A CONSTANT FACTOR APPROXIMATION ALGORITHM FOR UNSPLITTABLE FLOW ON PATHS

PAUL BONSMA*, JENS SCHULZ**, AND ANDREAS WIESE**

Abstract. We study the unsplittable flow problem on a path P. We are given a set of n tasks. Each task is specified by a sub path of P, a demand, and a profit. Moreover, each edge of P has a given capacity. The aim is to find a subset of the tasks with maximum profit, for which the given demands can be simultaneously routed along P, subject to the capacities. The best known polynomial time approximation algorithm for this problem achieves a performance ratio of $O(\log n)$ and the best known hardness result is weak NP-hardness. In this paper, we firstly show that the problem is strongly NP-hard, even when the capacities are constant, and all demands are chosen from $\{1, 2, 3\}$. Secondly, we present the first polynomial time constant-factor approximation algorithm for this problem, achieving an approximation factor of $7 + \epsilon$ for any $\epsilon > 0$. This answers an open question from Bansal et al. (SODA'09). We employ a novel framework which reduces the problem to instances where the capacities of the edges differ by at most a constant factor. Moreover, for the difficult "large" tasks – for which in particular the straightforward linear program has an integrality gap of $\Omega(n)$ – we present a new geometrically inspired dynamic program.

Our techniques yields several other results which are of independent interest: for any $\epsilon > 0$ and $\beta > 0$, we give an $(3 + \epsilon)$-approximation algorithm for the case that each task uses at most a $(1 - \beta)$-fraction of the capacities of its edges. Furthermore, we give a $(2 + \epsilon)$-approximation algorithm that violates the capacities by at most a factor $(1 + \epsilon)$ (resource augmentation). Finally, we show that already a running time of $O(n^4 \log n)$ suffices to obtain a constant factor approximation algorithm for the general case.

1. Introduction

In the Unsplittable Flow Problem on a Path (UFPP), we are given a path $P = (V, E)$ with a capacity u_e for each edge $e \in E$. In addition, we are given a set of n tasks T where each task $i \in T$ is characterized by a *start vertex* $s_i \in V$, an *end vertex* $t_i \in V$, a *demand* $d_i \in \mathbb{N}$, and a *profit* $w_i \in \mathbb{N}$. The aim is to compute a set of tasks $F \subseteq T$ with maximum total profit $\sum_{i \in F} w_i$ such that for each edge, the demand sum of the tasks in F using this edge does not exceed its capacity.

Date: February 17, 2011.

*Humboldt Universität zu Berlin, Computer Science Department, Unter den Linden 6, 10099 Berlin. bonsma@informatik.hu-berlin.de.

**Technische Universität Berlin, Institute of Mathematics, Straße des 17. Juni 136, 10623 Berlin. {jschulz,wiese}@math.tu-berlin.de.

The first author was supported by DFG grant BO 3391/1-1. The second author was supported by the DFG Research Center Matheon *Mathematics for key technologies* in Berlin. The third authors was supported by DFG project "Algorithm Engineering for Real-time Scheduling and Routing" within DFG focus program 1307.

The name of this problem is motivated by an interpretation as a multicommodity flow problem, where each task corresponds to a commodity. The term "unsplittable" means that the total amount of flow d_i from each commodity i has to flow completely along the path from the source s_i to the sink t_i or not at all. There are several settings and applications in which this problem occurs, and several other interpretations of the problem. Therefore, this problem, and close variants thereof, have also been studied under the names *bandwidth allocation*, *resource contrained scheduling*, *call admission control*, *temporal knapsack*, *maximum demand flow*, and *resource allocation*. In many applications, the vertices correspond to time points, and tasks have fixed start and end times. Within this time interval they consume a given amount of a common resource, of which the available amount varies over time. For instance, this occurs when scheduling tasks in a uniform machine environment with possible down-times for each machine. Jobs have fixed start and end times, may migrate to different machines, and require a certain number d_i of machines when selected.

This problem is known to be weakly NP-hard, since it contains the KNAPSACK problem as a special case (the case where there is just a single edge). In addition, Darmann et al. [13] recently showed that the special case where all profits and all capacities are uniform remains weakly NP-hard. This implies that the problem admits no Fully Polynomial Time Approximation Scheme (FPTAS) unless $P = NP$, but does not exclude pseudopolynomial time algorithms. While the special case of a single edge admits an FPTAS, no constant-factor approximation algorithm is known for UFPP. The best known polynomial time algorithm achieves an approximation factor of $O(\log n)$ [4]. In addition, there is a $(1 + \epsilon)$-approximation algorithm known with quasi-polynomial running time which additionally requires that the capacities and the demands are quasi-polynomial, i.e. bounded by $2^{\text{polylog}\, n}$ [3]. One special case which has been intensively studied is given by the *no-bottleneck-assumption* [8, 9, 12, 11], where it is required that $\max_i d_i \leq \min_e u_e$ holds. For this case there is a $(2+\epsilon)$-approximation algorithm known [11]. However, it turned out that there are several obstacles that prevent these algorithms to be generalized to the general case, see e.g. [9] and the discussion below.

In conclusion, even though the UFPP has been intensively studied, in terms of polynomial time approximation algorithms for the general case, the best known algorithm is a $O(\log n)$-approximation algorithm [4], and the best negative result is that there cannot be an FPTAS (unless $P = NP$) [13]. It is not known whether the problem can be solved optimally with a pseudopolynomial time algorithm.

1.1. Our Contribution.

In this paper we significantly diminish the gap between positive and negative results for UFPP: First, we present the first polynomial time constant-factor approximation for the general case. Secondly, we prove that the problem is strongly NP-hard even for the restricted case where all demands are chosen from $\{1, 2, 3\}$ and capacities are uniform. Note that in this case even the no-bottleneck-assumption holds. The result implies that unless $P = NP$, the problem admits no pseudopolynomial time algorithm, and it yields a different proof that there can be no FPTAS (see e.g. [28]).

Our main algorithmic result is a $(7 + \epsilon)$-approximation algorithm for the UFPP, for every $\epsilon > 0$. For practical purposes, we also provide a constant factor approximation algorithm with a reasonable running time of only $O(n^4 \log n)$. Along the way, our techniques yield several algorithmic results which are interesting in their own right: For a task i, denote by $b(i)$ the minimum capacity among all edges used by task i, and call the ratio $d_i/b(i)$ its *relative demand.* For every $\beta > 0$ and $\epsilon > 0$, if for every task its relative demand is at most $1 - \beta$, then we obtain a $(3 + \epsilon)$-approximation algorithm. For the setting of resource augmentation, we show how to compute a $(2 + \epsilon)$-approximative solution that is feasible when the capacities are incremented by only a modest factor of $1 + \epsilon$. Finally, we present a polynomial time algorithm for finding maximum weight independent sets in certain types of rectangle intersection graphs.

We now go into more detail about the new techniques we introduce, and give an outline of the paper. Similar to many previous papers, for our main algorithm we partition the tasks into groups, depending on their relative demand. We use three groups, called the *small*, *medium* and *large* tasks. In Section 2 we define this precisely. In Section 3 we introduce a novel framework to handle the small and medium tasks. Here these tasks are first partitioned into smaller sets, which can be solved via dynamic programming, LP-rounding and network flow techniques. (This is similar to many previous papers, such as [8].) Solutions to these smaller sets leave a small amount of the capacity of each edge unused. This allows to recombine these sets, yielding a $(3 + \epsilon)$-approximation for all small and medium tasks (i.e. tasks with relative demand at most $1 - \beta$). If we do not leave a small amount of the capacity unused, we obtain a $(2 + \epsilon)$-approximative solution for *all* tasks this way (i.e., small, medium, and large), at the cost of violating the capacities by at most a factor of $1 + \epsilon$.

The remaining large tasks (those with relative demand larger than $(1 - \beta)$) are treated in Section 4. For those, we present a polynomial time 4-approximation algorithm. We interpret UFPP instances geometrically, by drawing a curve in the plane determined by the capacities, and representing tasks by axis-parallel rectangles, that are drawn as high as possible under this curve. The demand of a task determines the height of the rectangle. Using a novel geometrically inspired dynamic program, we show that in polynomial time, a maximum weight set of pairwise non-intersecting rectangles can be found. Such a set corresponds to a feasible UFPP solution. In addition, we show that when for every task the relative demand is at least $1/k$, a maximum weight set of pairwise non-intersecting rectangles yields a $2k$-approximative solution for UFPP .

Section 5 finally combines the results of Sections 3 and 4 to our $(7+\epsilon)$-approximation algorithm. In addition, we obtain an algorithm with only a running time of $O(n^4 \log n)$ which still achieves a (constant) approximation ratio of 25.12. In Section 6, we present our strong NP-hardness proof. In Section 7 we conclude with a discussion.

1.2. **Background and Related Work.** UFPP has been studied from different perspectives, and many special cases under various assumptions were considered. In the following, we will give an overview of the main results. As mentioned above, the UFPP is weakly NP-hard due to the contained KNAPSACK problem. In fact, UFPP is a special case of MULTI-DIMENSIONAL KNAPSACK, which is obtained by allowing the tasks to have different demands for every edge. Therefore, if the number of edges m is constant, the problem admits a PTAS, see [17]. If all edge capacities are bounded by a constant C, then a straightforward dynamic programming algorithm solves the problem in polynomial time $n^{O(C)}$ (recall that all demands are integers).

The special case of UFPP in which all capacities are equal has received a lot of study, and is often called the Resource Allocation Problem (RAP). If all demands are 1, the RAP can be solved in time $O(n^2 \log n)$ by minimum-cost flow computations since the coefficient matrix corresponds to a network flow matrix, as shown by Arkin and Silverberg [2]. RAP admits a straightforward 0/1 integer linear programming formulation. Calinescu et al. [8] mention how its LP relaxation can be solved similarly in time $O(n^2 \log^2 n)$, using a minimum cost flow algorithm [25] (even with arbitrary demands).

We mentioned before that for the special case of UFPP where the no-bottleneck assumption holds (i.e. $\max_i d_i \leq \min_e u_e$), a $(2+\epsilon)$-approximation algorithm has been given by Chekuri et al. [11]. Note that the no-bottleneck assumption holds in particular for RAP. Before this, a $(2+\epsilon)$-approximation algorithm for RAP was given by Calinescu et al. [8]. Although Darmann et al. [13] rule out the existance of an FPTAS for RAP (unless $P = NP$), it remains open whether there exists a PTAS.

Phillips et al [27] obtain a 6-approximation algorithm for RAP by using LP-rounding techniques. They consider a more general version of the problem in which the start and end times are not fixed, so the tasks are allowed to slide in their time window $[s_i, t_i)$. A similar generalization, where for each task one out of a set of alternatives needs to be selected, Bar-Noy et al. [5] provide a constant factor approximation algorithm using the local ratio technique.

The Unsplittable Flow Problem (UFP) has also been considered for other graph classes than just paths. In general graphs, for every task selected in a solution one additionally needs to select a path from s_i to t_i. Many results for UFP again use the no-bottleneck assumption. Chakrabarti et al [9] give the first constant factor approximation for UFP on a path or cycle under this assumption, with a factor of 78.51. For the cycle, they reduce the problem to two UFP problems on a path, by splitting the cycle at a special edge. Chekuri et al. [11] improve this result for UFP to $(2+\epsilon)$. In addition, they give constant factor approximation algorithms for UFP on trees. UFP on general graphs has been studied by Chakrabarti et al. [9], who give non-constant factor approximation in this setting.

Recall that for our main algorithm, we develop a polynomial time algorithm to find a maximum weight set of pairwise non-intersecting rectangles, for sets of rectangles drawn as high as possible under a certain curve. This is closely related to the Maximum Independent Set of Rectangles (MISR) problem, studied by Chalermsook and Chuzhoy [10]. In this problem, a collection of n axis-parallel rectangles is given and the task is to find a maximum-cardinality subset of disjoint rectangles. They provide a randomized $O(\log \log n)$-approximation. For the weighted case, there are several $O(\log n)$-approximation algorithms known [1, 20, 24]. For geometric intersection graphs of disks, squares and similar objects, Erlebach, Jansen and Seidel [16] give polynomial time approximation schemes for the Maximum Weight Independent Set and the Minimum Weight Vertex Cover problem. Unfortunately, their approach does not carry over to rectangles if the ratio between their height and width can be arbitrary.

The MISR problem is related to adjacent resource scheduling problems. In such problems, one wants to schedule a job on several machines in parallel which must be *contiguous*, i.e., adjacent to each other. Duin and van Sluis [14] prove the decision variant of scheduling tasks on contiguous machines to be strongly NP-complete. RAP on contiguous machines has been considered under the name *storage allocation problem* (SAP), in which tasks are axis-aligned rectangles that are only allowed to move vertically. Leonardi et al. [22] provide a 12-approximation algorithm for SAP. Bar-Yehuda et al. [6] present a deterministic polynomial-time $(2 + \epsilon + 1/(e - 1))$-approximation algorithm based on the local ratio technique. This result also holds for RAP.

In this paper, we also study the UFPP problem in the setting of *resource augmentation*. This means that we find a solution which is feasible if we increase the capacity of each edge by a modest factor of $(1 + \epsilon)$. The paradigm of resource augmentation is very popular in real-time scheduling. There, the augmented resource is the speed of the machines. For instance, it is known that the natural earliest deadline first policy (EDF) is guaranteed to work on m machines with speed $2 - 1/m$ if the instance can be feasibly scheduled on m machines with unit speed [26]. Also, a matching feasibility test is known [7]. For further examples of resource augmentation results in real-time scheduling see [15, 21].

1.3. **Applications.** We want to add a further interesting application where UFPP occurs. In constraint programming throughout branch-and-bound search, infeasible subproblems occur that are analyzed in order to create new globally valid constraints or to use non-chronological backtracking. To make these constraints as reusable as possible one searches for minimum size explanations that are responsible for the infeasibility. In case of the cumulative constraint that models scheduling problems, see e.g. [19], one needs to report a minimum set of jobs such that an interval $[a, b)$ is overloaded at each point in time. Each job has a demand $d_j \in \mathbb{N}$ and a core $[s_j, t_j)$ when it must be scheduled under a capacity $C \in \mathbb{N}$. Using binary variables x_j that model whether a job is picked or not, we need to solve the following integer program $\min\{\sum_j x_j \mid \sum_j d_j \cdot x_j \geq C \; \forall t\}$. Let U_t denote the sum of all demands at time t, i.e., $U_t = \sum_{j : t \in [s_j, t_j)} d_j$. Then, the integer program can be stated as $\max\{\sum_j x_j \mid \sum_j d_j \cdot x_j \leq U_t - C \; \forall t\}$, which is exactly the UFPP formulation with uniform profits. Arbitrary profits are introduced when the lower and upper bounds of the variables shall be taken into account.

1.4. **Discussion of the No-Bottleneck Assumption.** The *no-bottleneck assumption* requires that no demand is larger than the minimum capacity, i. e. $\max_i d_i \leq \min_e u_e$ holds. This is a strict assumption since it removes certain complications which are not easy to handle. For example, it was shown by Chakrabarti et al. [9] that if the no-bottleneck assumption holds, the natural LP-relaxation of the problem has a constant integrality gap. However, without this assumption the integrality gap can be as large as $\Omega(n)$. Consider the following example adapted from [4]: the capacites of each edge $e = \{k, k+1\}, k = 0, \ldots, m-1$, are given by $u_e = 2^{m-k}$ and for each $i = 1, \ldots, m$ there is a task i with $s_i = 0$, $t_i = i$, $d_i = 2^{m-i+1}$, and unit profit. Observe that this instance does *not* obey the no-bottleneck assumption. Any integral solution can choose at most one task, but the natural LP relaxation achieves a profit of $n/2$. Moreover, the no-bottleneck assumption implies that if all tasks have relative demand at least δ, in any solution there can be at most $2\left\lfloor 1/\delta^2 \right\rfloor$ tasks i which use a given edge e, see [9]. This property is useful for setting up a dynamic program. However, a simple modification of the above instance shows that without the no-bottleneck assumption this no longer holds; when doubling all capacities, all tasks have relative demand $1/2$, yet an optimal solution contains all tasks. These are the main obstacles that need to be overcome in order to give a constant factor approximation algorithm for the general case of UFPP.

2. Preliminaries

In this section, we introduce some notation and terminology, and define precisely how we partition the tasks into small, medium, and large tasks.

We assume that the vertices of the path $P = (V, E)$ are numbered $V = \{0, \dots, m\}$, and $E = \{\{i, i+1\} \mid 0 \leq i \leq m-1\}$. We assume that the tasks are numbered $T = \{1, \dots, n\}$. Recall that tasks are characterized by two vertices s_i and t_i with $s_i < t_i$, and positive integer demand d_i and profit w_i. For each task $i \in T$ we denote by $P_i \subseteq E$ the edge set of the subpath of P from s_i to t_i. If $e \in P_i$, then task i is said to *use* e. For each edge e we denote by $T_e \subseteq T$ the set of tasks which use e. A set of tasks $F \subseteq T$ is said to obey the capacities of the edges if $\sum_{i \in F \cap T_e} d_i \leq u_e$ for each edge e. For a set of tasks F we define its profit by $w(F) := \sum_{i \in F} w_i$. Hence, formally our objective is to find a set of tasks F with maximum profit which obeys the capacities of the edges.

For each task i we define its *bottleneck capacity* $b(i)$ by $b(i) := \min_{e \in P_i} u_e$. An edge e is called a *bottleneck edge* for the task i if $e \in P_i$ and $u_e = b(i)$. In addition, we define for every task i that $\ell(i) := b(i) - d_i$. The value $\ell(i)$ can be interpreted as the remaining capacity of a bottleneck edge of i when i is selected in a solution.

Consider a vertex $v \in V$ and an edge $e \in E$ with $e = \{x, x+1\}$. We write $v < e$ (or $v > e$) if $v \leq x$ (resp. $v \geq x+1$). For two edges $e = \{x, x+1\}$ and $e' = \{x', x'+1\}$ in E, we write $e < e'$ if $x < x'$ and $e \leq e'$ if $x \leq x'$. In other words, we interpret an edge $\{x, x+1\}$ simply as a number between x and $x+1$. If $e < e'$ then we will also say that e lies *before* e', and e' lies *after* e. If $e = \{x-1, x\}$ and $e' = \{x, x+1\}$ then e lies *immediately before* e'.

Without loss of generality, we will assume throughout this paper that $u_e \geq 1$ for all edges e and $d_i \geq 1$ for all tasks i; zero demands and capacities can easily be handled in a preprocessing step. Moreover, observe that one can easily adjust any given instance to an equivalent instance in which each vertex is either a start or an end vertex of a task. Such an adjustment can be implemented in linear time and it hence does not dominate the running times of the algorithms presented in this paper. Therefore, we will henceforth assume that $m < 2n$.

We define an α-approximation algorithm for a maximization problem to be an algorithm which computes a feasible solution for a given instance such that its objective value is at least $\frac{1}{\alpha}$ times the optimal value. Unless otherwise stated, we always assume that an approximation algorithm has polynomial running time. Throughout, for a subset of the tasks $F \subseteq T$, $OPT(F)$ denotes an optimal solution for the UFPP instance restricted to the task set F.

Throughout this paper, we will use the notations defined above to refer to the UFPP instance currently under consideration; we will never consider multiple instances simultaneously, so there is no cause for ambiguity.

2.1. Task classification.

For our algorithms, we partition the tasks into small, medium and large tasks. Whether a task is small, medium, or large depends on its *relative demand* $d_i/b(i)$, which is a number in $(0, 1]$. Such a grouping of the tasks has been done by several authors before, see e.g. [3, 4, 6, 8, 9, 11].

Let $\epsilon > 0$ and let $\beta > 0$ with $\beta \leq 1/4$ and $\beta \leq \epsilon$. For technical reasons we require that $\epsilon < 1$. Choosing ϵ and β small will give a better approximation ratio but worse running time. In the sequel, we will need constants ℓ and q which depend on ϵ and β. We define them already here to clarify matters. We choose

q such that $2^{1-q} \leq \beta$ and ℓ such that $\frac{\ell+q}{\ell} \leq 1+\epsilon$. We define the constant δ such that $\delta' := \delta/(1-\epsilon) < \frac{3-\sqrt{5}}{2}$ and $\frac{1+\sqrt{\delta'}}{1-\sqrt{\delta'}-\delta'} \leq 1+\epsilon$ (note that this is always possible). The reason is that later we will invoke an algorithm by Chekuri et al. [11] as subroutine. That algorithm assumes that the no-bottleneck-assumption holds, i.e., $\max_i d_i \leq \min_e u_e$. It computes a $\left(\frac{1+\sqrt{\delta'}}{1-\sqrt{\delta'}-\delta'}\right)$-approximation if for each task i it holds that $d_i \leq \delta' \cdot b(i)$ and $\delta' < \frac{3-\sqrt{5}}{2}$. For technical reasons we additionally require that $\delta \leq (1-\beta)/2^{\ell}$. We define a task i to be

- *small*, if $d_i \leq \delta \cdot b(i)$,
- *medium*, if $\delta \cdot b(i) < d_i \leq (1-2\beta)b(i)$, and
- *large,* if $(1-2\beta)b(i) < d_i$.

We denote by F_{sml}, F_{med}, and F_{lrg} the set of small, medium, and large tasks of a given instance. In Section 3 we present approximation algorithms for the small and medium tasks. After that in Section 4 we present an approximation algorithm for the large tasks. In Section 5 we show how these results can be combined to yield our two main approximation algorithms.

3. Small and Medium Tasks

In this section we present constant-factor approximation algorithms for small and medium tasks. To this end, we give algorithms which compute sets of tasks $ALG\left(F^{k,\ell}\right) \subseteq F^{k,\ell}$ (for sets $F^{k,\ell}$ defined below) such that

- $w(ALG\left(F^{k,\ell}\right)) \geq \frac{1}{\alpha} \cdot w(OPT\left(F^{k,\ell}\right))$ (for some constant α) and
- in each edge the set $ALG\left(F^{k,\ell}\right)$ leaves a fraction of the edge-capacity unused.

We feed these algorithms into a framework which then yields a constant factor approximation algorithm for the set of *all* tasks which are small or medium.

For the small tasks, we obtain a $(1+O(\epsilon))$-approximation algorithm, for the medium tasks we get a $2+O(\epsilon)$-approximation algorithm. Putting these two algorithms together, this yields a $(3+O(\epsilon))$-approximation algorithm for the set of all tasks which are small or medium. Finally, we show that if we are allowed to increase the capacity of each edge by a factor of $1+\beta$ (resource augmentation) our framework even yields a $(2+O(\epsilon))$-approximation algorithm for *all* tasks (small, medium and large).

3.1. Framework.

We define the framework mentioned above. Assume we are given an instance of the UFPP problem. For our framework, we partition the tasks according to their bottleneck capacities. We define $F^{k,\ell} := \left\{i \in T | 2^k \leq b(i) < 2^{k+\ell}\right\}$ for each value k such that the resulting set is non-empty. Note that this includes negative values for k. Likewise, we define $F^{k,\ell}_{\text{sml}} := F_{\text{sml}} \cap F^{k,\ell}$ and $F^{k,\ell}_{\text{med}} := F_{\text{med}} \cap F^{k,\ell}$. Later we will present algorithms which compute task sets $ALG\left(F^{k,\ell}_{\text{med}}\right) \subseteq F^{k,\ell}_{\text{med}}$ and $ALG\left(F^{k,\ell}_{\text{sml}}\right) \subseteq F^{k,\ell}_{\text{sml}}$. These sets will have the property that they leave an amount of $\beta \cdot 2^k$ of the capacity of each edge unused. Also, their profit is only by a constant α smaller than that of $OPT\left(F^{k,\ell}_{\text{med}}\right)$ and $OPT\left(F^{k,\ell}_{\text{sml}}\right)$, respectively. We make this precise in the following definition.

Definition 1 ((α,β)-approximative)**.** Consider a set $F^{k,\ell}$. A set $F \subseteq F^{k,\ell}$ is called (α,β)*-approximative* if

- $w(F) \geq \frac{1}{\alpha} \cdot w(OPT(F^{k,\ell}))$ and,
- $\sum_{i \in F' \cap T_e} d_i \leq u_e - \beta \cdot 2^k$ for each edge e such that $T_e \cap F^{k,\ell} \neq \emptyset$.

An algorithm which computes (α, β)-*approximative* sets in polynomial time is called an (α, β)-*approximation algorithm.* We call the second condition the *modified capacity constraint.*

Our framework consists of a procedure which turns an (α, β)-approximation algorithm for each set $F^{k,\ell}$ into a γ-approximation algorithm for all tasks, where γ is a function of α, β and ℓ.

Lemma 2 (Framework)**.** *Let $\beta > 0$ and let $\ell, q \in \mathbb{N}$ such that $2^{1-q} \leq \beta$. Let the sets $F^{k,\ell}$ be defined as stated above. Assume we are given an (α, β)-approximation algorithm for each set $F^{k,\ell}$ with running time $O(p(n))$ for a polynomial p. Then there is a $\left(\frac{\ell+q}{\ell} \cdot \alpha\right)$-approximation algorithm with running time $O(m \cdot p(n))$ for the set of* all *tasks.*

Now we describe the algorithm which yields Lemma 2. Assume that the given (α, β)-approximation algorithm computes sets $ALG\left(F^{k,\ell}\right) \subseteq F^{k,\ell}$. The key idea is that due to the unused edge-capacities of the sets $ALG\left(F^{k,\ell}\right)$, the union of several of these sets still yields a feasible solution. With an averaging argument and a geometric bound we will show further that the indices k for the sets $ALG\left(F^{k,\ell}\right)$ that we want to combine can even be chosen such that the resulting set is again a constant factor approximation (and feasible).

Formally, for each offset $c \in \{0, ..., \ell+q-1\}$ we define $\eta(c) = \{c + i \cdot (\ell+q) \mid i \in \mathbb{Z}\}$. For each $c \in \{0, ..., \ell+q-1\}$ we compute the set $ALG(c) = \bigcup_{k \in \eta(c)} ALG(F^{k,\ell})$. In Lemma 4 we will prove that each set $ALG(c)$ is feasible. We output the set $ALG(c^*)$ with maximum profit among all sets $ALG(c)$. In Lemma 5 we will prove that the resulting set is a $(\frac{\ell+q}{\ell} \cdot \alpha)$-approximation.

First, we prove a lemma that we will employ to bound the capacity used by the tasks from each set $ALG(F^{k,\ell})$ on each edge. Observe that the lemma is proven for arbitrary feasible solutions of UFPP.

Lemma 3. *Let F be a feasible UFPP solution such that for all $i \in F$, $b(i) < c$, and for each edge e, $\sum_{i \in F \cap T_e} d_i \leq \max\{u_e - c', 0\}$ (with $c' < c$). Then for each edge e, $\sum_{i \in F \cap T_e} d_i < 2(c - c')$.*

Proof. Let e be an edge. If $u_e \leq c$, then the claim follows immediately. Now suppose that $u_e > c$. The idea is that any task $i \in F \cap T_e$ must use an edge whose capacity is less than c. In particular, it must use either the closest edge before e or the closest edge after e whose capacity is less than c. Since F leaves an amount of c' of the capacity of each edge unused, it follows that the total capacity used by tasks in F must be less than $2(c - c')$.

Formally, we define x to be the maximum vertex such that $x < e$ and $u_{\{x,x+1\}} < c$, if this exists. In that case, let $F_L \subseteq F \cap T_e$ consist of all tasks that use $\{x, x+1\}$. If such an edge does not exist, then let $F_L = \emptyset$. Either way, we have that $\sum_{i \in F_L} d_i < c - c'$. Similarly, we define y to be the minimum vertex such that $y > e$ and $u_{\{y,y+1\}} < c$, if this exists. Then, let $F_R \subseteq F \cap T_e$ consist of all tasks that use $\{y, y+1\}$. Otherwise, let $F_R = \emptyset$. Since for every task $i \in F$ it holds that $b(i) < c$, it follows that $F = F_L \cup F_R$. This implies that

$$\sum_{i\in F\cap T_e} d_i \leq \sum_{i\in F_L} d_i + \sum_{i\in F_R} d_i < 2(c - c').$$

□

Now we prove that each set $ALG(c)$ is feasible, since we chose $2^{1-q} \leq \beta$.

Lemma 4. *For each* $c \in \{0, ..., \ell + q - 1\}$ *the set* $ALG(c)$ *is feasible.*

Proof. Let $c \in \{0, ..., \ell + q - 1\}$ and let e be an edge. Denote $t = \ell + q$. Let k be the largest integer in $\eta(c)$ such that $2^k \leq u_e$. For every $x \in \eta(c)$, denote

$$U_e^x = \sum_{j\in T_e \cap ALG(F^{x,\ell})} d_j.$$

Since $ALG(F^{k,\ell})$ is an (α, β)-approximation and $2^{1-q} \leq \beta$,

$$U_e^k \leq u_e - \beta \cdot 2^k \leq u_e - 2^{k+1-q}.$$

For every $i \geq 1$, Lemma 3 shows that

$$U_e^{k-it} \leq 2(2^{k-it+\ell} - \beta \cdot 2^{k-it}) \leq 2^{k+1-it+\ell} - 2^{k+2-it-q}.$$

Summarizing, we have that

$$\begin{aligned}\sum_{j\in T_e \cap ALG(c)} d_j &= \sum_{i=0}^{\infty} U_e^{k-it} \\ &\leq u_e - 2^{k+1-q} + \sum_{i=1}^{\infty}(2^{k+1-it+\ell} - 2^{k+2-it-q}) \\ &< u_e + \sum_{i=1}^{\infty} 2^{k+1-(i-1)t-q} - \sum_{i=0}^{\infty} 2^{k+1-it-q} = u_e.\end{aligned}$$

□

Now, we use an averaging argument to prove the approximation factor of the set $ALG(c^*)$.

Lemma 5. *We have that* $w(ALG(c^*)) \geq \frac{\ell}{\ell+q} \cdot \frac{1}{\alpha} \cdot w(F^*)$, *where* F^* *denotes an optimal solution of the given instance.*

Proof. We calculate that

$$\begin{aligned}\sum_{c=0}^{\ell+q-1} w(ALG(c)) &\geq \sum_{c=0}^{\ell+q-1} \sum_{k\in\eta(c)} \frac{1}{\alpha} \cdot w\left(OPT\left(F^{k,\ell}\right)\right) \\ &= \frac{1}{\alpha} \cdot \sum_{k\in\mathbb{Z}} w\left(OPT\left(F^{k,\ell}\right)\right) \\ &\geq \frac{1}{\alpha} \cdot \sum_{k\in\mathbb{Z}} w\left(F^* \cap F^{k,\ell}\right) \\ &= \frac{1}{\alpha} \cdot \sum_{k\in\mathbb{Z}} \sum_{j=k}^{k+\ell-1} w\left(F^* \cap F^{j,1}\right) \\ &= \frac{\ell}{\alpha} \cdot w(F^*).\end{aligned}$$

Hence, there must be one value c such that $w(ALG(c)) \geq \frac{\ell}{\ell+q} \cdot \frac{1}{\alpha} \cdot w(F^*)$. Since we defined c^* such that $w(ALG(c^*))$ is maximized, the claim follows. □

Finally, we can prove Lemma 2 which completes our framework.

Proof of Lemma 2. Lemma 4 implies that $ALG(c^*)$ is feasible and Lemma 5 implies that $ALG(c^*)$ is a $\left(\frac{\ell+q}{\ell} \cdot \alpha\right)$-approximation. For computing $ALG(c^*)$ we need to compute the set $ALG\left(F^{k,\ell}\right)$ for each relevant value k. There are at most $m\ell \in O(m)$ relevant values k. Finding the optimal offset c^* can be done in $O(m)$ steps. This yields an overall running time of $O\left(m \cdot p(n)\right)$. □

We note that Bansal et al. [4] used a geometric partitioning of the tasks by demands (rather than by bottleneck capacites like here).

3.2. **Approximation for Small Tasks.** We present an algorithm which computes an $(1+O(\epsilon), \beta)$-approximative solution $ALG\left(F^{k,\ell}_{\text{sml}}\right)$ for each set $F^{k,\ell}_{\text{sml}}$. Later, we will feed the computed sets into the framework of Lemma 2 to get a $(1+O(\epsilon))$-approximation for the small tasks. Recall that our constants ϵ, β, δ, ℓ, and q are chosen according to Section 2.1.

Consider a set $F^{k,\ell}_{\text{sml}}$. The idea is to consider two IP formulations of the problem. The first one is the natural formulation $IP^{k,\ell}$ to solve UFPP over the set $F^{k,\ell}_{\text{sml}}$. The second formulation $IP^{k,\ell}_{\beta}$ tightens the capacity constraint and leaves $\beta \cdot 2^k$ units of each edge capacity unused, which affects the gained profit only by a factor of $(1-\beta)^{-1} \leq (1-\epsilon)^{-1}$ since $\beta \leq \epsilon$. We will show in Lemma 7 that solving $IP^{k,\ell}_{\beta}$ yields a certain approximation to the original problem. To solve $IP^{k,\ell}_{\beta}$ we use the algorithm by Chekuri et al. [11], which we will analyze for our purposes in Lemma 6.

The integer programming formulation for $IP^{k,\ell}$ is defined as follows:

$$\begin{aligned} IP^{k,\ell}: \qquad & \max \sum_{i \in F^{k,\ell}_{\text{sml}}} w_i \cdot x_i \\ & \text{s.t.} \sum_{i \in T_e \cap F^{k,\ell}_{\text{sml}}} x_i \cdot d_i \leq u_e && \forall e \in E \\ & \qquad\qquad x_i \in \{0,1\} && \forall i \in F^{k,\ell}_{\text{sml}} \end{aligned}$$

Next, we state $IP^{k,\ell}_{\beta}$, where each capacity constraint leaves a fraction β unused.

$$\begin{aligned} IP^{k,\ell}_{\beta}: \qquad & \max \sum_{i \in F^{k,\ell}_{\text{sml}}} w_i \cdot x_i \\ & \text{s.t.} \sum_{i \in T_e \cap F^{k,\ell}_{\text{sml}}} x_i \cdot d_i \leq u_e - \beta \cdot 2^k && \forall e \in E \\ & \qquad\qquad x_i \in \{0,1\} && \forall i \in F^{k,\ell}_{\text{sml}} \end{aligned}$$

We denote by $LP^{k,\ell}$ and $LP^{k,\ell}_{\beta}$ the natural LP relaxations of $IP^{k,\ell}$ and $IP^{k,\ell}_{\beta}$, where the constraint $x_i \in \{0,1\}$ is replaced by $0 \leq x_i \leq 1$. For a given δ we define $f(\delta) := \frac{1+\sqrt{\delta}}{1-\sqrt{\delta}-\delta}$.

Lemma 6. *For a set of tasks with relative demand at most δ, the algorithm of Chekuri et al.* [11] *computes a feasible solution to $IP^{k,\ell}$ with value at least $f(\delta)^{-1} \cdot w(OPT(LP^{k,\ell}))$, and can be implemented to run in time $O(n^3 \log n)$.*

Proof. The algorithm of Chekuri et al. assumes that all tasks have a relative demand of at most δ and that the no-bottleneck assumption holds ($\max_i d_i \le \min_e u_e$). Recall that we assumed that $\delta \le (1-\beta)/2^\ell$ and for all tasks $i \in F^{k,\ell}$ it holds that $d_i \le \delta \cdot b(i)$. Then, $d_i \le \delta \cdot 2^{k+\ell} \le 2^k - \beta \cdot 2^k \le u_e - \beta \cdot 2^k \le u_e$ holds for each task i and each edge e, and hence the no-bottleneck assumption holds.

The algorithm of Chekuri et al. works as follows. The tasks are partitioned into at most n groups, depending on their demands. The demands and the capacity are scaled such that a problem with uniform demands and uniform capacitiesSince the demands and capacities are uniform, this can be solved optimally in time $O(n^2 \log n)$, using the algorithm by Arkin and Silverberg [2]. Then the authors show that combining the solutions of each group yields a feasible solution to $IP^{k,\ell}$. Furthermore, Chekuri et al. have shown in [11, Corollary 3.4] that the obtained solution is at most a factor $f(\delta)$ worse than the optimal LP solution. □

We envoke the algorithm by Chekuri et al. on $IP^{k,\ell}_\beta$ and obtain the solution $ALG(F^{k,\ell}_{\text{sml}})$. In the following lemma, we bound its approximation factor.

Lemma 7. *The solution $ALG\left(F^{k,\ell}_{\text{sml}}\right)$ is $\left(\frac{1+\sqrt{\delta'}}{1-\sqrt{\delta'}-\delta'} \cdot \frac{1}{1-\beta}, \beta\right)$-approximative with $\delta' \le \frac{\delta}{1-\beta}$.*

Proof. First, observe that if every task i has a relative demand of at most δ in $LP^{k,\ell}$, then in $LP^{k,\ell}_\beta$ each task has a relative demand of at most δ' with $\delta' = \frac{\delta}{1-\beta}$ since

$$d_i \le \delta \cdot b(i) = \frac{\delta}{(1-\beta)} \cdot (1-\beta) \cdot b(i) \le \frac{\delta}{(1-\beta)} \cdot (b(i) - \beta \cdot 2^k).$$

According to the proof of Lemma 6 the no-bottleneck assumption also holds for $IP^{k,\ell}_\beta$. Hence, the algorithm returns a $f(\delta') = \frac{1+\sqrt{\delta'}}{1-\sqrt{\delta'}-\delta'}$-approximative solution $ALG\left(F^{k,\ell}_{\text{sml}}\right)$ to $IP^{k,\ell}_\beta$. Observe that $OPT\left(LP^{k,\ell}_\beta\right) \ge (1-\beta) \cdot OPT\left(LP^{k,\ell}\right)$ holds, since we can scale the capacity constraints from $LP^{k,\ell}$ with a factor of $(1-\beta)$ and conclude that every feasible solution of $LP^{k,\ell}$ scaled down by a factor of $(1-\beta)$ corresponds to a feasible solution of $LP^{k,\ell}_\beta$. This gives that

$$\begin{aligned} w(ALG\left(F^{k,\ell}_{\text{sml}}\right)) &\overset{\text{Lemma 6}}{\ge} f(\delta')^{-1} \cdot w(OPT\left(LP^{k,\ell}_\beta\right)) \\ &\ge f(\delta')^{-1} \cdot (1-\beta) \cdot w(OPT\left(LP^{k,\ell}\right)) \\ &\ge \left(f(\delta') \cdot \frac{1}{1-\beta}\right)^{-1} \cdot w(OPT\left(F^{k,\ell}_{\text{sml}}\right)). \end{aligned}$$

□

We summarize this section in the following lemma.

Lemma 8. *Let $\epsilon, \beta, \delta, \ell$, and q be as defined in Section 2.1. Then there is an algorithm with running time $O(n^3 \log n)$ which computes a $(1+O(\epsilon), \beta)$-approximative set $ALG\left(F^{k,\ell}_{\text{sml}}\right)$ for each set $F^{k,\ell}_{\text{sml}}$.*

Proof. Recall that we chose δ such that $\frac{1+\sqrt{\delta'}}{1-\sqrt{\delta'}-\delta'} \leq 1+\epsilon$ with $\delta' = \frac{\delta}{1-\epsilon}$ and $\beta \leq \epsilon$. Hence, by Lemma 7, the computed set $ALG\left(F^{k,\ell}_{\text{sml}}\right)$ is $(1+O(\epsilon), \beta)$-approximative, since $\frac{1+\sqrt{\delta'}}{1-\sqrt{\delta'}-\delta'} \cdot \frac{1}{1-\beta} \leq \frac{1+\epsilon}{1-\epsilon} = \frac{1+2\epsilon+\epsilon^2}{1-\epsilon^2} \in 1+O(\epsilon)$ (as ϵ goes to zero). □

At this point we have all necessary techniques for an approximation algorithm for the special case that all tasks are small. We will need this later for a separate result which shows that already with a running time of $O(n^4 \log n)$ we can obtain a constant factor approximation. For that result we define the constants $\epsilon, \beta, \delta, \ell$, and q differently. Therefore, we state the needed constants and their dependencies explicitly in the following lemma and give the approximation factor depending on them.

Lemma 9. *Let β, δ, q, ℓ be constants such that $\delta \leq (1-\beta)/2^\ell$, $2^{1-q} \leq \beta$, $\delta' := \delta/(1-\beta) < \frac{3-\sqrt{5}}{2}$, and $\beta < 1$. Consider the UFPP problem with the restriction that there is a value δ such that $d_i \leq \delta \cdot b(i)$ for each task i. For this problem there is a polynomial time approximation algorithm with running time $O(n^4 \log n)$ and approximation factor*

$$\frac{1+\sqrt{\delta'}}{1-\sqrt{\delta'}-\delta'} \cdot \frac{1}{1-\beta} \cdot \frac{\ell+q}{\ell}$$

Proof. Due to Lemmas 7 and 8 we can compute each $\left(\frac{1+\sqrt{\delta'}}{1-\sqrt{\delta'}-\delta'} \cdot \frac{1}{1-\beta}, \beta\right)$-approximative set $ALG\left(F^{k,\ell}_{\text{sml}}\right)$ in $O(n^3 \log n)$ steps. There are at most n such sets to compute. The computation of the sets $ALG\left(F^{k,\ell}_{\text{sml}}\right)$ dominate the overall running time. Hence, our algorithm needs $O(n^4 \log n)$ time. With Lemma 2 this yields an algorithm with an approximation factor of $\frac{1+\sqrt{\delta'}}{1-\sqrt{\delta'}-\delta'} \cdot \frac{1}{1-\beta} \cdot \frac{\ell+q}{\ell}$. □

Observe that Lemma 9 only depends on δ, β and ℓ. Therefore, for a given β the optimal value of q can be computed to be $\min\{q \in \mathbb{N} | 2^{1-q} \leq \beta\}$.

3.3. Approximation for Medium Tasks.

In this section we describe a dynamic program which computes an optimal solution for each set $F^{k,\ell}_{\text{med}}$. Recall that a task i is medium if $\delta \cdot b(i) < d_i \leq (1-2\beta)b(i)$. Afterwards, we show how to transform this set to obtain a $(2,\beta)$-approximative solution. With our framework stated in Lemma 2 this yields a 2-approximation algorithm for medium tasks.

First, we derive a technical property for medium tasks which follows from Lemma 3.

Lemma 10. *Let F be a feasible solution, and let e be an edge. For each set $F^{k,\ell}_{\text{med}}$ we have that $\left|F \cap F^{k,\ell}_{\text{med}} \cap T_e\right| \leq 2^{\ell+1}/\delta$.*

Proof. Applying Lemma 3 with $c' = 0$ shows that $\sum_{i \in F \cap F^{k,\ell} \cap T_e} d_i \leq 2^{k+\ell+1}$ for any feasible set F and any set $F^{k,\ell}$. Since $d_i > \delta \cdot b(i) \geq \delta \cdot 2^k$ for each $i \in F^{k,\ell}_{\text{med}}$ the claim follows. □

Now we show that there is a dynamic program for the sets $F^{k,\ell}_{\text{med}}$. A similar DP was given by Chakrabarti et al. [9].

Lemma 11. *There is a dynamic program which computes $OPT\left(F^{k,\ell}_{\text{med}}\right)$ for each set $F^{k,\ell}_{\text{med}}$ in time $n^{O\left(2^\ell/\delta\right)}$.*

Proof. We describe the dynamic program. For any edge e and set $S \subseteq F_{\text{med}}^{k,\ell} \cap T_e$, let $W_e(S)$ denote the maximum possible profit for a feasible solution $F \subseteq F_{\text{med}}^{k,\ell}$ that contains only tasks i with $s_i < e$, with $F \cap T_e = S$. We will show how to compute this for every e and relevant S, which yields the solution. By Lemma 10, it suffices to enumerate all sets $S \subseteq F_{\text{med}}^{k,\ell} \cap T_e$ which contain at most $2^{\ell+1}/\delta$ tasks from each set $F_{\text{med}}^{k,\ell} \cap T_e$. There can be at most $\binom{n}{d} \in O\left(n^d\right)$ such sets, with $d = 2^{\ell+1}/\delta$. Let S be such a set. Let e' be the edge immediately before e, and let $\mathcal{F}_{e'}$ denote all sets enumerated for e'. We check to what sets in $\mathcal{F}_{e'}$ the set S is compatible. A set $S' \in \mathcal{F}_{e'}$ is *compatible* to S if all tasks in $T_e \cap T_{e'}$ appear either in both S' and S or in none of the two sets. The reason for this definition is that some of the tasks in S also use e'. Hence, we are only interested in the configurations for S' in which those tasks appear as well. Moreover, when calculating $W_e(S)$ we need to bear in mind that the profits of the tasks in $S \cap S'$ are already counted in $W_{e'}(S')$. Denote by $\text{Comp}(S) \subseteq \mathcal{F}_{e'}$ all sets in $\mathcal{F}_{e'}$ which are compatible to S. Now it can be seen that the value $W_e(S)$ can be computed as follows:

$$W_e(S) := \max_{S' \in \text{Comp}(S)} \left\{ W_{e'}(S') + \sum_{i \in (T_e \setminus T_{e'}) \cap S} w_i \right\}$$

As seen above, the number of sets that need to be enumerated for each edge is bounded by $n^{O\left(2^\ell/\delta\right)}$. Hence, the calculation of $W_e(S)$ for each set S can be done in $\left(n^{O\left(2^\ell/\delta\right)}\right)^2 = n^{O\left(2^\ell/\delta\right)}$ steps. Since the number of edges is w. l. o. g. bounded by $2n$ (see Section 2) the claim follows. □

For being able to apply our framework, we turn $OPT\left(F_{\text{med}}^{k,\ell}\right)$ into a $(2,\beta)$-approximative solution. We describe the needed procedure in the next lemma.

Lemma 12. *For each set $F_{\text{med}}^{k,\ell}$ there is a set $ALG\left(F_{\text{med}}^{k,\ell}\right) \subseteq OPT\left(F_{\text{med}}^{k,\ell}\right)$ which is $(2,\beta)$-approximative. Moreover, the set $ALG\left(F_{\text{med}}^{k,\ell}\right)$ can be found in $O(n^2)$ time.*

Proof. We initialize two sets $H^1 := \emptyset$, $H^2 := \emptyset$. Assume that the tasks in $OPT\left(F_{\text{med}}^{k,\ell}\right)$ are ordered such that the start vertices are non-decreasing. We consider the tasks in $OPT\left(F_{\text{med}}^{k,\ell}\right)$ in this order. In the i-th iteration we take the task i. We add i to a set H^ℓ with $\ell \in \{1,2\}$ such that $H^\ell \cup \{i\}$ obeys the modified capacity constraint, i.e., it leaves a free capacity of $\beta \cdot 2^k$ in each edge.

It remains to show that indeed either $H^1 \cup \{i\}$ or $H^2 \cup \{i\}$ obeys the modified capacity constraint. Assume on the contrary that neither $H^1 \cup \{i\}$ nor $H^2 \cup \{i\}$ obey the modified capacity constraint. Then there are edges e, e' on the path of i such that

$$\sum_{i' \in (H^1 \cup \{i\}) \cap T_e} d_{i'} > u_e - \beta \cdot 2^k \tag{3.1}$$

and

$$\sum_{i' \in (H^2 \cup \{i\}) \cap T_{e'}} d_{i'} > u_{e'} - \beta \cdot 2^k. \tag{3.2}$$

Inequality 3.1 implies that $\sum_{i' \in H^1 \cap T_e} d_{i'} > u_e - \beta \cdot 2^k - d_i$. Inequality 3.2 gives that $\sum_{i' \in H^2 \cap T_{e'}} d_{i'} > u_{e'} - \beta \cdot 2^k - d_i$. Assume w.l.o.g. that $e < e'$ or $e = e'$. Recall that we considered the tasks by non-decreasing start index. Hence, all tasks in $(H^2 \cup \{i\}) \cap T_{e'}$ use e as well. For the next calculation we need that

$$d_i \leq (1-2\beta)b(i) \leq u_{e'}(1-2\beta)$$

and hence $u_{e'} - d_i \geq 2\beta \cdot u_{e'}$. Also, note that $u_{e'} \geq 2^k$ since $i \in OPT\left(F^{k,\ell}\right) \subseteq F^{k,\ell}$. We calculate that

$$\begin{aligned} u_e &\geq \left(\sum_{i' \in H^1 \cap T_e} d_{i'}\right) + \left(\sum_{i' \in H^2 \cap T_{e'}} d_{i'}\right) + d_i \\ &> \left(u_e - \beta \cdot 2^k - d_i\right) + \left(u_{e'} - \beta \cdot 2^k - d_i\right) + d_i \\ &= u_e + u_{e'} - d_i - 2\beta \cdot 2^k \\ &\geq u_e + 2\beta u_{e'} - 2\beta \cdot 2^k \\ &\geq u_e. \end{aligned}$$

This is a contradiction. Hence, task i can be added to one of the sets H^ℓ such that $H^\ell \cup \{i\}$ still obeys the capacity constraint. Finally, we define $ALG\left(F^{k,\ell}_{\text{med}}\right)$ to be the set among H^1 and H^2 with maximum profit. This ensures that $w\left(ALG\left(F^{k,\ell}_{\text{med}}\right)\right) \geq \frac{1}{2} \cdot w\left(OPT\left(F^{k,\ell}_{\text{med}}\right)\right)$.

When computing $ALG\left(F^{k,\ell}_{\text{med}}\right)$ we need to check for each task in $OPT\left(F^{k,\ell}_{\text{med}}\right)$ whether adding it to one of the sets violates the modified capacity constraint in one of the edges. Since w.l.o.g. $m < 2n$, this check can be done in $O(n)$ time for each task. There are n tasks in total, and hence the whole procedure can be implemented in $O(n^2)$. □

Now we have all necessary preparation to state our $(3+\epsilon)$-approximation algorithm for the case that for all tasks i it holds that $d_i \leq (1-2\beta)b(i)$, i.e., they are either small or medium.

Theorem 13. *Let $\epsilon > 0$, $\beta > 0$ and consider the UFPP problem with the restriction that all tasks i we have that $d_i \leq (1-\beta)b(i)$. There is a polynomial time $(3+\epsilon)$-approximation algorithm for this problem.*

Proof. Above, we defined a task i to be medium or small if $d_i \leq (1-2\beta)b(i)$. Based on this definition, we show how the above procedures yield a $(3+O(\epsilon))$-approximation algorithm. By working with an appropriately chosen value $\epsilon' \in O(\epsilon)$ rather than ϵ and a value $\beta' := \beta/2$ instead of β, this yields an $(3+\epsilon)$-approximation algorithm for tasks i with $d_i \leq (1-\beta)b(i)$.

For the given ϵ and β, we choose minimum integer values l and q such that $2^{1-q} \leq \beta$, and $\frac{l+q}{l} \leq 1+\epsilon$. For each set $F^{k,\ell}$ we compute $ALG\left(F^{k,\ell}_{\text{sml}}\right)$ and $ALG\left(F^{k,\ell}_{\text{med}}\right)$. We define $ALG\left(F^{k,\ell}\right)$ to be the set of maximum profit among the two. Due to Lemmas 8 and 12 we have that

$$\begin{aligned}(1+O(\epsilon))\,w(ALG\left(F^{k,\ell}_{\text{sml}}\right))+2w(ALG\left(F^{k,\ell}_{\text{med}}\right)) &\geq w(OPT\left(F^{k,\ell}_{\text{sml}}\right))+w(OPT\left(F^{k,\ell}_{\text{med}}\right)) \\ &\geq w(OPT\left(F^{k,\ell}\right))\end{aligned}$$

and hence $w(ALG\left(F^{k,\ell}\right))\geq(3+O(\epsilon))^{-1}\,w(OPT\left(F^{k,\ell}\right))$. With the framework of Lemma 2 this yields an α-approximation algorithm for the UFPP problem with the restriction that $d_i\leq(1-2\beta)b(i)$ for each task i, where $\alpha=(3+O(\epsilon))\,\frac{l+q}{l}\leq(3+O(\epsilon))\,(1+\epsilon)=3+O(\epsilon)+O(\epsilon^2)=3+O(\epsilon)$, as ϵ goes to zero.As argued above, this implies the claim of the theorem. □

3.4. **Resource Augmentation.** In this section we study the general UFPP problem – with small, medium, and large tasks – under resource augmentation. For an instance I of the problem denote by $(1+\beta)I$ a copy of I where the capacity of each edge is increased by a factor of $(1+\beta)$ (for a small value $\beta>0$). We present an algorithm that computes a set of tasks F whose total profit is at least $(2+O(\epsilon))^{-1}w(OPT(I))$ and that is feasible for $(1+\beta)I$.

When we increase the capacity of each edge by a factor of $1+O(\beta)$ then there are no large tasks anymore and Theorem 13 yields a $(3+O(\epsilon))$-approximation algorithm. However, we can do better. For each set $F^{k,\ell}_{\text{med}}$ it holds that $OPT\left(F^{k,\ell}_{\text{med}}\right)$ leaves some capacity in $(1+\beta)I$ unused. (Note here that $OPT\left(F^{k,\ell}_{\text{med}}\right)$ is still defined to be the optimal solution for $F^{k,\ell}_{\text{med}}$ in I.) Hence, in our framework we can use $OPT\left(F^{k,\ell}_{\text{med}}\right)$ instead of $ALG\left(F^{k,\ell}_{\text{med}}\right)$ and obtain a $(2+O(\epsilon))$-approximation.

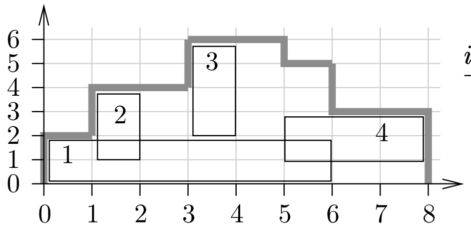
Theorem 14. *Let $\epsilon>0$, $\beta>0$ and let I be an instance of the UFPP problem. There is a polynomial time algorithm which computes a $(2+O(\epsilon))$-approximative solution which is feasible for $(1+\beta)I$.*

Proof. For the setting of resource augmentation, we change the notion of (α,β)-approximative sets slightly. Here, we define a set of tasks $F\subseteq F^{k,\ell}$ (for some values k,ℓ) to be (α,β)-approximative if $w(F)\geq\frac{1}{\alpha}\cdot w(OPT(F^{k,\ell}))$ and $\sum_{i\in F\cap T_e}d_i\leq u_e(1+\beta)-\beta\cdot 2^k$ for each edge e with $T_e\cap F^{k,\ell}\neq\emptyset$. According to this new definition, each set $OPT(F^{k,\ell})$ is (α,β)-approximative since

$$\begin{aligned}\sum_{i\in OPT(F^{k,\ell})\cap T_e}d_i &\leq u_e \\ &= u_e+\beta\cdot 2^k-\beta\cdot 2^k \\ &\leq u_e(1+\beta)-\beta\cdot 2^k\end{aligned}$$

for each edge e with $T_e\cap F^{k,\ell}\neq\emptyset$. For the purpose of this algorithm, we define a task to be medium if it is not small. Note that even with the new definition of medium tasks each set $OPT\left(F^{k,\ell}_{\text{med}}\right)$ can still be computed in polynomial time by the dynamic program described in Lemma 11 (for constant k and δ).

Consider a set $F^{k,\ell}$. We define $ALG\left(F^{k,\ell}\right)$ to be the set of maximum profit among $ALG\left(F^{k,\ell}_{\text{sml}}\right)$ and $OPT\left(F^{k,\ell}_{\text{med}}\right)$. We calculate that



i:	d_i:	w_i:	s_i:	t_i:
1	2	3	0	6
2	3	2	1	2
3	4	2	3	4
4	2	2	5	8

FIGURE 4.1. A UFPP instance consisting of a path of length eight (with capacities 2,4,4,6,6,5,3,3) and four tasks, represented using rectangles drawn as high as possible under the capacity profile.

$$\begin{aligned}(1+O(\epsilon))\,w(ALG\left(F^{k,\ell}_{\text{sml}}\right))+w(OPT\left(F^{k,\ell}_{\text{med}}\right)) &\geq w(OPT\left(F^{k,\ell}_{\text{sml}}\right))+w(OPT\left(F^{k,\ell}_{\text{med}}\right))\\ &\geq w(OPT\left(F^{k,\ell}\right)).\end{aligned}$$

Hence, $ALG\left(F^{k,\ell}\right)$ is $(2+O(\epsilon),\beta)$-approximative. Our framework defined in Lemma 2 can easily be adapted to the setting of resource augmentation. The only difference in the resource augmentation setting is that the profit of the computed solution is measured in comparison to $OPT(I)$ rather than $OPT((1+\beta)I)$. Hence, we obtain a $(2+O(\epsilon))$-approximation algorithm for the UFPP problem with factor $(1+\beta)$ resource augmentation. □

4. Large Tasks

In this section we provide a polynomial time constant factor approximation for instances consisting of only large tasks. Our main goal is to prove that for the task set F_{lrg} defined in Section 2.1, i.e. all tasks with relative demand at least $1/2$, we have a 4-approximation algorithm. However, we will prove a more general result: We assume that for a given integer $k \geq 2$, for all tasks i it holds that $d_i \geq b(i)/k$, and we will give a $2k$-approximation algorithm for this case. The main idea is to restrict to solutions of a certain form, so-called *top-drawn* solutions, which will be defined below. We will give a dynamic programming algorithm for finding optimal top-drawn solutions in Section 4.1. This dynamic program is based on a geometric viewpoint of the problem, where tasks are represented by rectangles, which must not overlap in a feasible solution. In Section 4.2 we will show that if $d_i \geq b(i)/k$ for all tasks i, the value of an optimal top-drawn solution is by at most a factor $2k$ worse than the value of an optimal (UFPP) solution. We will prove this by showing that any optimal solution can be partitioned into $2k$ top-drawn solutions. Together this gives the $2k$-approximation algorithm.

We first explain the geometric viewpoint behind top-drawn solutions, which will be used for the figures. A UFPP instance can be represented by a drawing in the plane as follows, see Figure 4.1. The vertices $0,\ldots,m$ of the path P are represented by the points $(0,0),\ldots,(m,0)$ on the x-axis. An edge $e=\{x,x+1\}\in E(P)$ with capacity u_e is represented by a horizontal line segment between (x,u_e) and $(x+1,u_e)$. We add vertical line segments to complete this into a closed curve from $(0,0)$ to $(m,0)$, called the *capacity profile* (shown in gray in the figure). The tasks are represented by rectangles drawn in the region enclosed by the capacity profile and the x-axis: the top of the rectangle is drawn at y-coordinate $b(i)$, the bottom at the y-coordinate $\ell(i)$. (Recall that $\ell(i) = b(i) - d_i$.) The left border of the rectangle is drawn at x-coordinate s_i, and the right border at t_i. So the height of the rectangle represents the demand of the task, and the left and right border the start and end point. Observe that the rectangle is drawn as high as possible under the capacity profile, which motivates the term *top-drawn.* A *top-drawn set* is now a set of tasks such that their rectangles, when drawn this way, pairwise do not overlap. To be precise, we say that two rectangles *overlap* when they share an internal point; touching is not considered overlapping. Two tasks for which the rectangles do not overlap are called *compatible.* The tasks in Figure 4.1 have profits 3, 2, 2, and 2, respectively. Therefore, an optimal UFPP solution consists of tasks 1, 3, and 4, with a total profit of 7. However, since the rectangles for the tasks 1 and 4 overlap, this is not a top-drawn set. The optimal top-drawn set consists of tasks 2, 3 and 4, with total profit 6. We now define these notions precisely.

Definition 15. *Two (distinct) tasks i and j are called* compatible *if at least one of the following holds:*

- $t_i \leq s_j$*,*
- $t_j \leq s_i$*,*
- $\ell(i) \geq b(j)$*, or*
- $\ell(j) \geq b(i)$*.*

Otherwise, i and j are called incompatible*. A set of tasks F is called a* top-drawn set *if all tasks in F are pairwise compatible.*

In Section 4.1, we will consider the optimization problem of finding top-drawn sets with maximum profit. First we observe that such sets are feasible solutions for UFPP, and thus we will call them *top-drawn solutions.*

Proposition 16. *Let $F \subseteq T$ be a top-drawn set. Then F is a feasible UFPP solution.*

Proof. Consider an edge $e \in E(P)$, and all tasks T_e that use e. Consider two tasks $i, j \in F \cap T_e$. They are compatible, but $s_i < e < t_j$ and $s_j < e < t_i$, so either $\ell(i) \geq b(j)$ or $\ell(j) \geq b(i)$ must hold. It follows that all tasks in $F \cap T_e$ can be numbered $i_1, \ldots, i_p$ such that $b(i_1) \leq \ell(i_2)$, $b(i_2) \leq \ell(i_3)$, etc. Hence

$$\begin{aligned}
\sum_{k=1}^{p} d_{i_k} &= \sum_{k=1}^{p} (b(i_k) - \ell(i_k)) \\
&= b(i_p) - \ell(i_1) + \sum_{k=1}^{p-1} \underbrace{(b(i_k) - \ell(i_{k+1}))}_{\leq 0} \\
&\leq u_e.
\end{aligned}$$

□

4.1. **A Dynamic Programming Algorithm for Finding Maximum Weight Top-drawn Sets.** To bound the complexity of our algorithm, it is useful to assume that all edge capacities are different. In the next lemma we first show that by making small perturbations to the capacities, this property can easily be guaranteed, without changing the set of feasible solutions.

Lemma 17. *Let $\mathcal{I}$ be a UFPP instance with task set $T = \{1, \dots, n\}$. In linear time, the capacities and demands can be modified such that for the resulting instance $\mathcal{I}'$:*

- *All edge capacities are distinct, and*
- *a set of tasks $F \subseteq T$ is feasible for $\mathcal{I}$ if and only if F is feasible for $\mathcal{I}'$.*

Proof. Recall that the m path edges are labeled $\{i, i+1\}$, for $i \in \{0, \dots, m-1\}$. For every edge $e = \{i, i+1\}$, we change the capacity to $u'_e = mu_e + i$. For every task $j \in T$, we change the demand to $d'_j = md_j$. This gives the instance $\mathcal{I}'$, in which all capacities are distinct.

Suppose $F \subseteq T$ is feasible for $\mathcal{I}'$. Then for every edge $e = \{i, i+1\}$, since all demands are integral,it holds that $\sum_{i \in T_e \cap F} d_i = \frac{1}{m} \sum_{i \in T_e \cap F} d'_i \leq \lfloor \frac{1}{m} u'_e \rfloor = \lfloor u_e + \frac{i}{m} \rfloor = u_e$. Hence F is also feasible for the original instance $\mathcal{I}$. Clearly, every task set F that is feasible for $\mathcal{I}$ remains feasible for $\mathcal{I}'$. □

The central concept of our dynamic program is that of a *corner* (x, y, z). First we give an informal, geometric explanation of this notion, using the representation of the problem explained before. Subsequently, we will give a formal definition.

A corner (x, y, z) corresponds to a certain region under the capacity profile, see Figure 4.2. In our dynamic program, we will compute for every such corner the profit of an optimal top-drawn solution that fits entirely within this region. This will be done using previously computed values for 'smaller' corners. A corner (x', y', z') is *smaller* than the corner (x, y, z) if its region is a strict subset of the region associated with (x, y, z) (we will make this more precise in the proof of Lemma 30).

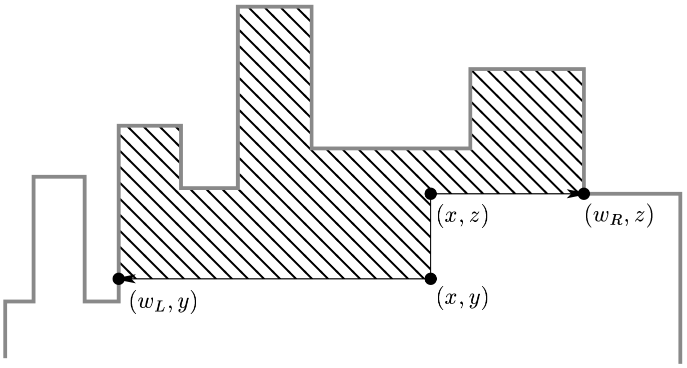


FIGURE 4.2. All (top-drawn) rectangles that fully lie in the shaded region fit into the corner $\mathrm{C}(x, y, z)$.

A corner is determined by an integer x-coordinate $0 \leq x \leq m$ (a path vertex), and two integer y-coordinates $y \geq 0$ and $z \geq 0$. If the capacity of the edge $\{x-1, x\} \in E(P)$ is more than y, then draw a horizontal line segment from the point (x, y) to the left, to the first point that lies on the capacity profile curve. This point will be called (w_L, y). Otherwise, if the capacity is at most y, then let $w_L = x$. Similarly, if the capacity of $\{x, x+1\} \in E(P)$ is more than z, then draw a horizontal line segment from (x, z) to the right, to the first point that lies on the capacity profile. This point will be called (w_R, z). Otherwise, let $w_R = x$. Connect these line segments into a single curve from (w_L, y) to (w_R, z) by adding a vertical line segment from (x, y) to (x, z). This curve and the capacity profile together now enclose a bounded region, which is shown as a shaded region in Figure 4.2. (In the special case that $u_{\{x,x+1\}} > z \geq u_{\{x-1,x\}} > y$ or $u_{\{x-1,x\}} > y \geq u_{\{x,x+1\}} > z$, the corner actually corresponds to two disjoint regions.) This is the region shown that we associate with the corner (x, y, z); we say that a task *fits into* the corner (x, y, z) when its rectangle, drawn as explained earlier, lies fully in (the closure of) this region. Now we give formal definitions.

Definition 18. *A triple (x, y, z) of integers is called a* corner *if $0 \leq x \leq m$, $y \geq 0$ and $z \geq 0$. For a corner (x, y, z), we denote by $w_L(x,y)$ or simply w_L the lowest numbered path vertex such that for all i with $w_L \leq i < x$, it holds that $u_{\{i,i+1\}} > y$. Similarly, $w_R(x,z)$ or simply w_R is defined to be the largest numbered path vertex such that for all i with $w_R \geq i > x$, it holds that $u_{\{i-1,i\}} > z$. So $w_L \leq x$ and $w_R \geq x$, but both may be equalities.*

We denote by $u_{\max} = \max_{e \in E(P)} u_e$ the maximum capacity of an edge.

Definition 19. *For a corner (x, y, z), by $\mathrm{C}(x, y, z)$ we denote the set of all tasks $i \in T$ for which at least one of the following holds:*

- *$w_L(x, y) \leq s_i$, $t_i \leq w_R(x, z)$ and $\ell(i) \geq \max\{y, z\}$,*
- *$w_L(x, y) \leq s_i$, $t_i \leq x$ and $\ell(i) \geq y$, or*
- *$x \leq s_i$, $t_i \leq w_R(x, z)$ and $\ell(i) \geq z$.*

We say that *task i fits into the corner (x, y, z)* or *corner (x, y, z) contains i* if $i \in \mathrm{C}(x, y, z)$.

For a given UFPP instance and corner (x, y, z), we denote by $P(x, y, z)$ the maximum value of $w(F)$ over all top-drawn sets F with $F \subseteq \mathrm{C}(x, y, z)$. A top-drawn task set F with $F \subseteq \mathrm{C}(x, y, z)$ and $w(F) = P(x, y, z)$ is said to *determine* $P(x, y, z)$. We will now show how $P(x, y, z)$ can be computed in various cases. First we observe that all tasks fit into the corner $(m, 0, u_{\max})$, so computing $P(m, 0, u_{\max})$ yields the desired value:

Proposition 20. *$P(m, 0, u_{\max})$ equals the value of an optimal top-drawn solution, where m is the path length.*

We first consider two simple cases in which different corners (integer triples) define the same region. The following two propositions follow easily from the definitions.

Proposition 21. *If $y = z$, then $\mathrm{C}(x, y, z) = \mathrm{C}(w_R(x, z), y, u_{\max})$.*

Because of this proposition, we only have to consider corners (x, y, z) where $y < z$ or $y > z$. Since these cases are symmetric, *we will only consider corners (x, y, z) for which $y < z$ holds, in the following definitions and lemmas.* One can easily deduce the corresponding statements for the case $y > z$. For the sake of brevity and readability, we will however not explicitly treat this case.

Proposition 22. *If $x = 0$ or $y \geq u_{\{x-1,x\}}$ then $\mathrm{C}(x, y, z) = \mathrm{C}(x, u_{\max}, z)$.*

The above observations show that we may restrict our attention to *standard* corners, defined as follows. Within this category we distinguish two further corner types.

Definition 23. *A corner (x, y, z) is called* standard *if $y < z$, $x \geq 1$ and $y < u_{\{x-1,x\}}$. A standard corner (x, y, z) is called* split *if $u_{\{x-1,x\}} \leq z < u_{\{x,x+1\}}$, and* proper *otherwise.*

In the case of a split corner, we can give a simple expression for $P(x, y, z)$.

Proposition 24. *Let (x, y, z) be a split corner. Then $P(x, y, z) = P(x, y, u_{\max}) + P(x, u_{\max}, z)$.*

Proof. Consider a top-drawn set F that determines $P(x, y, z)$. Since $u_{\{x-1,x\}} \leq z$, every task $i \in F$ with $s_i \leq x - 1$ and $t_i \geq x$ has $b(i) \leq z$. Hence F contains no tasks i with $s_i \leq x - 1$ and $t_i \geq x + 1$, and thus can be partitioned into two top-drawn sets that fit into the corners $(x, y, u_{\max})$ and $(x, u_{\max}, z)$ respectively. It follows that $P(x, y, z) \leq P(x, y, u_{\max}) + P(x, u_{\max}, z)$.

Now let F_1 and F_2 be top-drawn sets that determine $P(x, y, u_{\max})$ and $P(x, u_{\max}, z)$. These are disjoint and both fit in the corner (x, y, z), so $P(x, y, z) \geq w(F_1) + w(F_2) = P(x, y, u_{\max}) + P(x, u_{\max}, z)$. □

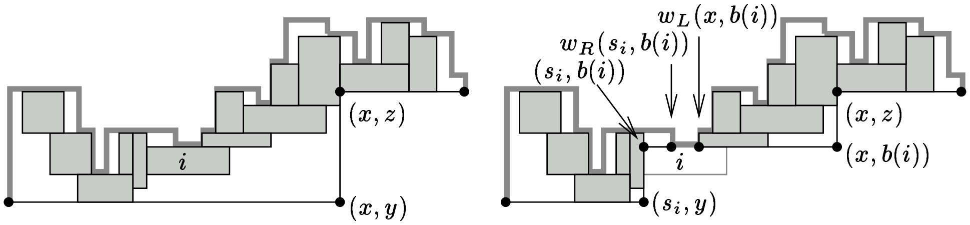


FIGURE 4.3. On the left, a top-drawn set $F \subseteq \mathrm{C}(x, y, z)$ is shown. When choosing a special task $i \in F$ with no tasks below or to the right of it, $F \backslash \{i\} \subseteq \mathrm{C}(s_i, y, b(i)) \cup \mathrm{C}(x, b(i), z)$.

Now we will consider the more complex case where the given corner (x, y, z) is proper. The main idea is as follows. Consider a top-drawn set F that determines $P(x, y, z)$. Either F also fits into the smaller corner $(x - 1, y, z)$, or there exists a task j with $\ell(j) < z$ and $t_j = x$. In the latter case, we show that F can be partitioned into two task sets that fit into smaller corners, and one single task. Given a task i, we will consider the two smaller corners $(s_i, y, b(i))$ and $(x, b(i), z)$. These are illustrated in Figure 4.3. For the indicated task i and the depicted top-drawn set F, we have the desired property that $F \backslash \{i\} \subset \mathrm{C}(s_i, y, b(i)) \cup \mathrm{C}(x, b(i), z)$. We will show that there always exists a task i for which this holds; such a task is called *special.* This explains the following recursive formula.

Lemma 25. *Consider a UFPP instance in which all edge capacities are distinct. Let (x, y, z) be a proper corner. Then*

$$P(x, y, z) = \max\left\{P(x-1, y, z), \max_{i \in \mathrm{C}(x,y,z), t_i \leq x} \left\{w_i + P\big(s_i, y, b(i)\big) + P\big(x, b(i), z\big)\right\}\right\}.$$

Before we can prove Lemma 25, we will define which properties are required for a special task, and prove that such a task always exists. Informally, the essential property of a special task i is that the rectangular region that is defined by the

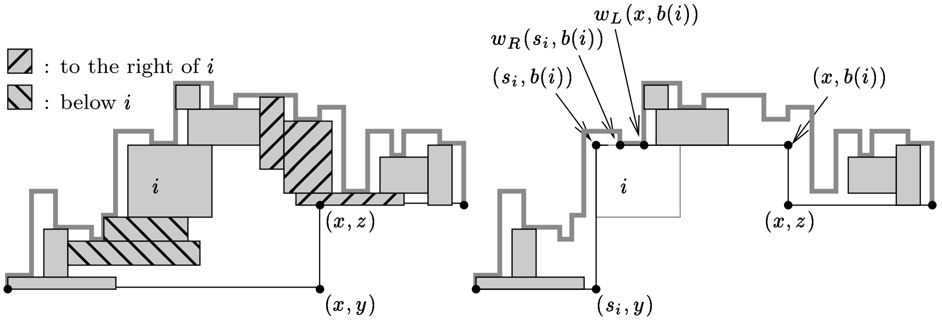


FIGURE 4.4. On the left, a top-drawn set $F \subseteq \mathrm{C}(x,y,z)$ is shown. The corners $(s_i,y,b(i))$ and $(x,b(i),z)$, given by a non-special candidate task $i \in F$, contain all tasks in F that do not lie below i or to the right of i.

horizontal interval $[s_i,x]$ and the vertical interval $[y,b(i)]$ does not overlap with any task rectangle other than i, and does not overlap with the capacity profile. In this case we will say that there are no tasks in F that *lie to the right of* i or *lie below* i. Figure 4.4 illustrates the case where a (non-special) task i is chosen, such that the rectangular region *does* overlap with other task rectangles. We now define this precisely.

Definition 26. *Consider a proper corner (x,y,z), and a task $i \in \mathrm{C}(x,y,z)$ with $t_i \leq x$.*

- *A task $j \in \mathrm{C}(x,y,z)$* lies to the right *of i if $t_i \leq s_j < x$ and $\ell(j) < b(i)$.*
- *A task $j \in \mathrm{C}(x,y,z)$* lies below *i if $b(j) \leq \ell(i)$ and $t_j > s_i$.*
- *A task $i \in \mathrm{C}(x,y,z)$ is called a* candidate *for (x,y,z) if*
  - *$t_i \leq x$,*
  - *$\ell(i) < z$ and*
  - *for all edges $e \in E(P)$ with $s_i < e < x$, $u_e \geq b(i)$ holds.*

Definition 27. *Let (x,y,z) be a proper corner and $F \subseteq \mathrm{C}(x,y,z)$ be a top-drawn set. A task $i \in F$ is called* special *(with respect to F and (x,y,z)) if it is a candidate for (x,y,z), and there is no task $j \in F$ that lies to the right of i or that lies below i.*

Lemma 28. *Let (x,y,z) be a proper corner and $F \subseteq \mathrm{C}(x,y,z)$ be a top-drawn set. Then at least one of the following holds:*

- *$F \subseteq \mathrm{C}(x-1,y,z)$, or*
- *there exists a special task $i \in F$ (with respect to F and (x,y,z)).*

Proof. From $u_{\{x-1,x\}} > y$ it follows that $w_L(x,y) = w_L(x-1,y)$. Since the corner is not split, we have that $w_R(x-1,z) = w_R(x,z)$ in the case that $z < u_{\{x,x+1\}}$ (and thus $z < u_{\{x-1,x\}}$), and $w_R(x-1,z) \leq w_R(x,z) = x$ in the case that $z > u_{\{x,x+1\}}$. Therefore, since $y < z$ it holds that $\mathrm{C}(x-1,y,z) \subseteq \mathrm{C}(x,y,z)$. Furthermore, this shows that the only case when $F \subseteq \mathrm{C}(x-1,y,z)$ does not hold is when there is a task $j \in F$ with $t_j = x$ and $\ell(j) < z$. Hence it is easily seen that j is a candidate (for (x,y,z)). In addition, since $t_j = x$, no task in F lies to the right of j. So to prove the lemma statement, we may assume that there exists at least one candidate task in F with no tasks to the right of it. Choose $i \in F$ to be such a task with minimum value for $b(i)$. We prove that no task in F lies below i, which will prove the lemma. Suppose to the contrary that a task $j \in F$ lies below i, so $b(j) \leq \ell(i) < b(i)$ and $t_j > s_i$. It is easily checked that j is a candidate as well. By choice of i, it must then hold that there exists a task $k \in F$ that lies to the right of j (otherwise j should have been chosen in the role of i). So $t_j \leq s_k < x$, and $\ell(k) < b(j) \leq \ell(i) < b(i)$. Therefore, $s_k < t_i$; otherwise k would also lie to the right of i. But now we can use the fact that i and k are compatible: Recall that we have that $s_k < t_i$, $s_i < t_j \leq s_k$, and $\ell(k) < b(j) < b(i)$. So the only case in which i and k can be compatible is when $b(k) \leq \ell(i) < b(i)$. This however contradicts the fact that all edges between s_i and x have capacity at least $b(i)$; the edge that determines $b(k)$ lies between s_i and x. We conclude that i is a candidate task that satisfies the additional properties that there are no tasks in F below it, or to the right of it, which therefore is special. □

Now we can prove Lemma 25.

Proof of Lemma 25. Let (x, y, z) be a proper corner in a UFPP instance in which all capacities are distinct. We will first show that

(4.1)
$$P(x,y,z) \leq \max\left\{P(x-1,y,z), \max_{i \in \mathrm{C}(x,y,z), t_i \leq x} \left\{w_i + P\big(s_i, y, b(i)\big) + P\big(x, b(i), z\big)\right\}\right\}.$$

Let F be a top-drawn set that determines $P(x,y,z)$. If $F \subseteq \mathrm{C}(x-1,y,z)$ then $P(x,y,z) \leq P(x-1,y,z)$. If not, then by Lemma 28, there exists a special task i. We argue that $F \backslash \{i\}$ can be partitioned into two top-drawn sets F_1 and F_2 with $F_1 \subseteq \mathrm{C}\left(s_i, y, b(i)\right)$ and $F_2 \subseteq \mathrm{C}\left(x, b(i), z\right)$, which will prove that $P(x,y,z) \leq \Big\{w_i + P\big(s_i, y, b(i)\big) + P\big(x, b(i), z\big)\Big\}$, and therefore the Inequality 4.1. First we consider the w_L and w_R vertices for both corners. Let $e_B = \{v, v+1\}$ be a bottleneck edge for i, which is unique since all capacities are distinct. So $u_{e_B} = b(i)$ and $s_i < e_B < t_i$. For the corner $(s_i, y, b(i))$, it is easy to see that $w_L(s_i, y) = w_L(x, y)$ and $w_R(s_i, b(i)) = v$. Now consider the corner $(x, b(i), z)$. Obviously the w_R value is the same as for $\mathrm{C}(x,y,z)$. In addition, since i is special, all edges e with $s_i < e_B < e < x$ have $u_e \geq b(i)$. Therefore, since we assumed all edge capacities are distinct, $u_e > b(i)$ holds for all such edges e. It follows that $w_L(x, b(i)) = v + 1$.

Now consider a task $j \in F \backslash \{i\}$. We distinguish five cases.

(1) $t_j \leq s_i$: Since $w_L(x,y) = w_L(s_i,y)$ and $\ell(j) \geq y$, we have that $j \in \mathrm{C}(s_i, y, b(i))$.
(2) $s_i < t_j < e_B$: Since j is compatible with i but does not lie below i, $\ell(j) \geq b(i)$ holds. Therefore $j \in \mathrm{C}(s_i, y, b(i))$.
(3) $s_j \geq x$: From $j \in \mathrm{C}(x,y,z)$ it easily follows that $j \in \mathrm{C}(x, b(i), z)$.
(4) $e_B < s_j < x$: Since j is compatible with i but does not lie to the right of i, $\ell(j) \geq b(i)$ must hold. In addition, if $t_j > x$ then $\ell(j) \geq z$. Therefore $j \in \mathrm{C}(x, b(i), z)$.
(5) $s_j < e_B$ and $e_B < t_j$: We argue that this is not possible. We have that $b(j) \leq u_{e_B} = b(i)$. Because i and j are compatible, it then must be that $b(j) \leq \ell(i)$. But this contradicts that there is no task below i.

Since we considered all possibilities, this concludes the proof of Inequality 4.1.

Next, we argue that

(4.2)

$$P(x,y,z) \geq \max\left\{P(x-1,y,z), \max_{i\in \mathrm{C}(x,y,z),t_i\leq x}\Big\{w_i+P\big(s_i,y,b(i)\big)+P\big(x,b(i),z\big)\Big\}\right\}.$$

Recall that since (x,y,z) is a standard corner, $\mathrm{C}(x-1,y,z) \subseteq \mathrm{C}(x,y,z)$, so $P(x,y,z) \geq P(x-1,y,z)$. Now consider a task $i \in \mathrm{C}(x,y,z)$ with $t_i \leq x$. Let F_1 and F_2 be the top-drawn sets that determine $P(s_i,y,b(i))$ and $P(x,b(i),z)$, respectively. We will argue that $F_1 \cap F_2 = \emptyset$ and $F_1 \cup F_2 \subseteq \mathrm{C}(x,y,z)$. Since clearly $i \notin F_1$ and $i \notin F_2$, it will follow that $P(x,y,z) \geq w_i + P(s_i,y,b(i)) + P(x,b(i),z)$, proving Inequality 4.2.

Now let e_B a bottleneck edge for i. Then $w_R(s_i,b(i)) < e_B$. On the other hand, since $t_i \leq x$, it holds that $w_L(x,b(i)) > e_B$. It follows that $F_1 \cap F_2 = \emptyset$. Finally, we prove that both sets fit in the corner (x,y,z). Recall that $w_R(s_i,b(i)) < e_B < t_i \leq x$. Since in addition $b(i) > y$, it follows that $\mathrm{C}(s_i,y,b(i)) \subseteq \mathrm{C}(x,y,z)$. (Because $w_R(s_i,b(i)) < x$, it is irrelevant whether $b(i) > z$ or not.) It is obvious that $\mathrm{C}(x,b(i),z) \subseteq \mathrm{C}(x,y,z)$, since $b(i) > y$. This concludes the proof of Inequality 4.2. □

Algorithm 1: $\mathcal{P}(x,y,z)$: a recursive algorithm for computing $P(x,y,z)$

```
Input: A UFPP instance where $u_{\max}$ is the maximum edge capacity, all edge
       capacities are distinct, and integers $0 \leq x \leq m$, $0 \leq y \leq u_{\max}$,
       $0 \leq z \leq u_{\max}$.
Output: $P(x,y,z)$.
if $x = 0$ or $y \geq u_{\{x-1,x\}}$ then $y := u_{\max}$
if $x = m$ or $z \geq u_{\{x,x+1\}}$ then $z := u_{\max}$
Compute $w_L := w_L(x,y)$ and $w_R := w_R(x,z)$
if $w_L = w_R$ then return 0
if $y = z$ then $z := u_{\max}$; $x := w_R$
if $y < z$ then
    if $u_{\{x-1,x\}} \leq z < u_{\{x,x+1\}}$ then return $(\mathcal{P}(x,y,u_{\max}) + \mathcal{P}(x,u_{\max},z))$
    return max$\Big\{\mathcal{P}(x-1,y,z),$
                   $\max_{i\in\mathrm{C}(x,y,z),t_i\leq x}\Big(w_i + \mathcal{P}\big(s_i,y,b(i)\big) + \mathcal{P}\big(x,b(i),z\big)\Big)\Big\}$
end
(In the remaining case, $y > z$ holds:)
if $u_{\{x,x+1\}} \leq y < u_{\{x-1,x\}}$ then return $(\mathcal{P}(x,y,u_{\max}) + \mathcal{P}(x,u_{\max},z))$
return max$\Big\{\mathcal{P}(x+1,y,z),$
               $\max_{i\in\mathrm{C}(x,y,z),s_i\geq x}\Big(w_i + \mathcal{P}\big(t_i,b(i),z\big) + \mathcal{P}\big(x,y,b(i)\big)\Big)\Big\}$
```

Our recursive routine $\mathcal{P}(x,y,z)$ for computing $P(x,y,z)$ is summarized in Algorithm 1. We first argue that the correct answer is returned.

Lemma 29. *When given a corner (x,y,z), Algorithm 1 returns $P(x,y,z)$.*

Proof. We prove the statement by induction over the total number of recursive calls. That is, we will prove that a call $\mathcal{P}(x,y,z)$ returns the value $P(x,y,z)$, assuming that this holds for recursive calls $\mathcal{P}(x', y', z')$ that may be made. (Note that we prove below in Lemma 30 that the algorithm indeed terminates.)

The modification in Line 1 yields an equivalent corner by Proposition 22. By symmetry, the same holds for Line 2. If $w_L = w_R$ then $\mathrm{C}(x, y, z) = \emptyset$, so the correct answer is returned in Line 4. The modification in Line 5 is correct as well (Proposition 21). So we may assume now that $w_L < w_R$, and $y \neq z$.

Consider the case that $y < z$. This implies that $y < u_{\max}$, so $x \geq 1$ and $y < u_{\{x-1,x\}}$, and thus (x, y, z) is a standard corner. If $u_{\{x-1,x\}} \leq z$ and $z < u_{\{x,x+1\}}$, then it is a split corner, so Proposition 24 shows that Line 7 returns the correct answer. So if Line 8 is reached, the considered corner is proper. By Lemma 25, Line 8 returns the correct answer.

The case that $y > z$ is analog; by symmetric arguments, the answers given in Lines 9 and 10 are correct. This shows that Algorithm 1 returns the correct answer in every case. □

Now we show that the recursion terminates, by showing that all recursive calls are made with arguments (x', y', z') that are 'smaller' than the original argument (x, y, z).

Lemma 30. *Algorithm 1 terminates.*

Proof. We show that when given an input (x, y, z), all recursive calls will be with arguments (x', y', z') such that the corner (x', y', z') is strictly smaller than (x, y, z), with respect to the following size measure: For a corner (x, y, z) with $y \leq u_{\max}$ and $z \leq u_{\max}$, we define

$$\mathrm{area}(x, y, z) = (x - w_L)(u_{\max} - y) + (w_R - x)(u_{\max} - z).$$

Note that for all corners (x, y, z) that may be considered, $\mathrm{area}(x, y, z)$ is a non-negative integer. Hence the lemma statement then follows.

A recursive call is made in Line 7 whenever a corner (x, y, z) is considered that satisfies the following bounds. Firstly $y < z \leq u_{\max}$, and therefore $x \geq 1$ and $y < u_{\{x-1,x\}}$. Hence (x, y, z) is a standard corner, and $w_L < x$. If the if-condition is satisfied then $w_R > x$ holds. This shows that for both calls $\mathcal{P}(x, y, u_{\max})$ and $\mathcal{P}(x, u_{\max}, z)$, the size measure area strictly decreases. Now consider Line 8, which is reached whenever the corner is a proper corner. In this case, when considering the corner $(x - 1, y, z)$, the w_L value does not change, and the w_R value may only decrease (see the proof of Lemma 28). Since $y < z$ it then follows that $\mathrm{area}(x - 1, y, z) < \mathrm{area}(x, y, z)$. Now consider a task $i \in \mathrm{C}(x, y, z)$ with $t_i \leq x$. We have that $w_R(s_i, b(i)) < x$ and $b(i) > y$, so $\mathrm{area}(s_i, y, b(i)) < \mathrm{area}(x, y, z)$. Clearly, $\mathrm{area}(x, b(i), z) < \mathrm{area}(x, y, z)$. So in all recursive calls in Line 8, the area decreases. By symmetry, the same holds for Lines 9 and 10. □

Theorem 31. *An optimum top-drawn set for a UFPP instance can be computed in time $O(nm^3) \subseteq O(n^4)$.*

Proof. The algorithm is as follows. First, in time $O(n + m)$, we transform the given instance into an equivalent instance, in which all edge capacities are distinct (Lemma 17). As mentioned in Section 2 in time $O(n + m)$ we can transform the instance to an equivalent instance in which all vertices occur as the start or end vertex of some task, so we may assume that $m < 2n$. This preprocessing does not change n and cannot increase m. We call Algorithm 1, with arguments

$(x, y, z) = (m, 0, u_{\max})$, to obtain the total profit of an optimal top-drawn set (Proposition 20, Lemma 29). With a straightforward extension, we can compute an optimal top-drawn set itself.

A small modification is necessary to obtain a time complexity of $O(nm^3)$: whenever any value $P(x, y, z)$ is computed during the course of computation, this value is stored in a table. Whenever a recursive call to $\mathcal{P}(x, y, z)$ would be made, the algorithm instead returns the stored value $P(x, y, z)$, if it has been computed before (in a different recursion branch). This, together with Lemma 30, ensures that for every corner (x, y, z) the routine $\mathcal{P}(x, y, z)$ is called at most once during the course of computation. (Lemma 30 ensures that a given corner is not considered more than once in the *same* recursion branch.)

We call a corner (x, y, z) *relevant* if $y = 0$ or y is equal to the capacity of some edge, and if the same holds for z. Observe that when calling Algorithm 1 with a relevant corner as argument, then every possible recursive call is with a relevant corner as well. So the total number of recursive calls is bounded by the total number of relevant corners. There are at most $m+1$ possibilities for x, and at most $m+1$ for y and z (since these need to correspond to actual capacities), so the total number of recursive calls is bounded by $O(m^3)$. Now consider the complexity of a single call. Line 3 can be done in time $O(m)$. Ignoring recursive calls, Lines 7 and 9 take constant time. Lines 8 and 10 take time $O(n)$; for every task i, in constant time one can decide whether $i \in \mathrm{C}(x, y, z)$ and $t_i \leq x$ (resp. $s_i \geq x$), and evaluate the given expressions. The remaining lines clearly take constant time, so it follows that a single call of Algorithm 1 takes time $O(n) + O(m) \subseteq O(n)$. Hence the total complexity becomes $O(nm^3) \subseteq O(n^4)$. □

4.2. **The Coloring Lemma.** For any integer $k \geq 2$, we call a task i *$1/k$-large* if its relative demand is at least $1/k$, that is, $d_i \geq 1/k \cdot b(i)$.Our goal in this section is to show that if F is a feasible solution to an UFP instance, where all tasks in F are $1/k$-large, then F can be partitioned into $2k$ top-drawn sets.

A partition $\{F_1, \dots, F_\ell\}$ of a task set F into ℓ top-drawn sets will be encoded by an *ℓ-coloring* α of F. This is a function $\alpha : F \to \{1, \dots, \ell\}$ such that if $\alpha(i) = \alpha(j)$, then i and j are compatible. (So $\alpha(i) = j$ means that $i \in F_j$.) Now we can formulate the main lemma of this section.

Lemma 32. *Let F be a feasible UFPP solution, and let $k \geq 2$ be an integer, such that every task in F is $1/k$-large. Then there exists a $2k$-coloring for F.*

Before we can prove Lemma 32, we first need some more definitions and the following proposition.

Proposition 33. *Let F be a feasible UFPP solution, and let $k \geq 2$ be an integer, such that every task in F is $1/k$-large. Let e be a bottleneck edge for $i \in F$. Then there are at most $k - 1$ tasks $j \in F \backslash \{i\}$ that are incompatible with i and use e.*

Proof. Let the set $F' \subseteq F \backslash \{i\}$ consist of all tasks that are incompatible with i and that use e. Suppose $|F'| \geq k$. Then

$$d_i + \sum_{j \in F'} d_j \geq d_i + \frac{1}{k} \sum_{j \in F'} b(j) > d_i + \frac{1}{k} |F'| (b(i) - d_i) \geq b(i) \geq u_e,$$

contradicting that F is a feasible solution. □

An edge e is called a *separator* for a task set F if it is a bottleneck edge for some task $i \in F$, such that all tasks in F that use e are incompatible with i. So by the above proposition, there are at most k tasks in F that use e, when F is a UFPP solution consisting of $1/k$-large tasks. A coloring α of F is called *nice* if for every edge e and task $i \in F$ that has e bottleneck, all tasks that use e and are incompatible with i are colored differently. The main idea behind these notions and our construction of a coloring is as follows: We will identify a separator edge e, and consider the set F_0 of tasks i with $s_i < e$, and F_1 of tasks i with $t_i > e$. (Note that $F_0 \cup F_1 = F$ and $F_0 \cap F_1 \neq \emptyset$.) Unless $F_0 = F$ or $F_1 = F$, we may use induction to conclude that both admit a nice $2k$-coloring. Then, since e is a separator edge and the colorings are nice, all tasks in $F_0 \cap F_1$ are colored differently in both colorings. Therefore these can be combined into a single nice $2k$-coloring for F. However, since it may occur that $F_0 = F$ or $F_1 = F$, we need a slightly more sophisticated argument, which we will now present in detail.

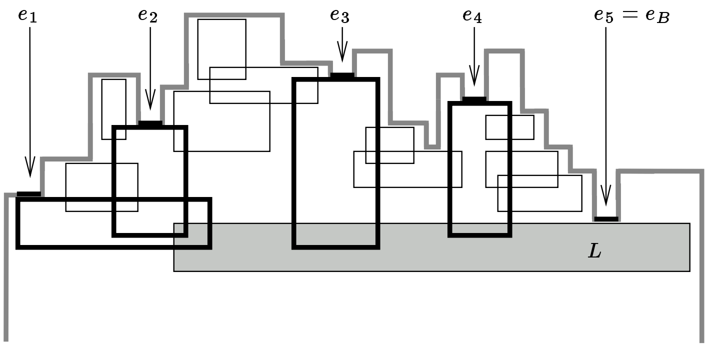


FIGURE 4.5. An illustration of the proof of Lemma 32 for the case $k = 2$ (using logarithmic scale on the y-axis, all tasks are $\frac{1}{2}$-large). The gray task is the single task in L, and the bold tasks are associated with the indicated separator edges $e_1, \ldots, e_5$.

Proof of Lemma 32. Now we will prove the main lemma, by induction over $|F|$. In fact, we will prove that the following stronger statement holds:

Let F be a feasible solution consisting of $1/k$-large tasks. Then there exists a nice $2k$-coloring for F.

The statement is trivially true when $|F| \leq 2k$. Now suppose that $|F| > 2k$, and assume that the above statement holds for every such set F' with $|F'| < |F|$. The proof is illustrated in Figure 4.5. Let e_B be a bottleneck edge for some task in F, with minimum capacity among all such edges. Let $L \subseteq F$ be the set of tasks that use e_B. By Proposition 33 and the choice of e_B, $|L| \leq k$.

Let $E_S = \{e_1, \ldots, e_p\}$ be the set of edges e for which there is a task i with e as bottleneck edge, such that $i \in L$ or i is incompatible with some $j \in L$. (So $e_B \in E_S$.) The edges in E_S are numbered such that if $x < y$, then $e_x < e_y$.

We first argue that all edges in E_S are separators for F. For $e_B \in E_S$, the statement is clear since e_B is a bottleneck edge with minimum capacity. Now consider an edge $e \in E_S \backslash \{e_B\}$, and let $i \in F$ be a task with e as bottleneck, that is incompatible with some task $i' \in L$. Then $\ell(i) < b(i')$ holds. Now suppose there is a task i'' using e that is compatible with i. Since both i and i'' use e and e is a bottleneck edge of i, from their compatibility it follows that $b(i'') \leq \ell(i) < b(i') = e_B$. But this contradicts that e_B is a bottleneck edge for F with minimum capacity.

Now, for $1 \leq j \leq p-1$, define $F_j \subseteq F$ to be all tasks i with $s_i < e_{j+1}$ and $t_i > e_j$. Similarly, define $F_0 \subseteq F$ to be all tasks i with $s_i < e_1$, and define $F_p \subseteq F$ to be all tasks i with $t_i > e_p$. Observe that $F_0 \cup \ldots \cup F_p = F$.

CASE 1: For every j, $|F_j| < |F|$.

In this case, we may use induction to conclude that for every F_j there exists a nice $2k$-coloring α_j. We can combine these into a nice $2k$-coloring of F: all tasks in $C = F_0 \cap F_1$ use the edge e_1. Since e_1 is a separator, there exists a task $i \in C$ with e_1 as bottleneck such that all tasks using e_1 are incompatible with i. So, in both the nice $2k$-coloring α_0 of F_0 and the nice $2k$-coloring α_1 of F_1, all tasks in C are colored differently. Therefore, we can permute the colors of α_1 such that the tasks in C receive the same colors in α_0 and α_1. At this point, the colorings can be combined into a $2k$-coloring α' of $F_0 \cup F_1$, which is again nice.

Next, we can combine the nice $2k$-coloring α_2 of F_2 with the nice $2k$-coloring α' of $F_0 \cup F_1$ in a similar way, and continue like this with $F_3, \ldots, F_p$, until a nice $2k$-coloring of $F = F_0 \cup \ldots \cup F_p$ is obtained. This proves the desired statement in this case.

CASE 2: There exists a j such that $F_j = F$.

For such a choice of j, define $F'_j = F_j \backslash L$. Since L is non-empty, $|F'_j| < |F|$ holds, so we may again use induction to conclude that F'_j admits a nice $2k$-coloring α. We will extend this to a nice $2k$-coloring of $F_j = F$. For ease of exposition we will assume that $1 \leq j < p$ (the cases $j = 0$ and $j = p$ are similar but easier). So both e_j and e_{j+1} are edges in E_S. We assume also w.l.o.g. that $e_B \geq e_{j+1}$.

We observe that every task $i \in F'_j$ that is incompatible with some task in L uses either e_j or e_{j+1}: by definition of E_S, such a task i must use some edge $e_x \in E_S$. If $x > j+1$ then by definition of F_j, i also uses e_{j+1}. If $x < j$ then similarly i also uses e_j.

Since $e_B \geq e_{j+1}$ and $F_j = F$, all tasks in L use e_{j+1}. Then, since e_{j+1} is a separator, there are at most $k - |L|$ tasks in F'_j that use e_{j+1} (Proposition 33); let F_R denote these tasks. In addition, there are at most k tasks in F'_j that use e_j; denote these by F_L. Since $|F_L| + |F_R| \leq 2k - |L|$ and there are $2k$ available colors, we can choose $|L|$ different colors that are not used for tasks in $F_L \cup F_R$, in the coloring α. We extend the nice $2k$-coloring α for F'_j to $F_j = F$ by using these $|L|$ colors for the tasks in L. Since we observed that all tasks in F'_j that are incompatible with some task in L are included in $F_L \cup F_R$, it follows that this yields again a nice $2k$-coloring for L. Since we now constructed a nice $2k$-coloring for F in all cases, this concludes the proof of Lemma 32. □

We observe that the bound from Lemma 32 is tight for every $k \geq 2$: consider a path P of length 5, with capacities $2k^2$, $2k^2 + 2k$, $2(2k^2 + 2k)$, $2k^2 + 2k$ and $2k^2$, in order along the path. (So $V(P) = \{0, \ldots, 5\}$.) Introduce $k-1$ tasks with $s_i = 0$, $t_i = 3$ and $d_i = 2k+1$, $k-1$ tasks with $s_i = 2$, $t_i = 5$ and $d_i = 2k+1$, one task with

$s_i = 1$, $t_i = 3$ and $d_i = 2k+3$, and one task with $s_i = 2$, $t_i = 4$ and $d_i = 2k+3$. All tasks can be verified to be $\frac{1}{k}$-large. In addition, they all satisfy $b(i) \geq 2k^2 > \ell(i)$ and $s_i \leq 2 < 3 \leq t_i$, so all are pairwise incompatible, and therefore at least $2k$ colors are needed. Finally, they constitute a feasible UFPP solution: for the first and fifth edge this is easy to see. For the second and fourth edge, the sum of demands using the edge is $(k-1)(2k+1) + (2k+3) = 2k^2 + k + 2 \leq 2k^2 + 2k = u_{(1,2)} = u_{(3,4)}$, since $k \geq 2$. All tasks use the third edge, so the demand sum is $2 \cdot ((k-1)(2k+1) + (2k+3)) = 2 \cdot (2k^2 + k + 2) = 4k^2 + 2k + 4 \leq 4k^2 + 4k = u_{\{2,3\}}$, since $k \geq 2$.

We summarize the results of Section 4 in the following theorem.

Theorem 34. *For every integer $k \geq 2$, there is a $O(n^4)$ time $2k$-approximation algorithm for the UFPP problem restricted to instances where every task is $\frac{1}{k}$-large.*

Proof. In time $O(n^4)$ we compute an optimum top-drawn set F^A for the instance (Theorem 31). This is a feasible UFPP solution (Proposition 16). Let F^* be an optimum UFPP solution. Then F^* can be partitioned into at most $2k$ top-drawn sets (Lemma 32), so $w(F^A) \geq \frac{1}{2k} w(F^*)$. □

5. Summary of the Main Approximation Algorithms

After the preparations in the previous sections we present our main $(7+\epsilon)$-approximation algorithm for UFPP. The algorithm calls the routines for small, medium, and large tasks which were presented in Sections 3 and 4 and outputs the returned set with maximum profit. After that, we show how to obtain a constant-factor approximation algorithm with a much better running time of $O(n^4 \log n)$.

Theorem 35. *For every $\epsilon > 0$ there is a $(7+\epsilon)$-approximation algorithm for the UFPP problem which runs in polynomial time.*

Proof. We assume that $0 < \epsilon \leq \frac{1}{2}$ (otherwise we simply set $\epsilon := \frac{1}{2}$) and we define $\beta := \epsilon$. Given an instance of the UFPP problem we divide the tasks into small/medium tasks and large tasks. A task i is small or medium if $d_i \leq (1-\beta)b(i)$ and large otherwise. Due to Theorem 13 there is a $(3+\epsilon)$-approximation algorithm for the small and medium tasks. Since $\beta \leq \frac{1}{2}$, for each large task $d_i > \frac{1}{2}b(i)$ holds. According to Theorem 34 there is then a 4-approximation algorithm for the large tasks. Returning the best solution of the two then yields a $(7+\epsilon)$-approximation algorithm for the set of all (small, medium, and large) tasks. □

The running time of the above algorithm is dominated by the dynamic program which is needed for handling the medium tasks, see Lemma 11. Unfortunately, the exponent of the running time is quite large. However, in the following theorem we show that already a moderate running time of $O(n^4 \log n)$ suffices to obtain a constant factor approximation. The key is to avoid the expensive dynamic program for the medium tasks. Instead, we divide the instance such that there are only small and large tasks. At the cost of a higher approximation factor, this algorithm has a much better running time.

Theorem 36. *There is a 25.12-approximation algorithm for the UFPP problem with running time $O(n^4 \log n)$.*

Proof. We choose our constants differently than in the previous algorithm. We define $\beta = \epsilon := 5/80$, $k := 9$, $\delta := 1/k$, $\ell := 3$, and $q := 5$. We define a task i to be

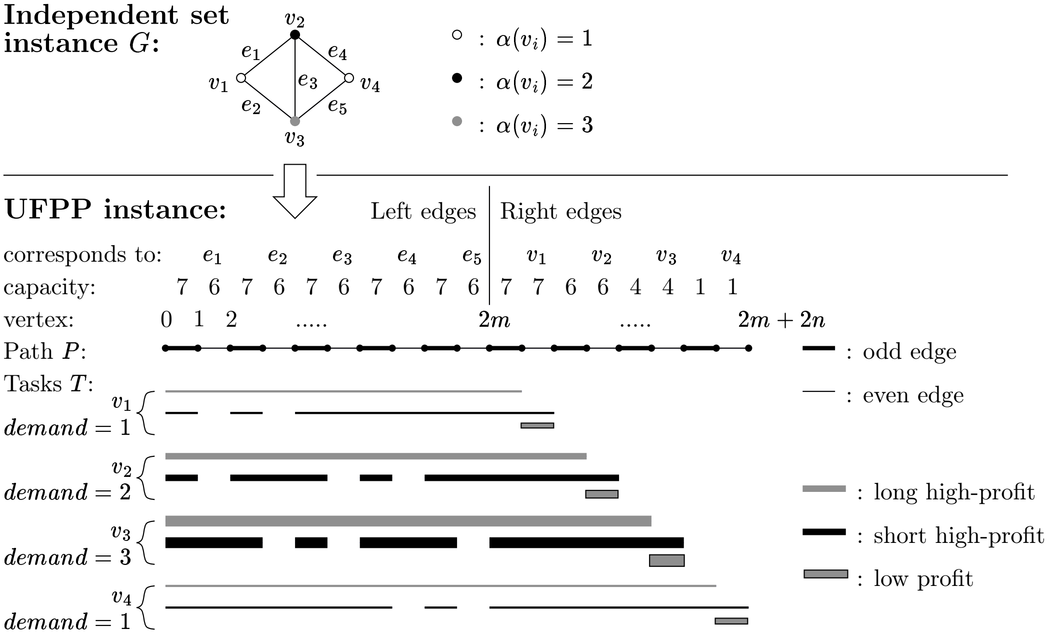


FIGURE 6.1. An example of a graph G with coloring α and the resulting instance UFPP(G). (The demand of a task is indicated by the width of its bar.)

small if $d_i \leq \delta \cdot b(i)$ and *large* otherwise. Due to Lemma 9, for the small tasks there is an α-approximation algorithm with $\alpha \approx 7.12$ with running time $O(n^4 \log n)$. For the large tasks, due to Theorem 34, there is a $2k$-approximation algorithm with a running time of $O(n^4)$. So the algorithm that returns the best solution of the two is an approximation algorithm with a performance ratio of $\alpha + 2k \leq 25.12$, and a running time of $O(n^4 \log n)$. □

6. Strong NP-Hardness

In this section we prove that UFPP is strongly NP-hard for instances with demands in $\{1, 2, 3\}$, using a reduction from Maximum Independent Set in Cubic Graphs. Let G, k be an independent set instance of maximum degree 3: $G \neq K_4$ is a graph with maximum degree 3 and the question is whether G contains an independent set of size at least k. This problem is NP-hard [18] (even for *cubic* graphs, in which all degrees are exactly 3). Let $V(G) = \{v_1, \ldots, v_n\}$, and $E(G) = \{e_1, \ldots, e_m\}$. By Brooks' Theorem (see e.g. [23]), G is 3-colorable since $G \neq K_4$. Such a coloring can be found in polynomial time [23], so for our polynomial transformation we may assume a proper 3-coloring $\alpha : V(G) \to \{1, 2, 3\}$ is given.

We now construct an equivalent instance of UFPP as follows, see Figure 6.1. The path P has $2m + 2n + 1$ vertices, labeled $0, \ldots, 2n + 2m$. For the proofs below it will be useful to distinguish four types of edges (called odd left, odd right, even left and even right): an edge $\{i - 1, i\} \in E(P)$ (with $1 \leq i \leq 2n + 2m$) is a *left edge* if $i \leq 2m$ and a *right edge* otherwise. It is an *odd edge* if i is odd, and an *even edge* otherwise. Even left edges $\{2k-1, 2k\}$ (with $1 \leq k \leq m$) will be associated with edges e_k of G, and even right edges $\{2k-1, 2k\}$ (with $m+1 \leq k \leq n+m$) will be associated with vertices v_{k-m} of G.

For every vertex $v_i \in V(G)$, introduce the following tasks: First, introduce one *long* task with start vertex 0 and end vertex $2m+2i-1$. Next, let $\sigma_1, \ldots, \sigma_d$ be the indices of the edges incident with v_i, in increasing order. Introduce $d+1$ *short* tasks with the following start and end vertex pairs: $(0, 2\sigma_1 - 1), (2\sigma_1, 2\sigma_2 - 1), \ldots, (2\sigma_{d-1}, 2\sigma_d - 1), (2\sigma_d, 2m+2i)$. For all of these tasks for vertex v_i, the demand is equal to $\alpha(v_i)$, and the profit is equal to $\alpha(v_i)n$ times the number of odd edges used by the task. The aforementioned tasks will be called the *high-profit tasks* (for v_i). Finally, for v_i we introduce one *low-profit task* from $2m+2i-1$ to $2m+2i$ with profit 1 and demand $\alpha(v_i)$.

Doing this for all vertices gives the tasks of the UFPP instance. It remains to set the edge capacities of P:

- For odd left edges $e = \{2k-2, 2k-1\} \in E(P)$ for integer $1 \leq k \leq m$, set $u_e = \sum_{v \in V(G)} \alpha(v)$.
- For even left edges $e = \{2k-1, 2k\} \in E(P)$ for integer $1 \leq k \leq m$, set $u_e = (\sum_{v \in V(G)} \alpha(v)) - 1$.
- For right edges $e = \{2k-2, 2k-1\} \in E(P)$ and $e = \{2k-1, 2k\} \in E(P)$ for integer $m+1 \leq k \leq n+m$, set $u_e = \sum_{i=k-m}^{n} \alpha(v_i)$.

This gives the instance $\text{UFPP}(G)$ of UFPP, to which we will refer in the remainder of this section. With T we will denote the set of all tasks of $\text{UFPP}(G)$. With an independent set I of G we associate a solution of $\text{UFPP}(G)$ of the following form.

Definition 37. *A set $F \subseteq T$ of tasks of* $\text{UFPP}(G)$ *is said to be in* standard form *if:*

(1) *For every $i \in \{1, \ldots, n\}$:*
   - *Either F contains the long high-profit task for v_i and the low-profit task for v_i, but no short high-profit tasks for v_i,*
   - *or F contains all short high-profit tasks for v_i, but neither the long high-profit task for v_i nor the low-profit task for v_i.*

(2) *For every $\{v_i, v_j\} \in E(G)$, F does not contain both the long high-profit task for v_i and the long high-profit task for v_j.*

Now we will first show that any task set F that is in standard form is indeed a feasible solution to $\text{UFPP}(G)$. Next we will show that the total profit of a solution F in standard form depends only (linearly) on the number of long high-profit tasks in F. By the second property in the definition above, this corresponds to an independent set in G. Subsequently we will show that any optimal solution for $\text{UFPP}(G)$ is in standard form. Together this will show that the instances are equivalent (when considering them as decision problems with appropriately chosen target objective values).

Proposition 38. *Let $F \subseteq T$ be in standard form. Then F is a feasible solution for* $\text{UFPP}(G)$.

Proof. Since F is in standard form, for every $v_i \in V(G)$, every edge of P is used by at most one task for v_i. Therefore, for odd left edges $e = \{2k-2, 2k-1\} \in E(P)$ (with $1 \leq k \leq m$), the capacity $u_e = \sum_{v \in V(G)} \alpha(v)$ is not exceeded. Similarly, for odd right edges $e = \{2k-2, 2k-1\} \in E(P)$ (with $m+1 \leq k \leq n+m$), the capacity $\sum_{i=k-m}^{n} \alpha(v_i)$ is not exceeded. Now we consider even right edges.

Every edge $e = \{2k-1, 2k\} \in E(P)$ with $m+1 \leq k \leq n+m$ is used by high-profit tasks corresponding to the vertices $v_{k-m+1}, \ldots, v_n$, using a total capacity of $\sum_{i=k-m+1}^{n} \alpha(v_i)$. In addition, this edge is either used by one short high-profit task for v_{k-m}, or by the single low-profit task for v_{k-m}, both with demand $\alpha(v_{k-m})$. This shows that for these edges the capacity bound is satisfied. Finally, consider even left edges $e = \{2k-1, 2k\} \in E(P)$ with $1 \leq k \leq m$. These correspond to edges $e_k \in E(G)$. Let $e_k = \{v_i, v_j\}$. Since F is in standard form, it cannot contain high-profit tasks for both v_i and v_j. Note that no short high-profit tasks for v_i and v_j use the edge $\{2k-1, 2k\}$. Hence the total capacity of e used by F is at most

$$\max\Big\{\Big(\sum_{w\in V(G)} \alpha(w)\Big) - \alpha(v_i), \Big(\sum_{w\in V(G)} \alpha(w)\Big) - \alpha(v_j)\Big\} \leq \Big(\sum_{w\in V(G)} \alpha(w)\Big) - 1 = u_e.$$

□

Proposition 39. *Let $F \subseteq T$ be a solution in standard form for* UFPP(G), *which contains x long high-profit tasks. Then $w(F) = x + \sum_{i=1}^{n} \alpha(v_i)n(m+i)$.*

Proof. Recall that for high-profit tasks for a vertex v_i, the profit equals $\alpha(v_i)n$ times the number of odd edges used by the task. The long-high profit task for v_i (from 0 to $2m+2i-1$) uses all odd edges $\{2k-2, 2k-1\}$ of P with $1 \leq k \leq m+i$. Hence if selected, it contributes $\alpha(v_i)n(m+i)$ to the total profit. Together, the low-profit tasks for v_i use exactly the same set of *odd* edges of P. So either way, the high-profit tasks for v_i contribute $\alpha(v_i)n(m+i)$ to the total profit. The single low-profit task for v_i, with a profit of 1, is selected if and only if the long high-profit task is selected. This yields the stated total profit. □

These two propositions show that independent sets of G correspond to solutions of UFPP(G).

Lemma 40. *If G has an independent set I with $|I| = x$, then* UFPP(G) *admits a solution of profit $x + \sum_{i=1}^{n} \alpha(v_i)n(m+i)$.*

Proof. Construct F by selecting the long high-profit task and the low-profit task for v_i if $v_i \in I$, and all short high-profit tasks for v_i otherwise. Since I is an independent set, this set F is in standard form. So by Proposition 38, it is a feasible solution to UFPP(G). Proposition 39 now gives the total profit. □

Now we will show that any optimal solution to UFPP(G) is in standard form, which will yield the converse of Lemma 40. We say that a solution F *fully uses the capacity* of an edge $e \in E(P)$ if $\sum_{i\in F\cap T_e} d_i = u_e$.

Proposition 41. *Let F be an optimal solution for* UFPP(G). *For every* odd *edge $e \in E(P)$, F uses the full capacity of e.*

Proof. Recall that for every high-profit task with a demand of d, the profit equals dnx, where x is the number of odd edges used by the task. Hence the total profit of high-profit tasks in a feasible solution F is bounded by n times the sum of capacities of all such edges. This capacity sum is

$$m\sum_{i=1}^{n} \alpha(v_i) + \sum_{k=1}^{n}\sum_{i=k}^{n} \alpha(v_i) = \sum_{i=1}^{n} \alpha(v_i)(m+i).$$

Suppose that for at least one odd edge the full capacity is not used by F. The low-profit tasks in F can in total only yield a profit of n, so then the total profit is at most

$$n\left(\sum_{i=1}^{n}\alpha(v_i)(m+i)-1\right)+n=n\sum_{i=1}^{n}\alpha(v_i)(m+i).$$

Since there exists a feasible solution for UFPP(G) with strictly higher profit (Lemma 40), this shows that F is not optimal. □

Lemma 42. *Every optimal solution F for* UFPP(G) *is in standard form.*

Proof. First we will prove by (backward) induction that Property 1 of Definition 37 holds for an optimal solution F. The induction hypothesis will be that the following claim holds for a given integer x. Note that for $x = 0$, this claim (almost) yields Property 1 of Definition 37.

Claim 1: An optimal solution F contains either

- the long high-profit task for v_i and no short high-profit tasks for v_i with end vertex $t \geq 2x$, or
- all short high-profit tasks for v_i with end vertex $t \geq 2x$ but not the long high-profit task for v_i.

For the induction base ($x = n+m$), consider the odd edge $e = \{2m+2n-2, 2m+2n-1\}$. Its capacity is $\alpha(v_n)$, and the only tasks using e are the long (high-profit) task for v_n, and one short (high-profit) task, both with demand $\alpha(v_n)$. Since F uses the full capacity of e (Proposition 41), F contains either this long task or the short task, which proves the statement for $x = n + m$.

For the induction step, we prove Claim 1 for $x = k$, assuming that it holds for $x = k + 1$ (the induction hypothesis).

First consider the case that $m + 1 \leq k \leq n + m - 1$. Consider the odd right edge $e = \{2k - 2, 2k - 1\}$. Let $e' = \{2k, 2k + 1\}$. The capacity difference is $u_e - u_{e'} = \alpha(v_{k-m})$. All tasks using e' use e as well. The only tasks using e but not e' are one long and one short task for v_{k-m}, both with demand $\alpha(v_{k-m})$. So F must contain exactly one of these (Proposition 41), which proves Claim 1 for $k \geq m + 1$.

In the case that $k = m$, Claim 1 for $x = k$ follows immediately from the induction hypothesis since for all *short* high-profit tasks with end vertex $t \geq 2m$, it in fact holds that $t \geq 2m + 2$.

Finally, consider the case that $0 \leq k \leq m - 1$. To prove Claim 1 it suffices to show that for any short-high profit task with end vertex $2k \leq t \leq 2k + 1$, that corresponds to a vertex v_i, it is included in F if and only if F contains at least one other short high-profit task for v_i. In fact, since there are no short high-profit tasks with end vertex $t = 2k$, we only need to consider tasks with $t = 2k + 1$. Therefore, consider the odd left edge $e = \{2k, 2k + 1\}$. Let $e' = \{2k + 2, 2k + 3\}$. To prove the statement, we only need to consider tasks that use e but not e' and tasks that use e' but not e. Let $e_{k+1} = \{v_i, v_j\}$ (this is the edge of G associated with the even edge between e and e'). The only tasks that use e' but not e are one short high-profit task for v_i of demand $\alpha(v_i)$, and one short high profit task for v_j of demand $\alpha(v_j)$. Similarly, there are two such tasks that use e but not e', of demand $\alpha(v_i)$ and $\alpha(v_j)$. Since α is a proper coloring of the vertices of G, $\alpha(v_i) \neq \alpha(v_j)$.

Since F fully uses the capacity of both e and e', and $u_e = u_{e'}$, F includes a short task for v_i that uses e if and only if it includes a short task for v_i that uses e', and an analog statement holds for v_j. Since there are no tasks with end vertex $t = 2k$, this proves Claim 1 for $0 \leq k \leq m - 1$.

For all choices of k we have now proved the induction step, so by induction, Claim 1 holds for $x = 0$. We conclude that in an optimal solution F, for every $v \in V(G)$, either only the long high-profit task is selected, or all short high-profit tasks are selected and not the long one. One can always add the single low-profit task for v_i to F, whenever no short high-profit tasks for v_i have been selected in F, so this proves Property 1 of Definition 37 for F.

Now we will prove that Property 2 of Definition 37 also holds for the optimal solution F. We will prove that the vertices of G for which F contains a long high-profit task form an independent set in G. More precisely, for any edge $\{v_i, v_j\} \in E(G)$, F does not contain both the long high-profit task for v_i and the long high-profit task for v_j. Let $e_k = \{v_i, v_j\}$ (with $k \in \{1, \ldots, m\}$). According to our construction this edge is associated with an even left edge $e = \{2k-1, 2k\} \in E(P)$ of UFPP(G). The capacity of e is one less than that of its adjacent odd edge $e' = \{2k-2, 2k-1\}$, and yet the capacity of e' is fully used by F (Proposition 41). The only tasks using e' but not e are a short task for v_i, and a short task for v_j. So for at least one of v_i and v_j, F includes this short task. Since we already showed in the first part of the proof that F cannot contain both a long high-profit task and a short high-profit task for the same vertex v_k, it follows that for at least one of v_i and v_j, F does not contain the long task. Hence Property 2 of Definition 37 also holds for F, and thus F is in standard form. □

Summarizing, any optimal solution to UFPP(G) corresponds to an independent set of G:

Lemma 43. *If* UFPP(G) *admits a solution of profit at least* $x + \sum_{i=1}^{n} \alpha(v_i)n(m+i)$, *then* G *has an independent set of size at least* x.

Proof. Consider an optimal solution F, so $w(F) \geq x + \sum_{i=1}^{n} \alpha(v_i)n(m+i)$. By Lemma 42, F is in standard form. Let I be the set of vertices v_i of G for which F contains the long high-profit task. Since F is in standard form, I is an independent set of G. Proposition 39 shows that $w(F) = |I| + \sum_{i=1}^{n} \alpha(v_i)n(m+i)$. It follows that $|I| \geq x$, so I is the desired independent set. □

Our construction used only polynomially bounded numbers (profits, demands and capacities), so even if the numbers are encoded in unary, this is a polynomial transformation. For this it is also essential that a 3-coloring α of G can be found in polynomial time [23]. Lemmas 40 and 43 show that G has an independent set of size at least x if and only if P, F admits a solution of profit at least $x + \sum_{i=1}^{n} \alpha(v_i)n(m+i)$. Since the problem of finding a maximum independent set is NP-hard when restricted to graphs of maximum degree 3 [18], this proves that UFPP is strongly NP-hard. The problem obviously lies in NP, which yields:

Theorem 44. *UFPP is strongly NP-complete when restricted to instances with demands in* $\{1, 2, 3\}$.

In fact, with a small addition we can show that the problem remains NP-complete when all edge capacities are equal. Let u_m be the maximum capacity used in the instance UFPP(G) constructed above, and let $X = n + \sum_{i=1}^{n} \alpha(v_i)n(m+i)$ be an upper bound on the profit of any solution (Lemma 43). Note that X is bounded by a polynomial in n. For every edge $e = \{k, k+1\}$ of P with $u_e < u_m$, we can increase the capacity to u_m, and introduce $u_m - u_e$ *dummy tasks*: tasks from k to $k+1$ with demand 1 and profit X. For a polynomial transformation we must ensure that the number of dummy tasks is polynomially bounded. Given an instance with demands in $\{1, 2, 3\}$, a maximum capacity of $3n$ suffices, so all capacities $u_e > 3n$ can be scaled down to $3n$. Hence, on each edge at most $O(n)$ dummy tasks are introduced. Since the number of edges is bounded by $O(n)$, at most $O(n^2)$ dummy tasks with profit X and demand 1 need to be added. In an optimal solution, clearly all of the dummy tasks are selected, so this yields an equivalent instance. Therefore the previous theorem can be strengthened to:

Theorem 45. *UFPP is strongly NP-complete when restricted to instances with demands in* $\{1, 2, 3\}$*, where all edges have equal capacities.*

7. Conclusion and Discussion

In this paper we proved strong NP-hardness for UFPP with uniform capacities (RAP), even if the input is restricted to demands in $\{1, 2, 3\}$, and gave the first constant factor polynomial time approximation algorithm for UFPP. The main remaining open question is whether there exists a PTAS for RAP or UFPP, or whether the problems are APX-hard. Due to our hardness result it would be interesting to know whether the special case of demands in $\{1, 2\}$ is also NP-hard, or admits a polynomial time algorithm. Note that if the demands are uniform, the problem can be solved in polynomial time with a small extension of the algorithm by Arkin and Silverberg [2]. Since finding a PTAS for UFPP seems very challenging, it would be interesting to see whether the factor of $7 + \epsilon$ can be improved.

In this paper we did not consider other graph classes than the path. Can our techniques be used for other graph classes? In particular, extending the algorithm to trees is not straightforward, since Lemma 3 does not apply in this case.

Finally, it may be interesting to study whether our new techniques for finding maximum non-overlapping sets of rectangles can be applied to other rectangle packing problems.

References

[1] P. K. Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *Computational Geometry*, 11:209 – 218, 1998.

[2] E. M. Arkin and E. B. Silverberg. Scheduling jobs with fixed start and end times. *Discrete Appl. Math.*, 18:1–8, November 1987.

[3] N. Bansal, A. Chakrabarti, A. Epstein, and B. Schieber. A quasi-PTAS for unsplittable flow on line graphs. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, STOC '06, pages 721–729, New York, NY, USA, 2006. ACM.

[4] N. Bansal, Z. Friggstad, R. Khandekar, and M. Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. In *SODA '09: Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 702–709, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[5] Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph (Seffi) Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, STOC '00, pages 735–744, New York, NY, USA, 2000. ACM.

[6] R. Bar-yehuda, M. Beder, and Y. Cohen. Approximation algorithms for bandwidth and storage allocation, 2005.

[7] V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller. A constant-approximate feasibility test for multiprocessor real-time scheduling. In Dan Halperin and Kurt Mehlhorn, editors, *Algorithms - ESA 2008*, volume 5193 of *Lecture Notes in Computer Science*, pages 210–221. Springer Berlin / Heidelberg, 2008.

[8] G. Calinescu, A. Chakrabarti, H. J. Karloff, and Y. Rabani. Improved approximation algorithms for resource allocation. In *Proceedings of the 9th International IPCO Conference on Integer Programming and Combinatorial Optimization*, pages 401–414, London, UK, 2002. Springer-Verlag.

[9] A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar. Approximation algorithms for the unsplittable flow problem. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*, APPROX '02, pages 51–66, London, UK, 2002. Springer-Verlag.

[10] P. Chalermsook and J. Chuzhoy. Maximum independent set of rectangles. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 892–901, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[11] C. Chekuri, M. Mydlarz, and F. Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Trans. Algorithms*, 3:27, 2007.

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

[13] A. Darmann, U. Pferschy, and J. Schauer. Resource allocation with time intervals. *Theor. Comput. Sci.*, 411:4217–4234, November 2010.

[14] C. W. Duin and E. Van Sluis. On the complexity of adjacent resource scheduling. *J. of Scheduling*, 9:49–62, February 2006.

[15] F. Eisenbrand and T. Rothvoß. A PTAS for static priority real-time scheduling with resource augmentation. In *Automata, Languages and Programming*, volume 5125 of *Lecture Notes in Computer Science*, pages 246–257. Springer Berlin / Heidelberg, 2008.

[16] T. Erlebach, K. Jansen, and E. Seidel. Polynomial-time approximation schemes for geometric graphs. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, SODA '01, pages 671–679, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

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

[18] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoret. Comput. Sci.*, 1(3):237–267, 1976.

[19] S. Heinz and J. Schulz. Explanation algorithms for the cumulative constraint: An experimental study. Technical report, TU Berlin, 2011.

[20] S. Khanna, S. Muthukrishnan, and M. Paterson. On approximating rectangle tiling and packing. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, SODA '98, pages 384–393, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[21] T. W. Lam and K. K. To. Trade-offs between speed and processor in hard-deadline scheduling. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '99, pages 623–632, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.

[22] S. Leonardi, A. Marchetti-Spaccamela, and A. Vitaletti. Approximation algorithms for bandwidth and storage allocation problems under real time constraints. In *Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science*, FST TCS 2000, pages 409–420, London, UK, 2000. Springer-Verlag.

[23] L. Lovász. Three short proofs in graph theory. *J. Combinatorial Theory Ser. B*, 19(3):269–271, 1975.

[24] F. Nielsen. Fast stabbing of boxes in high dimensions. *Theoretical Computer Science*, 246(1-2):53 – 72, 2000.

[25] J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. In *Operations Research*, pages 377–387, 1988.

[26] C. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32:163–200, 2008. 10.1007/s00453-001-0068-9.

[27] C. A. Phillips, R. N. Uma, and J. Wein. Off-line admission control for general scheduling problems. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, SODA '00, pages 879–888, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[28] V. V. Vazirani. *Approximation algorithms*. Springer-Verlag, Berlin, 2001.