2011 IEEE 52nd Annual Symposium on Foundations of Computer Science

A Constant Factor Approximation Algorithm for Unsplittable Flow on Paths

Paul Bonsma
Computer Science Department
Humboldt Universität zu Berlin
Berlin, Germany
Email: bonsma@informatik.hu-berlin.de

Jens Schulz and Andreas Wiese
Institute of Mathematics
Technische Universität Berlin
Berlin, Germany
Email: {jschulz,wiese}@math.tu-berlin.de

Abstract— In this paper, we present a constant-factor approximation algorithm for the unsplittable flow problem on a path. This improves on the previous best known approximation factor of $O(\log n)$. The approximation ratio of our algorithm is $7+\epsilon$ for any $\epsilon > 0$. In the unsplittable flow problem on a path, we are given a capacitated path P and n tasks, each task having a demand, a profit, and start and end vertices. The goal is to compute a maximum profit set of tasks, such that for each edge e of P, the total demand of selected tasks that use e does not exceed the capacity of e. This is a well-studied problem that occurs naturally in various settings, and therefore it has been studied under alternative names, such as resource allocation, bandwidth allocation, resource constrained scheduling, temporal knapsack and interval packing. Polynomial time constant factor approximation algorithms for the problem were previously known only under the no-bottleneck assumption (in which the maximum task demand must be no greater than the minimum edge capacity).

We introduce several novel algorithmic techniques, which might be of independent interest: a framework which reduces the problem to instances with a bounded range of capacities, and a new geometrically inspired dynamic program which solves a special case of the maximum weight independent set of rectangles problem to optimality. In addition, we show that the problem is strongly NP-hard even if all edge capacities are equal and all demands are either 1, 2, or 3.

Keywords-unsplittable flow; resource allocation; maximum weight independent set; constant factor approximation; strong NP-hardness

1. Introduction

In the Unsplittable Flow Problem on a Path (UFPP), we are given a path $P = (V, E)$ with an integral 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* $w_i \in \mathbb{N}$. A task i *uses an edge* $e \in E$ if e lies on the path from s_i to t_i. 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 sum of the demands of all tasks in F that use this edge does not exceed its capacity.

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 been studied under the names *bandwidth allocation* [9], [20], [29], *admission control* [32], *interval packing* [21], *temporal knapsack* [10], *multicommodity demand flow* [19], *unsplittable flow problem* [6], [7], [16], [18], *scheduling with fixed start and end times* [4], and *resource allocation* [8], [13], [22], [23], [32]. 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.

UFPP is easily seen to be (weakly) NP-hard, since it contains the Knapsack problem as a special case (in case the path is just a single edge). In addition, Darmann et al. [22] show that the special case where all profits and all capacities are uniform is also weakly NP-hard. Chrobak et al. [21] strengthen this result by showing strong NP-hardness in this case. In addition, they show that the case where the profits equal the demands is strongly NP-hard [21]. These results show that the problem admits no FPTAS unless $P = NP$. On the other hand, the special case of a single edge (Knapsack) admits an FPTAS. When the number of edges is bounded by a constant, UFPP admits a PTAS since it is a special case of Multi-Dimensional Knapsack [25].

Most of the research on UFPP has focused on two restricted cases: firstly, the special case in which all capacities are equal has been well-studied, which is also known as the *Resource Allocation Problem (RAP)* [8], [13], [22], [23], [32]. A more general special case of UFPP is given by the *No-Bottleneck Assumption (NBA)*: in that case it is required that $\max_i d_i \leq \min_e u_e$ (this holds in particular for RAP).

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 author was supported by the DFG project "Algorithm Engineering for Real-time Scheduling and Routing" within DFG focus program 1307.

0272-5428/11 $26.00 © 2011 IEEE
DOI 10.1109/FOCS.2011.10

We will denote this restriction of the problem by *UFPP-NBA*. For UFPP-NBA, a $(2+\epsilon)$-approximation algorithm is known [19], which matches the earlier best approximation ratio for RAP [13].

Both RAP and UFPP-NBA have been generalized in various ways: in a scheduling context, constant factor approximation algorithms for RAP have been given for the case that the start and end times of tasks are not fixed, and can be chosen among several options [8], [32]. In a network flow context, there is an extensive literature on the generalization of UFPP-NBA to more general graphs, in which a single path has to be chosen for each selected terminal pair s_i, t_i: For instance, Baveja and Srinivasan [11] provide an $O(\sqrt{|E|})$-approximation algorithm for UFPP-NBA generalized to all graphs, improving on various earlier results. A simpler combinatorial algorithm with the same guarantee is given by Azar and Regev [5]. Chakrabarti et al. [15], [16] also give an approximation algorithm for all graphs, and the first constant factor approximation for UFPP-NBA generalized to cycles. Chekuri et al. [19] obtain a 48-approximation for the generalization of UFPP-NBA to trees. In addition, many hardness results are known for UFPP generalized to various graphs: for general graphs, it is hard to approximate within a factor of $\Omega(|E|^{1-\epsilon})$ unless $P = NP$ [5], and for depth-3 trees the problem is APX-hard [27]. Hardness-of-approximation results are known even for the special case with unit demands and capacities (the *Edge Disjoint Path Problem*); see [2], [3]. When viewing UFPP as a packing problem, the corresponding *covering problem* has also been studied. In that case, tasks have *costs* instead of profits, and the objective is to find a minimum cost set of tasks F, such that for each edge, the sum of the demands of all tasks in F that use this edge is *at least* its capacity. Recently, Chakaravarthy et al. [14] designed a primal-dual 4-approximation algorithm for this problem.

The two main algorithmic techniques that have been used in previous results on UFPP are dynamic programming and rounding of solutions to the linear programming relaxation of the problem. These techniques work well when the NBA holds. However, there are several important obstacles that prevent these techniques to be generalized to the general case of UFPP. For example, Chakrabarti et al. [16] show that under the NBA the natural LP-relaxation of UFPP has a constant integrality gap. However, without this assumption the integrality gap can be as large as $\Omega(n)$ [16]. Moreover, the NBA implies that if all tasks are δ-large (see the definition below), then in any solution there can be at most $2\lfloor 1/\delta^2 \rfloor$ tasks which use each edge; see [16]. This property is useful for setting up a dynamic program. Without the NBA this is no longer possible.

Despite these obstacles, there are a few breakthrough results for (general) UFPP: The best known polynomial time algorithm achieves an approximation factor of $O(\log n)$ [7], thus beating the integrality gap of the natural LP-relaxation. Chekuri et al. [18] give a linear programming relaxation for UFPP with integrality gap $O(\log^2 n)$ [18]. Finally, 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 $2^{\mathrm{polylog}\, n}$ [6].

Thus, even though UFPP and its variants have been intensively studied, in terms of polynomial time approximation algorithms for the general case, much remains unknown: the best known algorithm is an $O(\log n)$-approximation algorithm [7], and the best negative result is strong NP-hardness [21].

1.1. Our Contribution

We present the first polynomial time constant-factor approximation algorithm for the general case of UFPP. The algorithm has an approximation ratio of $7+\epsilon$, for every $\epsilon > 0$. This answers an open question from [7], [18].

To obtain this result we introduce several new algorithmic techniques, which are interesting in their own right. We develop a useful viewpoint which allows us to reduce the problem to a special case of the maximum weight independent set of rectangles problem. In addition, we design a framework which reduces the problem to solving instances where essentially the edge capacities are within a constant factor of each other. The techniques can be applied and combined in various ways. For instance, for practical purposes, we also sketch how our results can be used to obtain a constant factor approximation algorithm with a reasonable running time of only $O(n^4 \log n)$. We now go into more detail about these results, the new techniques we introduce, and give an outline of the paper.

For a task i, denote by $b(i)$ the minimum capacity among all edges used by task i. For $0 < \delta \leq 1$, we call a task i *δ-small* if $d_i < \delta b(i)$ holds, and *δ-large* otherwise. Similar to many previous papers, for our main algorithm we partition the tasks into 'small' and 'large' tasks, depending on δ. For the small tasks our main result is as follows: Given $\gamma > 0$ and $\epsilon > 0$, if each task is $(1-\gamma)$-small, then we present a $(3+\epsilon)$-approximation algorithm in Section 3. We remark that a similar result is given by Chekuri et al. [18], who give an $O(\log(1/\gamma)/\gamma^3)$-approximation algorithm if each task is $(1-\gamma)$-small. Their result also applies to trees. To prove our $(3+\epsilon)$-approximation, we introduce a novel framework in which the tasks are first grouped into smaller sets, according to their $b(i)$ value, such that the techniques for the NBA case can be applied. So the resulting sets can be solved via relatively standard dynamic programming, LP-rounding and network flow techniques. (This is similar to e.g. [13], [19].) Solutions to these smaller sets leave a small amount of the capacity of each edge unused. In our framework we recombine these solutions into a feasible solution for all tasks.

For the remaining large tasks, we present the following main result: for any integer $k \geq 2$, if all tasks are $\frac{1}{k}$-large, we give a $2k$-approximation algorithm in Section 4. This is based on a geometric viewpoint of the problem: we represent UFPP instances by drawing a curve in the plane determined by the edge 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 its 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 every task is $\frac{1}{k}$-large, this solution yields a $2k$-approximative solution for UFPP. With our dynamic program we contribute towards the well-studied problem of finding a maximum weight independent sets of (arbitrary) rectangles (MWISR). For the unweighted case of this problem, a randomized $O(\log \log n)$-approximation is known [17]. For the weighted case, there are several $O(\log n)$-approximation algorithms [1], [28], [31]. The algorithm by Erlebach et al. [24] gives a PTAS for the case that the ratio between height and width of rectangles is bounded (note that this does not apply in the special case that we need here for approximating UFPP). Our new dynamic programming technique might be useful for further research on this problem.

For our main algorithm, we partition the tasks into $\frac{1}{2}$-small tasks and $\frac{1}{2}$-large tasks. For the first group, we apply the aforementioned $(3+\epsilon)$-approximation algorithm. For the second group, our second algorithm gives a 4-approximation. Returning the best solution of the two yields the $(7+\epsilon)$-approximation algorithm.

In addition, we give an alternative proof of the strong NP-hardness of UFPP, which shows that a different restriction remains strongly NP-hard. In the existing NP-hardness proofs [21], [22], arbitrarily large demands are used in the reductions. In Section 5, 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 (RAP). Note that in contrast to our hardness result, it is known that in the slightly more restricted case where the capacities *and demands* are uniform, the problem admits a polynomial time algorithm [4].

In Section 6 we discuss further extensions, variants and consequences, such as the $O(n^4 \log n)$ time constant factor approximation algorithm, and an algorithm which computes a $(2+\epsilon)$-approximative solution in the setting of *resource augmentation*. To be precise, the latter algorithm returns a $(2+\epsilon)$-approximative solution which is feasible if we increase the capacity of each edge by a factor of $1+\beta$ (for arbitrarily small $\epsilon > 0$ and $\beta > 0$). In addition, we discuss how our results carry over to the generalization from a path to a cycle network. We begin in Section 2 by introducing notation and terminology. Due to space constraints, for full proofs of our results we refer to our technical report [12].

2. PRELIMINARIES

We assume that the vertices of the path $P = (V, E)$ are numbered $V = \{0, \ldots, m\}$, and $E = \{\{i, i+1\} \mid 0 \leq i \leq m-1\}$. We assume that the tasks are numbered $T = \{1, \ldots, n\}$. 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. For a set of tasks F we define its profit by $w(F) := \sum_{i \in F} w_i$. Our objective is to find a set of tasks F with maximum profit such that $\sum_{i \in F \cap T_e} d_i \leq u_e$ for each edge e. 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 $\ell(i) := b(i) - d_i$. 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 $x+1$. 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. Therefore, we will henceforth assume that $m < 2n$. 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.

We define an α-approximation algorithm for a maximization problem to be a polynomial time 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. 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. The following simple fact shows how we can combine our approximation algorithms for different task subsets into one algorithm for all tasks.

Fact 1: Consider a UFPP instance with task set T, and a partition $\{T_1, T_2\}$ of T. If for $i = 1, 2$, there exists an α_i-approximation algorithm for the instance restricted to the tasks in T_i, then there exists an $(\alpha_1 + \alpha_2)$-approximation algorithm for the entire instance.

3. SMALL TASKS

In this section we present a $(3+\epsilon)$-approximation algorithm for any set of tasks which are $(1-\gamma)$-small (for arbitrarily small $\epsilon > 0$ and $\gamma > 0$). In our main algorithm

(for a general set of tasks) we will invoke this algorithm as a subroutine for all tasks that are $\frac{1}{2}$-small.

Our strategy here is to define groups of tasks such that the bottleneck capacities of all tasks in one group are within a certain range. This allows us to compute a feasible solution for each group, whose profit is at most a factor $3+\epsilon'$ smaller than the profit of an optimal solution for the group. In addition, each computed solution leaves a certain amount of capacity of every edge unused. Therefore, we can combine the solutions for a selection of groups into a feasible solution for the entire instance, in a way that yields a $(3+\epsilon)$-approximation (with an appropriate choice of ϵ' for the given ϵ).

3.1. Framework

We define the framework sketched above. We group the tasks into sets according to their bottleneck capacities. Let $\ell \in \mathbb{N}$ be a constant. We define $F^{k,\ell} := \left\{i \in T \mid 2^k \leq b(i) < 2^{k+\ell}\right\}$ for each integer k. Note that this includes negative values for k, and that at most $\ell \cdot n$ sets are non-empty (only those will be relevant later). In the sequel, we will present an algorithm which computes feasible solutions $ALG\left(F^{k,\ell}\right) \subseteq F^{k,\ell}$. These solutions will satisfy the following properties.

Definition 2: Consider a set $F^{k,\ell}$ and let $\alpha, \beta > 0$. 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 $T_e \cap F^{k,\ell} \neq \emptyset$. (Hence it is a feasible solution.)

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 that turns an (α,β)-approximation algorithm for each set $F^{k,\ell}$ into a $\left(\frac{\ell+q}{\ell} \cdot \alpha\right)$-approximation algorithm for all given tasks, where q and β are chosen such that $\beta = 2^{1-q}$.

Lemma 3 (Framework): Let $\ell \in \mathbb{N}$ and $q \in \mathbb{N}$ be constants and let $\beta = 2^{1-q}$. Let the sets $F^{k,\ell}$ be defined as stated above for an instance of UFPP. 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 that yields Lemma 3. Assume that we are given an (α,β)-approximation algorithm that computes solutions $ALG\left(F^{k,\ell}\right) \subseteq F^{k,\ell}$. 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 we will show further that the indices k for the sets $ALG\left(F^{k,\ell}\right)$ that we want to combine can be chosen such that the resulting set is an $\left(\frac{\ell+q}{\ell} \cdot \alpha\right)$-approximation. 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 $ALG(c) := \bigcup_{k\in\eta(c)} ALG(F^{k,\ell})$. We output the set with maximum profit among all sets $ALG(c)$, which is denoted by $ALG(c^*)$. In Lemma 6 we will prove that the resulting set is an $(\frac{\ell+q}{\ell} \cdot \alpha)$-approximation. First, in Lemma 5 we will prove that each set $ALG(c)$ is feasible (using that $\beta \geq 2^{1-q}$). This requires the following property, which follows from the path structure.

Proposition 4: Let F be a feasible UFPP solution such that $b(i) < c$ for all $i \in F$. Then for every edge e, $\sum_{i\in F\cap T_e} d_i < 2c$.

Lemma 5: For each $c \in \{0,...,\ell+q-1\}$ the set $ALG(c)$ is feasible.

Proof sketch: Consider a set $ALG(F^{k,\ell})$. By definition of (α,β)-approximative solutions, $ALG(F^{k,\ell})$ leaves $\beta \cdot 2^k \geq 2^{k+1-q}$ units of the capacity of every used edge free. Observe that this is at least twice the maximum bottleneck capacity of tasks in $F^{k-(\ell+q),\ell}$. Therefore, by Proposition 4, the set $ALG(F^{k,\ell}) \cup ALG(F^{k-(\ell+q),\ell})$ is feasible. In fact, it again leaves a fraction of the capacities free, which makes it possible to continue this argument for further sets $ALG(F^{k-i(\ell+q)})$ with $i \geq 2$, and prove that their union is feasible. This way it can be shown that for each offset c, the set $ALG(c)$ is feasible. ■

Lemma 6: Let F^* denote an optimal solution of the given instance of UFPP. For the offset c^* it holds that $w(ALG(c^*)) \geq \frac{\ell}{\ell+q} \cdot \frac{1}{\alpha} \cdot w(F^*)$.

Proof: Every task is included in ℓ different sets $F^{k,\ell}$. Using this fact, we calculate that

$$\sum_{c=0}^{\ell+q-1} w(ALG(c)) \geq \sum_{c=0}^{\ell+q-1} \sum_{k\in\eta(c)} \frac{1}{\alpha} w\left(OPT\left(F^{k,\ell}\right)\right) =$$

$$\frac{1}{\alpha} \sum_{k\in\mathbb{Z}} w\left(OPT\left(F^{k,\ell}\right)\right) \geq \frac{1}{\alpha} \sum_{k\in\mathbb{Z}} w\left(F^* \cap F^{k,\ell}\right) = \frac{\ell}{\alpha} w(F^*).$$

So there must be a value c such that $w(ALG(c)) \geq \frac{\ell}{\ell+q} \cdot \frac{1}{\alpha} \cdot w(F^*)$. In particular, this holds for c^*. ■

Now we can prove Lemma 3, which completes our framework.

Proof of Lemma 3: Lemma 5 shows that $ALG(c^*)$ is feasible and Lemma 6 shows 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 $O\left(m \cdot p(n)\right)$. ■

3.2. An Approximation Algorithm for Small Tasks

Now that we have developed the framework to translate (α,β)-approximation algorithms for the sets $F^{k,\ell}$ into an

approximation algorithm for the entire instance (Lemma 3), it remains to present such an (α, β)-approximation algorithm. In this section, we will prove the following lemma, and combine it with the framework to obtain a $(3+\epsilon)$-approximation algorithm for UFPP in case all tasks are $(1-\gamma)$-small. To get some intuition, the reader can think of β being equal to $\min\{\epsilon, \gamma/2\}$.

Lemma 7: Let $\epsilon > 0$, $\beta > 0$, and $\ell \in \mathbb{N}$ be constants, and assume we are given an instance of UFPP in which all tasks are $(1-2\beta)$-small. Then for each set $F^{k,\ell}$, there is a $(2+\frac{1+\epsilon}{1-\beta}, \beta)$-approximation algorithm.

In order to prove Lemma 7, we choose a $\delta > 0$ and further partition the tasks into δ-small tasks and δ-large tasks (assuming that $\delta < 1-2\beta$). For the δ-small tasks, we can use linear programming techniques and a result by Chekuri et al. [19]. UFPP with task set T can be formulated in a straightforward way as an integer linear program (IP):

$$\max \sum_{i \in T} w_i \cdot x_i$$
$$\sum_{i \in T_e} x_i \cdot d_i \leq u_e \qquad \forall e$$
$$x_i \in \{0,1\} \qquad \forall\, i$$

The LP relaxation is obtained by replacing the constraint $x_i \in \{0,1\}$ by $0 \leq x_i \leq 1$. Chekuri et al. [19] give an algorithm for instances of UFPP-NBA with δ-small tasks, which returns a UFPP solution that is at most a factor $f(\delta)$ worse than the optimum of the LP relaxation. Here $f(\delta)$ is a function for which the limit is 1 as δ approaches zero.

This result can be used for sets $F^{k,\ell}$: By definition, tasks in $F^{k,\ell}$ use only edges e with $u_e \geq 2^k$. Call these *relevant edges*. It is therefore possible to choose the value δ small enough to ensure that the NBA holds, when considering only the relevant edges and δ-small tasks in $F^{k,\ell}$. Furthermore, modifying the capacities by choosing $u'_e = u_e - \beta \cdot 2^k$ decreases the capacities of relevant edges at most by a *factor* $(1-\beta)$. Therefore the optimal value of the LP relaxation also becomes at most a factor $(1-\beta)$ smaller. A detailed analysis based on these observations yields:

Lemma 8: For every combination of constants $\epsilon > 0$, $0 < \beta < 1$, and $\ell \in \mathbb{N}$, there exists a $\delta > 0$ such that if all tasks are δ-small, then for each set $F^{k,\ell}$ there is a $(\frac{1+\epsilon}{1-\beta}, \beta)$-approximation algorithm.

In case that $\delta < 1-2\beta$ holds for the above δ, the remaining task is to find an algorithm for tasks in $F^{k,\ell}$ that are both δ-large and $(1-2\beta)$-small. When restricting the instance to δ-large tasks in $F^{k,\ell}$, the essential property is that for any edge e and any feasible solution F, there are at most $O\left(\frac{2^\ell}{\delta}\right)$ tasks in F that use e. This property allows for a straightforward dynamic program to be used to compute an optimal solution (see e.g. [16]). This can be turned into a $(2,\beta)$-approximate solution: since tasks are $(1-2\beta)$-small, it can be shown that in polynomial time, any solution F can be partitioned into two sets which are both feasible for the modified capacities. This yields the following lemma.

Lemma 9: Let $\beta > 0$, $\delta > 0$ and $\ell \in \mathbb{N}$ be constants with $\delta < 1-2\beta$, and assume we are given an instance of UFPP in which all tasks are both $(1-2\beta)$-small and δ-large. Then, for each set $F^{k,\ell}$ there is a $(2,\beta)$-approximation algorithm.

Combining Lemmas 8 and 9 now yields Lemma 7:

Proof of Lemma 7: Given ϵ, β and ℓ, Lemma 8 shows that there exists a $\delta > 0$ such that for the δ-small tasks an $(\frac{1+\epsilon}{1-\beta}, \beta)$-approximation algorithm exists for each $F^{k,\ell}$. The remaining tasks are both δ-large and $(1-2\beta)$-small. If $\delta \geq 1-2\beta$, we are done. Otherwise, Lemma 9 shows that we have a $(2,\beta)$-approximation algorithm for these tasks, for each set $F^{k,\ell}$. Together this gives an $(2+\frac{1+\epsilon}{1-\beta}, \beta)$-approximation algorithm for each $F^{k,\ell}$ (observe that Fact 1 also applies to (α,β)-approximation algorithms). ■

Theorem 10: For every choice of constants $\epsilon > 0$ and $\gamma > 0$, there is a polynomial time $(3+\epsilon)$-approximation algorithm for UFPP restricted to instances where every task is $(1-\gamma)$-small.

Proof: Choose $\ell \in \mathbb{N}$, $q \in \mathbb{N}$, $\beta := 2^{1-q}$ and $\epsilon' > 0$ such that $1-2\beta \geq 1-\gamma$ (hence all tasks are $(1-2\beta)$-small), and $\frac{\ell+q}{\ell} \cdot \left(2 + \frac{1+\epsilon'}{1-\beta}\right) \leq 3+\epsilon$, which is always possible [1]. Now, combining the framework using the chosen ℓ and q (Lemma 3) with $(2+\frac{1+\epsilon'}{1-\beta}, \beta)$-approximation algorithms for $F^{k,\ell}$ (Lemma 7) yields a $(3+\epsilon)$-approximation algorithm. ■

4. Large Tasks

In this section we provide a polynomial time $2k$-approximation algorithm for instances consisting of only $1/k$-large tasks. Then we give our main algorithm, which uses this as a 4-approximation algorithm for the set of $1/2$-large tasks. The main idea is to restrict to UFPP solutions of a certain form: we will represent tasks by rectangles drawn in the plane, and search for independent sets of rectangles.

By (x_1, y_1, x_2, y_2) we will denote the axis-parallel rectangle in the two-dimensional plane with upper left point (x_1, y_1) and lower right point (x_2, y_2). We will call two rectangles (x_1, y_1, x_2, y_2) and (x'_1, y'_1, x'_2, y'_2) *compatible* if they do not overlap (i.e. do not share an internal point). More precisely, they are compatible if at least one of the following holds: $x_2 \leq x'_1$, $x'_2 \leq x_1$, $y_1 \leq y'_2$, or $y'_1 \leq y_2$.

Definition 11 (associated rectangle): With a task i we will associate the rectangle $(s_i, b(i), t_i, \ell(i))$.

[1] For instance, choose β small enough such that $1/(1-\beta) \leq 1+\epsilon'$, $1-2\beta \geq 1-\gamma$, and such that there is an integer $q \geq 2$ with $\beta = 2^{1-q}$. Now choose $\ell \in \mathbb{N}$ such that $\frac{\ell+q}{\ell} \leq 1+\epsilon'$. We obtain an approximation factor of $(2+\frac{1+\epsilon'}{1-\beta}) \cdot \frac{\ell+q}{\ell} \leq (2+(1+\epsilon')^2) \cdot (1+\epsilon')$, which is at most $3+\epsilon$ if ϵ' is sufficiently small.

Note that s_i and t_i are integers since the path vertices are labeled $0, \ldots, m$, and that $b(i)$ and $\ell(i) := b(i) - d_i$ are non-negative integers as well. Tasks are called *compatible* if their associated rectangles are compatible. A task set F is called an *independent task set (ITS)* if all tasks are pairwise compatible.

The following geometric interpretation motivates this choice of rectangles (see also Figure 1): For every edge $e = \{x, x+1\} \in E(P)$ we draw 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* (see the bold grey curve in Figure 1). For every task i, the associated rectangle is now a rectangle of height d_i, drawn as high as possible under the capacity profile. Using this viewpoint, it can easily be checked that if a task set F is an ITS, it is a feasible UFPP solution. To stress this fact, we will also call these sets *ITS solutions*. In Section 4.1 we will show that if all tasks are $\frac{1}{k}$-large, the value of an optimal (i.e. maximum profit) ITS is at most a factor $2k$ worse than the value of an optimal UFPP solution. We will prove this by showing that any UFPP solution can be partitioned into $2k$ ITSs, which is encoded by a $2k$-coloring for the rectangles associated with the solution. After that, we will give a dynamic programming algorithm for finding an optimal ITS in Section 4.2.

4.1. The Coloring Lemma

The result in this section is based on the following property.

Proposition 12: 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 $j \in F \setminus \{i\}$ that are incompatible with i and use e.

Proof: Let the set $F' \subseteq F \setminus \{i\}$ consist of all tasks that are incompatible with i and that use e. Suppose $|F'| \geq k$. Consider $j \in F'$. Since j and i are incompatible, but both use e and e is a bottleneck edge for i, it follows that $b(j) > \ell(i) = b(i) - d_i$. Therefore, $d_i + \sum_{j \in F'} d_j \geq d_i + \frac{1}{k} \sum_{j \in F'} b(j) > d_i + \frac{1}{k}|F'|(b(i) - d_i) \geq b(i) = u_e$, contradicting that F is a feasible solution. ■

A partition $\{F_1, \ldots, F_\ell\}$ of a task set F into ℓ ITSs will be encoded by an *ℓ-coloring* α of F. This is a function $\alpha : F \to \{1, \ldots, \ell\}$ such that if $\alpha(i) = \alpha(j)$, then i and j are compatible. (So $\alpha(i) = x$ means that $i \in F_x$.) 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 Proposition 12, if e is a separator edge, then 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 as its bottleneck edge, 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. In the case that $F = F_0$ or $F = F_1$, we consider a bottleneck edge e_B with minimum capacity, and the set of tasks L with e_B as bottleneck edge. A nice $2k$-coloring of F can be deduced from a nice $2k$-coloring for $F \setminus L$, using the fact that all tasks that are incompatible with tasks in L have separator edges as bottleneck edges, by choice of e_B. This way we can prove the following lemma.

Lemma 13: 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 nice $2k$-coloring for F. For every integer $k \geq 2$, there exist sets F for which $2k$ colors are necessary.

4.2. A Dynamic Programming Algorithm for Finding Optimal ITSs

Throughout this section, we assume w.l.o.g. that all edge capacities are different. This can be achieved by scaling demands and perturbing capacities appropriately. The central concept of our dynamic program is that of a *corner* (x, y, z), which is defined by a pair of points (x, y) and (x, z) in the plane. We first give an informal geometric interpretation of corners: draw a horizontal line segment from point (x, y) to the left until the capacity profile is hit, a horizontal line segment from point (x, z) to the right until the capacity profile is hit, and a vertical line segment between (x, y) and (x, z) (see Figure 1). Together with the capacity profile, these line segments enclose a *region* which we associate with the corner. In our dynamic program, we will compute for every corner the profit of an optimal ITS that fits entirely into this region. This will be done using previously computed values for 'smaller' corners. Informally, 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 now define these notions precisely.

Definition 14: 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 $u_{\{i-1,i\}} > z$. Hence, $w_L \leq x$ and $w_R \geq x$.

For each corner (x, y, z) we define a set $\mathrm{C}(x, y, z)$ which—intuitively speaking—consists of all tasks which are

contained in the region defined by this corner.

Definition 15: For a corner (x, y, z), we denote by $\mathrm{C}(x, y, z)$ 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 $\ell(i) \geq z$.

Due to this definition, we say that *task i fits into the corner* (x, y, z) or *corner* (x, y, z) *contains* i if $i \in \mathrm{C}(x, y, z)$. Hence, $\mathrm{C}(x, y, z)$ is the set of all tasks that fit into the corner. Note that they are possibly incompatible.

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 ITS solutions F with $F \subseteq \mathrm{C}(x, y, z)$. An ITS F with $F \subseteq \mathrm{C}(x, y, z)$ and $w(F) = P(x, y, z)$ is said to *determine* $P(x, y, z)$. One can easily verify that $P(m, 0, u_{\max})$ equals the value of an optimal ITS solution, where $u_{\max} := \max_{e \in E(P)} u_e$. We will now show how $P(x, y, z)$ can be computed in various cases. Proposition 16 collects the easy cases (which are easily understood using the above geometric interpretation), and Lemma 19 covers the complex case which gives the main recursion formula to compute an optimal ITS. In Proposition 16, statement (i) and (ii) show how to rewrite a corner into 'standard form' without changing the corresponding region. Statement (iii) deals with the case where the corner corresponds to two disconnected regions, and shows how to reduce this to two corners for which the region is connected.

Proposition 16:

(i) If $y = z$, then $\mathrm{C}(x, y, z) = \mathrm{C}(w_R(x, z), y, u_{\max})$, and hence $P(x, y, z) = P(w_R(x, z), y, u_{\max})$.

(ii) If $x = 0$ or $y \geq u_{\{x-1,x\}}$ then $\mathrm{C}(x, y, z) = \mathrm{C}(x, u_{\max}, z)$, and hence $P(x, y, z) = P(x, u_{\max}, z)$.

(iii) If $y < z$, $x \geq 1$, $y < u_{\{x-1,x\}}$ and $u_{\{x-1,x\}} \leq z < u_{\{x,x+1\}}$ then $\mathrm{C}(x, y, z) = \mathrm{C}(x, y, u_{\max}) \cup \mathrm{C}(x, u_{\max}, z)$ and hence $P(x, y, z) = P(x, y, u_{\max}) + P(x, u_{\max}, z)$.

Because of Proposition 16 (i), we only need to consider corners (x, y, z) where $y < z$ or $y > z$ holds. Due to the symmetry we will only focus on corners (x, y, z) with $y < z$. By Proposition 16 (ii), we may furthermore assume that $x \geq 1$ and $y < u_{\{x-1,x\}}$. Together with Proposition 16 (iii), this shows that we may restrict our attention to the following type of corners: A corner (x, y, z) is called *proper* if $y < z$, $x \geq 1$, $y < u_{\{x-1,x\}}$ and either $z < u_{\{x-1,x\}}$ or $z \geq u_{\{x,x+1\}}$ holds.

The main idea to handle a proper corner (x, y, z) is as follows. Consider an ITS F that determines $P(x, y, z)$. Either F also fits into the smaller corner $(x - 1, y, z)$, or there exists a task $j \in F$ 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 i. More precisely, we show that there exists a task $i \in F$ such that $F \setminus \{i\} \subseteq \mathrm{C}(s_i, y, b(i)) \cup \mathrm{C}(x, b(i), z)$. The partition is illustrated in Figure 1. Such a task i is called *special*. Informally, the essential property of a special task i is that the rectangle $(s_i, b(i), x, y)$ is compatible with any associated rectangle of a task in $F \setminus \{i\}$, and does not overlap with the capacity profile.

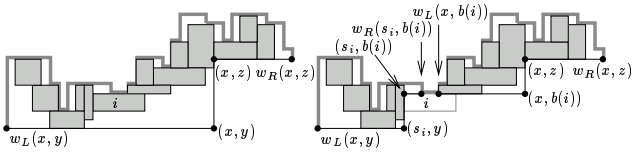

Figure 1. On the left, a proper corner (x, y, z) and an ITS $F \subseteq \mathrm{C}(x, y, z)$ is shown. When choosing a special task $i \in F$, it holds that $F \setminus \{i\} \subseteq \mathrm{C}(s_i, y, b(i)) \cup \mathrm{C}(x, b(i), z)$.

Definition 17: Let (x, y, z) be a proper corner and $F \subseteq \mathrm{C}(x, y, z)$ be an ITS. A task $i \in F$ with $t_i \leq x$ is called *special with respect to F and (x, y, z)* if (1) for all e with $s_i < e < x$ it holds that $u_e \geq b(i)$, and (2) there is no task $j \in F \setminus \{i\}$ whose associated rectangle is incompatible with the rectangle $(s_i, b(i), x, y)$.

We now sketch how to find a special task $i \in F$ in case that $F \not\subseteq \mathrm{C}(x - 1, y, z)$. We start with a task $i \in F$ such that $t_i = x$ and $\ell(i) < z$ holds. Now, assume there is a task $j \in F$ whose associated rectangle is incompatible with the rectangle $(s_i, b(i), x, y)$. Then, j lies below i (that is, $b(j) \leq \ell(i)$), and no associated rectangle of a task in F crosses the line segment from $(s_j, b(j))$ to $(x, b(j))$. This follows essentially from the fact that all tasks in F are compatible with each other, and all associated rectangles touch the capacity profile. We continue the same procedure with j in the role of i until we find a special task i. This yields the following lemma.

Lemma 18: Let (x, y, z) be a proper corner and $F \subseteq \mathrm{C}(x, y, z)$ be an ITS. Then $F \subseteq \mathrm{C}(x - 1, y, z)$ or there exists a special task $i \in F$ with respect to F and (x, y, z).

From Lemma 18 it follows that either $P(x, y, z) = P(x - 1, y, z)$, or the set F that determines $P(x, y, z)$ contains a special task i. Then the partition indicated in Figure 1 yields:

Lemma 19: Consider a UFPP instance in which all edge capacities are distinct. Let (x, y, z) be a proper corner. Then

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

Theorem 20: There is an $O(n^4)$ algorithm for computing maximum profit ITS solutions for a UFPP instance.

Proof sketch: We call a corner triple (x, y, z) *relevant* if both y and z equal the capacity of some edge (or are zero). Our dynamic program uses Proposition 16 and Lemma 19 (or symmetric variants for the case $y > z$) to compute $P(x, y, z)$ for all relevant corner triples (x, y, z), using previously computed values $P(x', y', z')$. Note that every such corner (x', y', z') that is used is actually relevant. By considering corners in order according to an appropriate size measure (related to their region), one can guarantee that all necessary values $P(x', y', z')$ have previously been computed. In the end we return $P(m, 0, u_{\max})$. Since we may assume w.l.o.g. that the path has $m < 2n$ edges, there are at most $O(n^3)$ relevant corner triples. Evaluating the expressions in Proposition 16 and Lemma 19 takes time $O(n)$. ■

Our results can be formulated as follows in terms of the Maximum Weight Independent Set of Rectangles (MWISR) problem:

Theorem 21: MWISR can be solved in time $O(n^4)$ for instances with n axis-parallel rectangles, in which each rectangle contains a point (x, y) such that no point (x, y') with $y' > y$ is part of a rectangle.

Recall that ITSs are feasible UFPP solutions. Combining Theorem 20 with Lemma 13 (applied to an optimal UFPP solution) yields:

Theorem 22: For every integer $k \geq 2$, there is an $O(n^4)$ time $2k$-approximation algorithm for UFPP restricted to instances where every task is $\frac{1}{k}$-large.

Finally, we can combine this with Theorem 10 to obtain our main result.

Theorem 23: For every $\epsilon > 0$ there is a $(7 + \epsilon)$-approximation algorithm for the UFPP problem.

Proof: For the $\frac{1}{2}$-small tasks, we have a $(3 + \epsilon)$-approximation algorithm (Theorem 10), and for the $\frac{1}{2}$-large tasks, a 4-approximation algorithm (Theorem 22). Returning the best solution of the two then yields a $(7+\epsilon)$-approximation algorithm (Fact 1). ■

5. Strong NP-Hardness

In this section we shortly sketch our proof that UFPP is strongly NP-hard for instances with demands in $\{1, 2, 3\}$, which uses a reduction from Maximum Independent Set in Cubic Graphs (for details see [12]). Let G, k be an independent set instance: $G \neq K_4$ is a connected cubic graph (i.e. all vertices have degree 3), and the question is whether G contains an independent set of size at least k. This problem is NP-hard [26]. Let $V(G) = \{v_1, \ldots, v_n\}$, and $E(G) = \{e_1, \ldots, e_m\}$. By Brooks' Theorem (see e.g. [30]), G is 3-colorable since $G \neq K_4$ and G is connected. Such a coloring can be found in polynomial time [30], so for our polynomial transformation we may assume that a proper 3-coloring $\alpha : V(G) \to \{1, 2, 3\}$ is given.

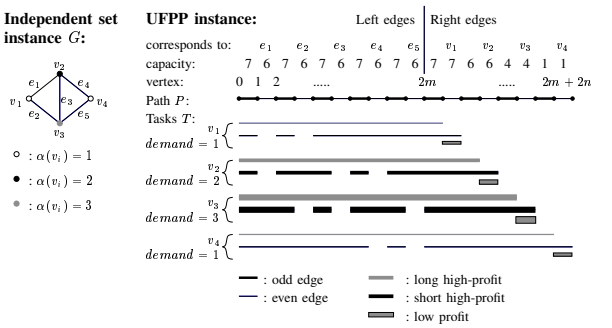

Figure 2. An example of a graph G with coloring α and the resulting instance UFPP(G).

We now construct an equivalent instance of UFPP as follows; see Figure 2. (For ease of presentation, in the figure we have chosen G to be a small *non-cubic* graph. This is no problem for the transformation.) The path P has $2n+2m+1$ vertices, labeled $0, \ldots, 2n+2m$. An edge $\{i-1, i\} \in E(P)$ is called an *odd edge* if i is odd, and an *even edge* otherwise. 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 $\sigma_1, \sigma_2, \sigma_3$ be the indices of the edges incident with v_i, in increasing order. Introduce four *short* tasks with the following start and end vertex pairs: $(0, 2\sigma_1 - 1), (2\sigma_1, 2\sigma_2 - 1), (2\sigma_2, 2\sigma_3 - 1), (2\sigma_3, 2m + 2i)$. 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 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 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 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 UFPP(G) of UFPP. With an independent set I of G we associate a solution of UFPP(G) of the following form: A set $F \subseteq T$ of tasks of UFPP(G) is said to be in *standard form* if the following two conditions hold. Firstly, 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, or F contains all short high-profit tasks for v_i. Secondly, 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.

For every independent set I of G, we can construct a set F in standard form by selecting the long high-profit task for v_i if and only if $v_i \in I$. It can be shown that any set F in standard form is a feasible solution for UFPP(G), and that

its profit is $M + x$, where M is a large (but polynomially bounded) constant and x is the number of included long high-profit tasks. In the other direction, it can be shown that every optimal solution F for UFPP(G) is in standard form: because of the way the profits were chosen (depending on the number of odd edges used by the task, and proportional to the demand of the task), it can be verified that any optimal solution must use the full capacity of all odd edges. Using this observation and the fact that the demands were chosen according to a proper coloring of G, one can show that every optimal solution is in standard form, by arguing from right to left along the path. Hence UFPP(G) has a solution with profit at least $M + x$ if and only if G has an independent set of size at least x. This proves that UFPP is strongly NP-hard. In fact, by extending this construction using high-profit dummy tasks, which have to be selected in any optimal solution, it can be shown that the problem remains NP-hard for uniform edge capacities.

Theorem 24: UFPP is strongly NP-hard when restricted to instances with demands in $\{1, 2, 3\}$, where all edges have equal capacities.

6. Discussion and Further Results

We presented a number of new techniques for UFPP, and combined these to obtain our main result, a $(7 + \epsilon)$-approximation algorithm. By combining these techniques and lemmas differently, we can obtain a number of related results, which we will now shortly discuss. For proofs we refer to our technical report [12]. We essentially partitioned the tasks into three groups: the $\frac{1}{2}$-large tasks were handled with a geometric dynamic program, in time $O(n^4)$ (Theorem 22). For some $\delta > 0$, the δ-small tasks were handled with linear programming techniques (Lemma 8), which can be done in time $O(n^4 \log n)$ using flow techniques (see also [4], [13]). The remaining tasks were handled with a computationally much more costly dynamic program. By omitting the latter step, partitioning the tasks into $\frac{1}{9}$-small and $\frac{1}{9}$-large tasks, and applying our framework with $\ell = 3$ and $q = 5$, we can prove:

Theorem 25: There is a 25.12-approximation algorithm for UFPP with running time $O(n^4 \log n)$.

In addition, using our techniques we can show that if we increase the capacity of each edge by a factor of $(1 + \beta)$ for a small value $\beta > 0$ (called *resource augmentation*), then we can compute in polynomial time a solution whose value differs by at most a factor of $(2 + \epsilon)$ from the original optimum. In this case we do not need the dynamic program for the large tasks from Section 4, and we may use an optimal solution for the δ-large tasks, instead of a $(2, \beta)$-approximation (cf. Lemma 9).

Theorem 26: Let $\epsilon > 0$ and $\beta > 0$ be constants. There is a polynomial time algorithm that, given a UFPP instance with task set T, computes a task set $F \subseteq T$ such that

- $(2 + \epsilon)w(F) \geq w(OPT(T))$, and
- F is feasible if the capacity of each edge is increased by a factor of $(1 + \beta)$.

Chakrabarti et al. [16] show that any α-approximation algorithm for UFPP implies a $(1 + \alpha + \epsilon)$-approximation algorithm for cycles (also called ring networks). Combined with Theorem 23, this yields an $(8 + \epsilon)$-approximation algorithm for the *Unsplittable Flow Problem on Cycles*. Note that on a cycle, for each task i two possible paths can be chosen from s_i to t_i.

As we discussed in the introduction, constant factor approximation algorithms for many generalizations of UFPP-NBA are known, such as in a scheduling context [8], [32] or in a network flow context [16], [19]. However, all such known results require the no-bottleneck assumption, or more restrictively, consider the uniform capacity case. Now that a constant approximation for general UFPP instances is known, an obvious open question is for which generalizations of UFPP constant factor approximation algorithms can be obtained. In particular, can our results be generalized to other graph classes, such as trees?

The most intriguing question that we leave open is whether there exists a PTAS for UFPP or whether the problem is in fact APX-hard. This is also open for the uniform capacity case (which implicitly satisfies the NBA).

Acknowledgment

We would like to thank David Williamson for helpful discussions on a draft of this paper, and Marek Chrobak for pointing us to [21].

References

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

[2] M. Andrews, J. Chuzhoy, S. Khanna, and L. Zhang, "Hardness of the undirected edge-disjoint paths problem with congestion," in *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS '05)*. IEEE Computer Society, 2005, pp. 226–244.

[3] M. Andrews and L. Zhang, "Hardness of the undirected edge-disjoint paths problem," in *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC '05)*. ACM, 2005, pp. 276–283.

[4] E. M. Arkin and E. B. Silverberg, "Scheduling jobs with fixed start and end times," *Discrete Appl. Math.*, vol. 18, pp. 1–8, 1987.

[5] Y. Azar and O. Regev, "Combinatorial algorithms for the unsplittable flow problem," *Algorithmica*, vol. 44, no. 1, pp. 49–66, 2006.

[6] N. Bansal, A. Chakrabarti, A. Epstein, and B. Schieber, "A quasi-PTAS for unsplittable flow on line graphs," in *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC '06)*. ACM, 2006, pp. 721–729.

[7] N. Bansal, Z. Friggstad, R. Khandekar, and R. Salavatipour, "A logarithmic approximation for unsplittable flow on line graphs," in *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '09)*. SIAM, 2009, pp. 702–709.

[8] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber, "A unified approach to approximating resource allocation and scheduling," in *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC '00)*. ACM, 2000, pp. 735–744.

[9] R. Bar-Yehuda, M. Beder, Y. Cohen, and D. Rawitz, "Resource allocation in bounded degree trees," in *Proceedings of the 14th Annual European Symposium Algorithms (ESA '06)*, ser. LNCS, vol. 4168. Springer, 2006, pp. 64–75.

[10] M. Bartlett, A. M. Frisch, Y. Hamadi, I. Miguel, A. Tarim, and C. Unsworth, "The temporal knapsack problem and its solution," in *Proceedings of Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR '05)*, ser. LNCS, vol. 3524. Springer, 2005, pp. 34–48.

[11] A. Baveja and A. Srinivasan, "Approximation algorithms for disjoint paths and related routing and packing problems," *Math. Oper. Res.*, vol. 25, pp. 255–280, May 2000.

[12] P. Bonsma, J. Schulz, and A. Wiese, "A constant factor approximation algorithm for unsplittable flow on paths," *CoRR*, vol. abs/1102.3643, 2011.

[13] G. Calinescu, A. Chakrabarti, H. Karloff, and Y. Rabani, "Improved approximation algorithms for resource allocation," in *Proceedings of the 9th International Conference on Integer Programming and Combinatorial Optimization (IPCO '02)*, ser. LNCS, vol. 2337. Springer, 2002, pp. 401–414.

[14] V. T. Chakaravarthy, A. Kumar, S. Roy, and S. Sabharwal, "Minimum cost resource allocation for meeting job requirements," in *Proceedings of the 19th European Symposium on Algorithms (ESA '11)*. Springer, 2011, to appear.

[15] 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)*, ser. LNCS, vol. 2462. Springer, 2002, pp. 51–66.

[16] ——, "Approximation algorithms for the unsplittable flow problem." *Algorithmica*, pp. 53–78, 2007.

[17] P. Chalermsook and J. Chuzhoy, "Maximum independent set of rectangles," in *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '09)*. SIAM, 2009, pp. 892–901.

[18] C. Chekuri, A. Ene, and N. Korula, "Unsplittable flow in paths and trees and column-restricted packing integer programs," in *Proceedings of the 12th International Workshop (APPROX '09) and 13th International Workshop (RANDOM '09) on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, ser. LNCS, vol. 5687. Springer, 2009, pp. 42–55.

[19] C. Chekuri, M. Mydlarz, and F. Shepherd, "Multicommodity demand flow in a tree and packing integer programs," *ACM Trans. on Algorithms*, vol. 3, 2007, 27 pp.

[20] B. Chen, R. Hassin, and M. Tzur, "Allocation of bandwidth and storage," *IIE Transactions*, vol. 34, pp. 501–507, 2002.

[21] M. Chrobak, G. Woeginger, K. Makino, and H. Xu, "Caching is hard even in the fault model," in *Proceedings of the 18th Annual European Symposium on Algorithms (ESA '10)*, ser. LNCS, vol. 6346. Springer, 2010, pp. 195–206.

[22] A. Darmann, U. Pferschy, and J. Schauer, "Resource allocation with time intervals," *Theor. Comp. Sc.*, vol. 411, pp. 4217–4234, 2010.

[23] C. W. Duin and E. V. Sluis, "On the complexity of adjacent resource scheduling," *J. of Scheduling*, vol. 9, pp. 49–62, 2006.

[24] T. Erlebach, K. Jansen, and E. Seidel, "Polynomial-time approximation schemes for geometric graphs," in *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '01)*. SIAM, 2001, pp. 671–679.

[25] 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.*, vol. 15, pp. 100–109, 1984.

[26] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified NP-complete graph problems," *Theor. Comp. Sc.*, vol. 1, pp. 237–267, 1976.

[27] N. Garg, V. V. Vazirani, and M. Yannakakis, "Primal-dual approximation algorithms for integral flow and multicut in trees," *Algorithmica*, vol. 18, no. 1, pp. 3–20, 1997.

[28] S. Khanna, S. Muthukrishnan, and M. Paterson, "On approximating rectangle tiling and packing," in *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '98)*. SIAM, 1998, pp. 384–393.

[29] 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 (FSTTCS '00)*, ser. LNCS, vol. 1974. Springer, 2000, pp. 409–420.

[30] L. Lovász, "Three short proofs in graph theory," *J. Combinatorial Theory Ser. B*, vol. 19, pp. 269–271, 1975.

[31] F. Nielsen, "Fast stabbing of boxes in high dimensions," *Theor. Comp. Sc.*, vol. 246, pp. 53 – 72, 2000.

[32] C. A. Phillips, R. N. Uma, and J. Wein, "Off-line admission control for general scheduling problems," in *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '00)*. ACM, 2000, pp. 879–888.