

- $d(S) \leq h(b)$,
- $|\operatorname{tw}(i) \cap \mathrm{pb}(b)| \geq p(i)$ for each $i \in S$, and
- $w(i) = w(b)$ and $d(i) = d(b)$ for each $i \in S$.

We use the next lemma to show that there is a set of constantly many boxes $B_{w,d}$ in which almost all tasks in $T_{w,d} \cap \overline{\mathsf{OPT}}_r^{(M)}$ fit, i.e., up to a factor of $1 + \varepsilon$ in the profit. Also, the capacity used by these boxes is at most the capacity used by the tasks in $T_{w,d} \cap \overline{\mathsf{OPT}}_r^{(M)}$ on each edge. Formally, we apply the following lemma to the set $T_{w,d} \cap \overline{\mathsf{OPT}}_r^{(M)}$ and obtain a set of at most $1/\varepsilon^2$ boxes that we denote by $B_{w,d}$. We prove this lemma with a similar linear grouping scheme as in [21] (see also [22]) to show that we can approximate the demand profile of such a set of tasks by a simpler profile with a constant number of steps only.

Lemma 2. *Let T' be a set of tasks, all having identical weight w and identical demand d, and a schedule $P(i) \subseteq E$ for every $i \in T'$ such that there is an edge f that is contained in each path $P(i)$. Then, there exists a set $S \subseteq T'$, a set of boxes B with $|B| \leq 1/\varepsilon^2$, and a partition of S into sets $\{S_b\}_{b \in B}$ such that:*

(1) $\sum_{b \in B : e \in \mathrm{pb}(b)} h(b) \leq \sum_{i \in T' : e \in P(i)} d(i)$ *for each* $e \in E$,
(2) $w(S) \geq (1 - O(\varepsilon)) w(T')$,
(3) the tasks in S_b fit into b for each $b \in B$.

We guess the at most $1/\varepsilon^2$ boxes $B_{w,d}$. For each box $b \in B_{w,d}$, let $S_b \subseteq T_{w,d} \cap \overline{\mathsf{OPT}}_r^{(M)}$ be the subset of $T_{w,d} \cap \overline{\mathsf{OPT}}_r^{(M)}$ that is assigned to b when applying Lemma 2 to $T_{w,d} \cap \overline{\mathsf{OPT}}_r^{(M)}$; we guess $|S_b| =: n_b$. We do this procedure for the set $T_{w,d}$ for each combination of w and d. Let B_0 denote the set of all boxes that we guessed this way.

Let I_r^L and I_r^R be the two children of I_r. We recurse on a *left subproblem* corresponding to I_r^L and on a *right subproblem* corresponding to I_r^R. In the right

subproblem, we are given as input the set consisting of each task $i \in T$ such that $\mathrm{tw}(i) \subseteq I_r^R$. Note that such a task must have level 1 or larger, so the tasks of level 0 are not considered in the subproblem corresponding to I_r^R. The capacity of each edge $e \in I_r^R$ is defined as $u'(e) \coloneqq u(e) - \sum_{b \in B_0 : e \in \mathrm{pb}(b)} h(b)$, that is, the boxes in B_0 use a certain amount of the capacities of the edges that we cannot use. The goal is to compute a feasible solution to this smaller instance, i.e., a set of tasks Q^R with a path $P(i) \subseteq \mathrm{tw}(i) \subseteq I_r^R$ for each task $i \in Q^R$ that respects the edge capacities $u'(e)$. The objective is to maximize $w(Q^R)$.

In the left subproblem, we are given as input the set of all tasks $i \in T$ such that $\mathrm{tw}(i) \subseteq I_r^L$ or $\ell(i) = 0$; let T^L denote this set of tasks. The capacity of each edge $e \in I_r^L$ is defined as $u'(e) \coloneqq u(e) - \sum_{b \in B_0 : e \in \mathrm{pb}(b)} h(b)$ analogously to the right subproblem. Also, we are given the set of boxes B_0 and a value n_b for each box b. The goal is to select a subset $Q \subseteq T^L$, a partition of Q into a set Q^L and a set Q_b for each box $b \in B_0$, and a path $P(i)$ for each task $i \in Q$ such that

- the set Q^L with the path $P(i) \subseteq \mathrm{tw}(i) \cap I_r^L$ for each task $i \in Q^L$ forms a feasible solution for the given edge capacities $u'(e)$, i.e., for each edge $e \in I_r^L$ we have $\sum_{i \in Q^L : e \in P(i)} d(i) \leq u'(e)$, and
- for each box $b \in B_0$ we have that $|Q_b| = n_b$, Q_b fits into b, and $P(i) \subseteq \mathrm{tw}(i) \cap \mathrm{pb}(b)$ for each $i \in Q_b$.

Observe that the given boxes B_0 do not interact at all with the edges I_r^L and their given capacities. However, the subpath $\mathrm{pb}(b)$ for a box $b \in B_0$ is important since a task i fits into a box b only if $|\mathrm{tw}(i) \cap \mathrm{pb}(b)| \geq p(i)$.

Given a solution to the left and the right subproblems, we combine them to a solution to the overall problem in the obvious way: the solution is given by the tasks $Q = Q^L \cup Q^R \cup (\cup_{b \in B_0} Q_b)$ with the respective paths $P(i)$. Each of our quasi-polynomially many guesses yields a candidate solution; among all feasible candidate solutions, we output the solution with maximum profit $w(Q)$.

2.2 Arbitrary subproblems

In the following, we describe our routine for arbitrary subproblems. The reader may think of the subproblem for the left child of the root I_r, i.e., the interval I_r^L (which already incorporates all the challenges of a generic subproblem). We assume that the input consists of

- an interval $I \in \mathcal{I}$ of level ℓ with a capacity $u'(e)$ for each edge $e \in I$,
- a set of tasks T', and
- a set of boxes B and a value n_b for each box $b \in B$.

We remark that, for a box $b \in B$, not necessarily $\mathrm{pb}(B) \subseteq I$. Furthermore, given a task $i \in T'$, it might happen that $\mathrm{tw}(i) \not\subseteq I$; in this case, we will ensure that $\mathrm{tw}(i)$ contains the rightmost edge of I by construction. We remark that it might happen that $i \notin T$, namely i is not one of the input task. This happens because during the process, we introduce some *artificial tasks* whose role will become clear later. Like the regular tasks, each artificial task i has a weight $w(i)$, a demand $d(i)$, a length $p(i)$, and a time window $\mathrm{tw}(i)$.

Let I^L and I^R denote the two children of I and let $\mathsf{mid}(I)$ denote the middle vertex of I. The goal is to compute a set of tasks $Q \subseteq T'$, a partition of Q into sets Q_I and a set Q_b for each box $b \in B$, and a path $P(i)$ for each task $i \in Q$ such that

- the tasks in Q_I with their paths $P(i) \subseteq \mathrm{tw}(i) \cap I$ obey the edge capacities of I, i.e., for each edge $e \in I$ we have $\sum_{i \in Q_I : e \in P(i)} d(i) \leq u'(e)$,
- for each box $b \in B$ the tasks in Q_b fit into B, $|Q_b| = n_b$, and $P(i) \subseteq \mathrm{tw}(i) \cap \mathrm{pb}(b)$ for each $i \in Q_b$.

The objective is to maximize $w(Q_I)$, i.e., the weight of the tasks in Q_I. Note that the tasks assigned to the boxes B do not yield any profit in this subproblem. Intuitively, we accounted the profit of such tasks already in subproblems of higher levels in the recursion.

Let $(\overline{\mathsf{OPT}}, P^*(\cdot))$ denote an optimal left constrained solution to the given subproblem. We denote by $\overline{\mathsf{OPT}}_I$ and $\{\overline{\mathsf{OPT}}_b\}_{b \in B}$ the corresponding partition of $\overline{\mathsf{OPT}}$ (i.e., $\overline{\mathsf{OPT}}_I$ consists of the tasks in $\overline{\mathsf{OPT}}$ scheduled on I and $\overline{\mathsf{OPT}}_b$ contains the task in $\overline{\mathsf{OPT}}$ that are placed inside b). The base cases of our recursion are there subproblems where $|I| = 1$. In this case, we can solve the subproblem exactly in quasi-polynomial time. We guess for each combination of a weight w and a demand d how many tasks with this weight and demand are contained in $\overline{\mathsf{OPT}}_I$. Then, the remaining problem can be reduced easily to an instance of b-matching.

Lemma 3. *Any subproblem with $|I| = 1$ can be solved exactly in $n^{O(WD)}$ time, where W and D denote the number of distinct weights and demands, respectively.*

Assume now that $|I| > 1$. For each task $i \in \overline{\mathsf{OPT}}_I$ there are three possibilities:

1. $P^*(i)$ is contained in I^L, or
2. $\mathsf{mid}(I)$ is in the interior of $P^*(i)$, i.e., $\mathsf{mid}(I)$ is a vertex of $P^*(i)$ but it is neither the leftmost nor the rightmost vertex of $P^*(i)$, or
3. $P^*(i)$ is contained in I^R.

Let $\overline{\mathsf{OPT}}^{(L)}, \overline{\mathsf{OPT}}^{(M)}$, and $\overline{\mathsf{OPT}}^{(R)}$ denote the tasks in $\overline{\mathsf{OPT}}_I$ for which case 1, 2 and 3 applies, respectively. Note that the partition of $\overline{\mathsf{OPT}}$ into $\overline{\mathsf{OPT}}^{(L)}, \overline{\mathsf{OPT}}^{(M)}$, and $\overline{\mathsf{OPT}}^{(R)}$ is different from the partition of OPT into $\mathsf{OPT}^{(L)}, \mathsf{OPT}^{(M)}$, and $\mathsf{OPT}^{(R)}$, since the former is with respect to $\mathsf{mid}(I)$ for each task $i \in \overline{\mathsf{OPT}}$ of *any* level. Recall also that $\overline{\mathsf{OPT}}$ is left constraint and, hence, assuming that I is an interval of level ℓ, for no task i of level ℓ, its scheduled path $P^*(I)$ in the optimal solution is contained in $\overline{\mathsf{OPT}}^{(R)}$. More precisely, for each task $i \in \overline{\mathsf{OPT}}^{(R)}$, it must be the case that either $\mathrm{tw}(i) \subseteq I^R$ (so i is of level $\ell + 1$ or deeper), $I^R \subseteq \mathrm{tw}(i)$ (so i is of level $\ell - 1$ or higher) or the leftmost edge of $\mathrm{tw}(i)$ is in I^R but not all edges of $\mathrm{tw}(i)$ (so i is of level $\ell - 1$ or higher).

First, we guess boxes for the tasks in $\overline{\mathsf{OPT}}^{(M)}$. Formally, for each combination of a weight w and demand d, we apply Lemma 2 to the set $\{i \in \overline{\mathsf{OPT}}^{(M)} :$

$w(i) = w \wedge d(i) = d\}$. Let $\bar{B}_{w,d}$ denote the resulting set of boxes and for each box $b \in \bar{B}_{w,d}$ let $\overline{\mathsf{OPT}}_b^{(M)}$ be the resulting set of tasks. We guess the boxes in $\bar{B}_{w,d}$ and $|\overline{\mathsf{OPT}}_b^{(M)}| =: n_b$ for each box $b \in \bar{B}_{w,d}$. Let $\bar{B}$ denote the union of all sets of boxes $\bar{B}_{w,d}$ that we guessed in this way.

Let $\overleftarrow{T}' \subseteq T'$ denote all tasks in $i \in T'$ such that $I^R \subseteq \mathrm{tw}(i)$ and $I^L \cap \mathrm{tw}(i) \neq \emptyset$, i.e., task whose time windows stick into I from the right and intersect I^L. For the subproblem for I_r^L, the reader may imagine that $\overleftarrow{T}'$ are all tasks in T_0 whose time windows start in the left child of I_r^L. For a task $i \in \overleftarrow{T}' \cap \overline{\mathsf{OPT}}_I$ it is possible that $i \in \overline{\mathsf{OPT}}^{(L)}$ or that $i \in \overline{\mathsf{OPT}}^{(R)}$. Therefore, at first glance it would make sense to pass i to our subproblem for I^L and to our subproblem for I^R. However, we must avoid that our subproblem for I^L and our subproblem for I^R both select i. On the other hand, we do not want to sacrifice a factor of 2 by, e.g., omitting $\overline{\mathsf{OPT}}^{(L)}$ or $\overline{\mathsf{OPT}}^{(R)}$. For example, if $\overleftarrow{T}' = T_0$ in the subproblem for I_r^L, then this would lose a second factor of 2 on the profit of the tasks in T_0, while we want to bound our approximation ratio by $2 + O(\varepsilon)$.

We execute the following steps instead:

I. Perform a linear grouping step in which we round up the task lengths so that there are at most polylogarithmically many distinct lengths in $\overleftarrow{T}' \cap \overline{\mathsf{OPT}}^{(R)}$;
II. For each combination of a demand, weight and (rounded) length, guess how many such tasks are contained in $\overleftarrow{T}' \cap \overline{\mathsf{OPT}}^{(R)}$;
III. Give a suitable number of these rounded tasks as *artificial input tasks* to our right subproblem for I^R;
IV. Enforce our left subproblem for I^L to leave sufficiently many tasks from $\overleftarrow{T}'$ unassigned such that we can schedule these tasks later in the places in which the solution to the right subproblem schedules the artificial tasks (we model this requirement for I^L via suitable additional box constraints).

Formally, for each combination of a weight w and demand d we apply the following lemma to the set $\overleftarrow{T}'_{w,d} = \left\{i \in \overleftarrow{T}' \cap \overline{\mathsf{OPT}}^{(R)} : w(i) = w \wedge d(i) = d\right\}$.

Lemma 4. *Let T' be a set of tasks, all having identical weight w and identical demand d, and a schedule $P'(i) \subseteq I \subseteq \mathrm{tw}(i)$ for every $i \in T'$. Then, there exists a set of* artificial tasks *$\tilde{T}$ (not contained in T), a set of boxes $\tilde{B}$ and a value $n_b \in \mathbb{N}$ for each box $b \in \tilde{B}$ with $\sum_{b \in \tilde{B}} n_b = |\tilde{T}|$ such that*

(1) $(1-\varepsilon)|T'| \leq |\tilde{T}| \leq |T'|$, and the tasks in $\tilde{T}$ have at most $1/\varepsilon$ distinct lengths,
(2) $\mathrm{tw}(i) = I$, $d(i) = d$, and $w(i) = w$ for each task $i \in \tilde{T}$,
(3) there is a solution $(\tilde{T}, \tilde{P}(\cdot))$ for $\tilde{T}$ such that $\sum_{i \in \tilde{T} : e \in \tilde{P}(i)} d(i) \leq \sum_{i \in T' : e \in P'(i)} d(i)$ for each edge $e \in I$,
(4) there is a collection $\{Q'_b\}_{b \in \tilde{B}}$ of pairwise disjoint subsets of T' such that for each $b \in \tilde{B}$ the set Q'_b fits into b and $|Q'_b| = n_b$,
(5) given a combination of a solution $(\tilde{S}, \tilde{P}(\cdot))$ for a subset $\tilde{S} \subseteq \tilde{T}$ and any collection of pairwise disjoint sets of tasks $\{Q_b\}_{b \in \tilde{B}}$, in which for each box $b \in \tilde{B}$, the set Q_b fits into b, $|Q_b| = n_b$, and each task $i \in Q_b$ has a time

window containing the leftmost edge of I; then there is an injective function $f : \tilde{S} \to \bigcup_{b\in\tilde{B}} Q_b$ such that $f(i)$ can be scheduled instead of i, i.e.,$|\tilde{P}(i) \cap \mathrm{tw}(f(i))| \geq p(f(i))$, for each $i \in \tilde{S}$.

We sketch now how we will apply Lemma 4. Since we are unable to guess each set $\overleftarrow{T}'_{w,d}$ at this stage, we rather guess the corresponding set of artificial tasks $\tilde{T}_{w,d}$ according to Lemma 4. Notice that this is doable in polynomial time (per pair (w,d)) since it is sufficient to guess the distinct lengths $p_1, ..., p_{1/\varepsilon}$ and the respective number of tasks $\tilde{n}_1, ..., \tilde{n}_{1/\varepsilon}$. We will pass on the tasks $\tilde{T}_{w,d}$ to our right subproblem as placeholders: intuitively, they will reserve some capacity on the right subproblem which will eventually be occupied by actual tasks (potentially the tasks $\overleftarrow{T}'_{w,d}$) computed in the left subproblem: this replacement exploits Property 5 of the lemma. Also, we guess the boxes $\tilde{B}_{w,d}$ and the number $n_b \in \mathbb{N}$ for each $b \in \tilde{B}_{w,d}$ corresponding to $\tilde{T}_{w,d}$, which also takes polynomial time. These boxes will be passed on to the left subproblem: intuitively, the tasks placed in these boxes in the left subproblem will replace the artificial tasks in $\tilde{T}_{w,d}$.

Let $\tilde{T}$ denote the union of all sets $\tilde{T}_{w,d}$ guessed in this way and let $\tilde{B}$ denote the union of all the sets of boxes $\tilde{B}_{w,d}$ guessed in this way. We will call the tasks in $\tilde{T}$ *artificial tasks.*

We will recurse on a left and a right subproblem, corresponding to I^L and I^R, respectively. For each box $b \in B$, we intuitively guess how many input tasks from the left and the right subproblem are assigned to b in $\overline{\mathsf{OPT}}$. Formally, we guess values $n_b^L, n_b^R \in \mathbb{N}_0$ where n_b^L is the number of tasks $i \in \overline{\mathsf{OPT}}_b$ for which $\mathrm{tw}(i) \cap I^L \neq \emptyset$ and n_b^R is the number of tasks $i \in \overline{\mathsf{OPT}}_b$ for which $\mathrm{tw}(i) \cap I^L = \emptyset$ (in particular, $n_b^L + n_b^R = n_b$). The input for the left subproblem consists of

- the interval I^L, where each edge $e \in I^L$ has capacity $u'(e) - \sum_{b\in\bar{B}:e\in\mathrm{pb}(b)} h(b)$,
- the task set $\{i \in T' : \mathrm{tw}(i) \cap I^L \neq \emptyset\}$,
- the boxes $B \cup \bar{B} \cup \tilde{B}$, the value n_b for each box $b \in \bar{B} \cup \tilde{B}$, and the value n_b^L for each box $b \in B$.

The input for the right subproblem consists of

- the interval I^R, where each edge $e \in I^R$ has capacity $u'(e) - \sum_{b\in\bar{B}:e\in\mathrm{pb}(b)} h(b)$,
- the task set $\{i \in T' : \mathrm{tw}(i) \cap I^L = \emptyset\} \cup \tilde{T}$,
- the boxes B and the value n_b^R for each box $b \in B$.

Suppose that recursively we computed a solution to the left and the right subproblem. We combine them to a solution to the (given) subproblem corresponding to I as follows. Let $Q_{I^L}, \{Q_b\}_{b\in B\cup\bar{B}\cup\tilde{B}}$ denote the computed tasks in the left subproblem, and let $Q'_{I^R}, \{Q'_b\}_{b\in B}$ denote the computed tasks in the right subproblem. For each task i in these computed sets of tasks, the returned solutions include a computed path $P(i)$. We compute the injective function f due to the last property of Lemma 4 by solving a b-matching instance. If such a function f does not exist, we simply discard the considered combination of guesses.

We want to return a solution that consists of the tasks in $Q_{I^L} \cup \big(\bigcup_{b\in\bar{B}} Q_b\big) \cup (Q_{I^R} \setminus \tilde{T})$ and additionally a set of tasks $\tilde{Q}$ that we use to replace the artificial tasks in $\tilde{T}$ in the solution Q_{I^R}. For each task $i \in Q_{I^L} \cup \big(\bigcup_{b\in\bar{B}} Q_b\big) \cup (Q_{I^R} \setminus \tilde{T})$ we keep the path $P(i)$ that we obtained from the solution of the left or right subproblem, respectively. We want to replace the tasks in $Q_{I^R} \cap \tilde{T}$ and, to this end, we compute the set of tasks $\tilde{Q} \coloneqq f(Q_{I^R} \cap \tilde{T}) \subseteq \bigcup_{b\in\bar{B}} Q_b$ and define paths for them as follows. For each task $i \in Q_{I^R} \cap \tilde{T}$ we schedule the task $f(i) \in \tilde{Q}$ within the edges of $P(i)$, i.e., we compute a path $P(f(i))$ for $f(i)$ such that $P(f(i)) \subseteq P(i)$. This is possible since $|P(i) \cap \mathrm{tw}(f(i))| \geq p(f(i))$ due to the last property of Lemma 4. Notice that this replacement does not violate the edge capacities since we omit $i \in \tilde{T}$ from our solution and $d(i) = d(f(i))$.

This yields a candidate solution consisting of the tasks $Q_I \coloneqq Q_{I^L} \cup \big(\bigcup_{b\in\bar{B}} Q_b\big) \cup (Q_{I^R} \setminus \tilde{T}) \cup \tilde{Q}$ and for each box $b \in B$ the tasks in $Q_b \cup Q'_b$ and their respective computed paths $P(i)$. Each combination of our quasi-polynomial number of guesses yields a candidate solution. We reject a candidate if the resulting solution is infeasible; among the remaining candidate solution, we return the solution that maximizes $w(Q_I)$. Recall here that $\tilde{Q}$ are real tasks that we used to replace the artificial tasks $\tilde{T}$; we count the profit of the tasks in $\tilde{Q}$ but not the profit of $\tilde{T}$. Also, we do not consider the profit of the tasks in B.

2.3 Analysis

We argue that our algorithms runs in quasi-polynomial time and that it returns a (feasible) $(2+\varepsilon)$-approximate solution. The bound on the running time holds easily since in each recursive call we make quasi-polynomially many guesses and the recursion depth is $O(\log m)$, which yields the total quasi-polynomial running time bound. Recall that we reject any infeasible candidate solution, so our returned solution is assuredly feasible.

Lemma 5. *Let W be the number of different profits, and let D be the number of different demands of the tasks in our* twUFP *instance. Our algorithm runs in time $O((mn)^{O(WD\log^2 m)/\varepsilon^2})$ and its returned solution is always feasible.*

We now bound our approximation ratio. Assuming now that we guess all boxes b, their corresponding values n_b, n_b^L, n_b^R and the artificial tasks correctly at every stage of the algorithm (i.e., that match Lemma 2, Lemma 4, and $\overline{\mathsf{OPT}}_r$), at each level we lose at most an $O(\varepsilon)$-fraction of the profit when we replace the artificial tasks by real tasks and place tasks in the boxes (see Lemma 4). Since we have $O(\log m)$ levels, this yields a profit of at least $(1 + O(\varepsilon \log m))^{-1} \cdot w(\overline{\mathsf{OPT}}_r) \geq (2 + O(\varepsilon \log m))^{-1} w(\mathsf{OPT})$. We rescale ε by a factor $\log m$ to obtain our approximation ratio of $2 + O(\varepsilon)$.

Theorem 2. *For any $\varepsilon > 0$, there is a $(2+\varepsilon)$-approximation for* twUFP *under $(1 + O(\varepsilon))$-resource augmentation with running time $O\big((mn)^{O(\log^2 n \log^5 m/\varepsilon^4)} \cdot \log U\big)$, where $U \coloneqq \max_{e\in E} u(e)$ is the maximum capacity of the instance.*

We adopt our algorithm to a $(1+\varepsilon)$-approximation for spanUFP under resource augmentation: we pad E with dummy edges on the right such that all feasible solutions become left constrained; thus, we avoid losing the factor of 2.

Theorem 3. *For any $\varepsilon > 0$ there is a QPTAS for* spanUFP *under $(1 + O(\varepsilon))$-resource augmentation.*

Acknowledgments. The authors thank the anonymous reviewers for their helpful comments. Fabrizio Grandoni, Edin Husić and Antoine Tinguely were supported by the Swiss National Science Foundation (SNSF) Grant 200021 200731/1.

References

1. Anagnostopoulos, A., Grandoni, F., Leonardi, S., Wiese, A.: A mazing 2+ε approximation for unsplittable flow on a path. In: SODA. pp. 26–41 (2014)
2. Bansal, N., Chakrabarti, A., Epstein, A., Schieber, B.: A quasi-PTAS for unsplittable flow on line graphs. In: STOC. pp. 721–729 (2006)
3. Bansal, N., Friggstad, Z., Khandekar, R., Salavatipour, R.: A logarithmic approximation for unsplittable flow on line graphs. In: SODA. pp. 702–709 (2009)
4. Bar-Noy, A., Guha, S., Naor, J., Schieber, B.: Approximating the throughput of multiple machines in real-time scheduling. SIAM J. Comput. **31**(2), 331–352 (2001). https://doi.org/10.1137/S0097539799354138, https://doi.org/10.1137/S0097539799354138
5. Bar-Noy, A., Bar-Yehuda, R., Freund, A., Naor, J., Schieber, B.: A unified approach to approximating resource allocation and scheduling. Journal of the ACM (JACM) **48**(5), 1069–1090 (2001)
6. Batra, J., Garg, N., Kumar, A., Mömke, T., Wiese, A.: New approximation schemes for unsplittable flow on a path. In: SODA. pp. 47–58 (2015). https://doi.org/10.1137/1.9781611973730.5, http://epubs.siam.org/doi/abs/10.1137/1.9781611973730.5
7. Berman, P., DasGupta, B.: Multi-phase algorithms for throughput maximization for real-time scheduling. J. Comb. Optim. **4**(3), 307–323 (2000). https://doi.org/10.1023/A:1009822211065, https://doi.org/10.1023/A:1009822211065
8. Bonsma, P., Schulz, J., Wiese, A.: A constant-factor approximation algorithm for unsplittable flow on paths. SIAM Journal on Computing **43**, 767–799 (2014)
9. Chakaravarthy, V.T., Choudhury, A.R., Gupta, S., Roy, S., Sabharwal, Y.: Improved algorithms for resource allocation under varying capacity. In: ESA. pp. 222–234 (2014)
10. Chlebík, M., Chlebíková, J.: Complexity of approximating bounded variants of optimization problems. Theor. Comput. Sci. **354**(3), 320–338 (2006). https://doi.org/10.1016/j.tcs.2005.11.029, https://doi.org/10.1016/j.tcs.2005.11.029
11. Chrobak, M., Woeginger, G., Makino, K., Xu, H.: Caching is hard, even in the fault model. In: ESA. pp. 195–206 (2010)
12. Chuzhoy, J., Ostrovsky, R., Rabani, Y.: Approximation algorithms for the job interval selection problem and related scheduling problems. Math. Oper. Res. **31**(4), 730–738 (2006). https://doi.org/10.1287/MOOR.1060.0218, https://doi.org/10.1287/moor.1060.0218
13. Grandoni, F., Ingala, S., Uniyal, S.: Improved approximation algorithms for unsplittable flow on a path with time windows. In: WAOA. pp. 13–24 (2015)
14. Grandoni, F., Mömke, T., Wiese, A.: Faster (1+ϵ)-approximation for unsplittable flow on a path via resource augmentation and back. In: ESA. vol. 204, pp. 49:1–49:15 (2021)

15. Grandoni, F., Mömke, T., Wiese, A.: A PTAS for unsplittable flow on a path. In: STOC. pp. 289–302. ACM (2022)
16. Grandoni, F., Mömke, T., Wiese, A.: Unsplittable flow on a path: The game! In: SODA. pp. 906–926. SIAM (2022)
17. Grandoni, F., Mömke, T., Wiese, A., Zhou, H.: To augment or not to augment: Solving unsplittable flow on a path by creating slack. In: SODA. pp. 2411–2422 (2017)
18. Grandoni, F., Mömke, T., Wiese, A., Zhou, H.: A $(5/3 + \epsilon)$-approximation for unsplittable flow on a path: placing small tasks into boxes. In: STOC. pp. 607–619 (2018)
19. Im, S., Li, S., Moseley, B.: Breaking 1 - 1/e barrier for nonpreemptive throughput maximization. SIAM J. Discret. Math. **34**(3), 1649–1669 (2020). https://doi.org/10.1137/17M1148438, https://doi.org/10.1137/17M1148438
20. Spieksma, F.C.R.: On the approximability of an interval scheduling problem. Journal of Scheduling **2**, 215–227 (1999)
21. Fernandez de la Vega, W., Lueker, G.S.: Bin packing can be solved within $1+\varepsilon$ in linear time. Combinatorica **1**, 349–355 (1981)
22. Williamson, D.P., Shmoys, D.B.: The design of approximation algorithms. Cambridge university press (2011)
23. Woeginger, G.J.: There is no asymptotic PTAS for two-dimensional vector packing. Inf. Process. Lett. **64**(6), 293–297 (1997). https://doi.org/10.1016/S0020-0190(97)00179-8, https://doi.org/10.1016/S0020-0190(97)00179-8