Constant Integrality Gap LP formulations of Unsplittable Flow on a Path *

Aris Anagnostopoulos[1], Fabrizio Grandoni[2], Stefano Leonardi[1], and Andreas Wiese[3]

[1] Sapienza University of Rome, Italy. {aris,leon}@dis.uniroma1.it
[2] University of Lugano, Switzerland. fabrizio@idsia.ch
[3] Max-Planck-Institut für Informatik, Germany. awiese@mpi-inf.mpg.de

Abstract. The *Unsplittable Flow Problem on a Path* (UFPP) is a core problem in many important settings such as network flows, bandwidth allocation, resource constraint scheduling, and interval packing. We are given a path with capacities on the edges and a set of tasks, each task having a demand, a profit, a source and a destination vertex on the path. The goal is to compute a subset of tasks of maximum profit that does not violate the edge capacities.

In practical applications generic approaches such as integer programming (IP) methods are desirable. Unfortunately, no IP-formulation is known for the problem whose LP-relaxation has an integrality gap that is provably constant. For the unweighted case, we show that adding a few constraints to the standard LP of the problem is sufficient to make the integrality gap drop from $\Omega(n)$ to $O(1)$. This positively answers an open question in [Chekuri et al., APPROX 2009].

For the general (weighted) case, we present an extended formulation with integrality gap bounded by $7 + \varepsilon$. This matches the best known approximation factor for the problem [Bonsma et al., FOCS 2011]. This result exploits crucially a technique for embedding dynamic programs into linear programs. We believe that this method could be useful to strengthen LP-formulations for other problems as well and might eventually speed up computations due to stronger problem formulations.

1 Introduction

In the *Unsplittable Flow Problem on a Path* (UFPP) we are given a set of n tasks $\mathcal{T}$ and a path $G = (V, E)$ on m edges. For each edge e denote by u_e its capacity. Each task $T_i \in \mathcal{T}$ is specified by a start vertex $s_i \in V$, a destination vertex $t_i \in V$, a demand d_i and a weight (or profit) w_i. For each edge $e \in E$ denote by $\mathcal{T}_e$ all tasks T_i such that the (unique) path from s_i to t_i uses e. Also, we abuse slightly notation and we denote by T_i the set of edges in the path from

* Partially supported by EU FP7 Project N. 255403 SNAPS, by the ERC Starting Grant NEWNET 279352, by the ERC Starting Grant PAAl 259515, and by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD).

s_i to t_i. For each task T_i we define its *bottleneck capacity* $b_i := \min\{u_e : e \in T_i\}$. For a value $\delta \in (0,1)$ we say that a task T_i is δ-large if $d_i > \delta \cdot b_i$ and δ-*small* otherwise. The goal is to select a subset of the tasks $\mathcal{T}' \subseteq \mathcal{T}$ with maximum total weight $w(\mathcal{T}') := \sum_{T_i \in \mathcal{T}'} w_i$ such that $\sum_{T_i \in \mathcal{T}_e \cap \mathcal{T}'} d_i \leq u_e$ for all edges e. In the *unweighted* case, all weights are 1.

This problem occurs in various settings and important applications. As the name suggests, it is a special case of multi-commodity demand flow, with one task associated to each commodity. This problem clearly generalizes well known problems such as knapsack and maximum independent set in interval graphs. It can be used to model the availability over time of a resource of varying capacity, with each task demanding a specific amount of the resource within a fixed time interval. Despite their fundamental nature, the combinatorial structure and the polynomial-time approximability of this problem are not yet well understood. UFPP is strongly NP-hard [5, 12] and the best known approximation results are a quasi-PTAS [1] and a polynomial time $(7+\epsilon)$-approximation algorithm [5].

When solving optimization problems in practice, a common method is to formulate the problem as an integer linear program (ILP) and use an Integer Programming (IP) solver such as CPLEX or Gurobi. However, for many problems there are several possible ILP formulations which perform very differently in practice. One desired property of a good ILP formulation is that the resulting LP relaxation has a small integrality gap. This is helpful since in branch-and-cut algorithms LP relaxations are used to derive good lower bounds, which allow one to neglect certain subtrees and thus speed up the computation. Most of previous LP based approaches for UFPP refer to the following natural LP formulation:

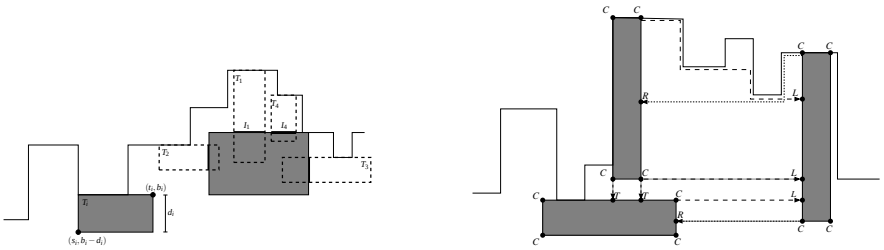
$$\mathrm{LP}_{\mathrm{UFPP}} = \{\max \sum_{i=1}^{n} w_i \cdot x_i \ : \ \sum_{T_i \in \mathcal{T}_e} d_i \cdot x_i \leq u_e, \forall e \in E; 0 \leq x_i \leq 1, i = 1, \ldots, n.\}$$

Unfortunately, $\mathrm{LP}_{\mathrm{UFPP}}$ suffers from an integrality gap $\Omega(n)$ [7]. Chekuri et al. [10] presented an LP formulation with integrality gap of at most $O(\log^2 n)$ obtained by adding an exponential number of constraints to the above LP, which can be approximately separated in polynomial time. (Recently they showed how to obtain a polynomial-size formulation and improved the integrality gap to $O(\log n)$ [9].)

1.1 Our Contribution

In this paper we address the open problem of designing LP relaxations for UFPP with small (namely constant) integrality gap. Our main contributions are as follows:

Unweighted UFPP. We present the first LP relaxation for unweighted UFPP with provably constant integrality gap (see Section 2). Even though the canonical LP-relaxation $\mathrm{LP}_{\mathrm{UFPP}}$ has a very large integrality gap of $\Omega(n)$, we show that by adding only $O(n^2)$ constraints it drops to $O(1)$, independently of the input size. We show that up to constant factors our integrality gap is bounded by the



(a) The shaded and dashed rectangles are in $\mathcal{T}_k$ and $\mathcal{T} \setminus \mathcal{T}_{\text{points}}$, respectively. Top ($T_1$ and T_4), left (T_2), and right (T_3) intersecting rectangles.

(b) Set of points Q. For each rectangle in $\mathcal{T}_k$ there are the 4 corner points (C), (at most) 2 points downwards (T), 2 points rightwards (L), and 2 points leftwards (R).

Fig. 1. Examples of some of the notions.

worst-case integrality gap of the canonical LP for the *Maximum Independent Set of Rectangles* problem (MISR) for instances that stem from 1/2-large tasks in the sense described in [5]. Intuitively, we construct a rectangle with base along the subpath of T_i and height equal to its demand d_i, and then push it as high as possible while remaining below the curve induced by the capacities (see Figure 1(a)).

Bounding the integrality gap of the canonical MISR formulation has been a challenging open problem for a long time. We show that for unweighted instances stemming from UFPP the worst-case integrality gap is $O(1)$. Even more, we provide a purely combinatorial algorithm whose profit is by at most a constant factor smaller than the optimal LP value on those rectangles. Given that the general case of that problem is hard to tackle (no constant factor approximation algorithms are known, whereas the best known lower bound is just NP-hardness) we hope that our result helps understanding this important problem better.

The authors of [10] consider a relaxation of UFPP with an exponential number of additional constraints, and show that it has an $O(\log^2 n)$ integrality gap. Our reasoning implies that in the unweighted case already a polynomial size subset of those constraints yields a formulation with $O(1)$-integrality gap, thus answering an open question posed in [10].

Weighted UFPP. In [16] Martin et al. show a generic method to formulate dynamic programs as linear programs. Roughly speaking, they show that for any (well-behaved) DP one can construct a linear program whose extreme points correspond to the possible outputs of the DP, given suitable weights to the items (jobs). in the input. In the course of our research, and unaware of the result in [16], we developed a slightly different approach for embedding a DP into an LP. It turns out that our approach is somewhat simpler and marginally more general (their approach requires $\alpha_i^C = 1$—see Section 3, whereas a simple extension gives a pseudopolynomial number of variables), so we include it for completeness.

We combine this DP-embedding theorem with dynamic programs [5] for subcases of UFPP for which the canonical LP has a large integrality gap. By embedding the DP for 1/2-large tasks from [5] we obtain an extended LP relaxation for weighted UFPP with a constant integrality gap (see Section 4). No relaxation with a constant integrality gap was known before. By embedding additionally the DPs for the tasks that are δ-large and 1/2-small, we obtain a formulation whose integrality gap is bounded by $7+\epsilon$, together with a matching polynomial-time rounding procedure. This improves slightly the result in [5], where the same approximation factor is proved w.r.t. the integral optimal profit only.

To the best of our knowledge, this is the first time that the structural insight of [16] is used to strengthen a linear programming formulation, especially to this magnitude (from $\Omega(n)$ to $7+\epsilon$). We believe that this technique could be useful for improving the IP-formulations of other problems as well. Eventually, this might lead to better running times of IP-solvers in practice due to stronger formulations.

1.2 Preliminaries and Related Work

UFPP is weakly NP-hard for the special case of a single edge since then it is equivalent to the knapsack problem. It admits a PTAS for constant number of edges since it reduces to multi-dimensional knapsack [14]. For an arbitrary number of edges the problem is strongly NP-hard [5, 12], thus excluding an FPTAS if P $\neq$ NP. In terms of approximation algorithms, the first nontrivial result for UFPP was given by Bansal et al. [2] who gave an $O(\log n)$ approximation algorithm. In a previous paper, Bansal et al. [1] gave a QPTAS for the problem, which requires a quasi-polynomial bound on the edge-capacities and demands. Motivated by the work of [2], Chekuri et al. [10] presented an LP formulation with integrality gap $O(\log^2 n)$, obtained by adding a super-polynomial number of constraints to the natural LP formulation given above. The separation routine given in [10] loses an $O(1)$ factor. The algorithms for UFPP usually distinguish between small and large tasks. In a recent result, Bonsma et al. [5] gave a first $O(1)$ approximation $(7+\epsilon)$ for general UFPP, by designing a dynamic-programming algorithm for MISR and using the solution for UFPP.

A well-studied special case of UFPP is given by the *no-bottleneck assumption* (NBA), which requires that $\max_i\{d_i\} \leq \min_e\{u_e\}$. For UFPP-NBA, dynamic programming exploits the fact that on each edge any solution can have at most $2\lfloor 1/\delta^2 \rfloor$ tasks that are δ-large [7]. (Unfortunately, this property does not hold in the general case). Together with an LP rounding procedure for the remaining tasks this yields the best known approximation algorithm for UFPP-NBA, having a ratio of $2+\epsilon$ [11]. Previously, a similar approximation result was obtained, after a sequence of improvements, for the special case of the *resource-allocation problem* (RAP), which is given by the constraint that all the edges have equal capacity [3, 6, 13]. The natural LP formulation of UFPP has an $O(1)$ integrality gap for UFPP-NBA [7, 11]. The integrality gap is however not less than 2.5 [11], that is, larger than the best known approximation ratio. In [7] it is left open to

find an LP relaxation with constant integrality gap for general UFPP on large tasks even for the unweighted case where all tasks have equal profit.

As we mentioned, the authors of [5] established a connection between UFPP and MISR. For the latter problem, the best known approximation ratio is $O(\log \log n)$ [8] (and $O(\log n)$ in the weighted case, see for example, [4, 15]). It is still open to find an $O(1)$ approximation algorithm for MISR, even only for the unweighted case. In particular, the exact integrality gap of the standard LP formulation for the problem is not known. The best known lower and upper bounds for it (in the unweighted case) are $3/2$ and $O(\log \log n)$ [8], respectively.

2 Constant Integrality Gap for Unweighted UFPP

In this section we show how to strengthen the canonical linear program $\mathrm{LP}_{\mathrm{UFPP}}$ for unsplittable flow to make its integrality gap drop from $\Omega(n)$ to $O(1)$. Let us focus for a moment on 1/2-large tasks $\mathcal{T}_{\mathrm{large}}$ only (which we next call *large* for brevity). For those, in [5] the following geometrical interpretation was introduced: for each task T_i draw a rectangle T_i specified by the upper left point (s_i, b_i) and the lower right point (t_i, ℓ_i) where $\ell_i := b_i - d_i$, where we interpret the vertices of the path as integers. In [5] the authors show that any feasible integral solution $\mathcal{T}'$ consisting of only large tasks has the property that any point in the plane can be covered by (i.e., is contained in the interior of) at most four rectangles in $\mathcal{T}'$ [5, Lemma 13]. Due to the geometry of the rectangles, to check the above property it is sufficient to consider a proper subset P of only $O(n^2)$ points.

Our main idea is to add the corresponding set of feasible constraints to the standard LP for unweighted UFPP, therefore obtaining the following refined LP:

$$\mathrm{LP}^{+}_{\mathrm{UFPP}} := \{\max \sum_{T_i \in T} x_i \text{ s.t. } \sum_{T_i \in T_e} x_i \cdot d_i \leq u_e, \forall e \in E; \sum_{T_i \in \mathcal{T}_{\mathrm{large}}:\, p \in T_i} x_i \leq 4, \forall p \in P; \;\; x_i \geq 0, \forall T_i \in \mathcal{T}\}$$

By the above reasoning any integral solution satisfies the added constraints.

We recall that for any $\delta \in (0, 1)$ (hence, in particular, for $\delta = 1/2$), the canonical LP has already a constant integrality gap for instances with only $(1 - \delta)$-small tasks. Therefore, it is sufficient to bound the integrality gap for large tasks only. Observe that, given a feasible solution x to $\mathrm{LP}^{+}_{\mathrm{UFPP}}$, the vector $y_i := x_i/4$, $T_i \in \mathcal{T}_{\mathrm{large}}$, yields a feasible solution for the following linear program:

$$\mathrm{LP}_{\mathrm{MISR}} := \{\max \sum_{T_i \in \mathcal{T}_{\mathrm{large}}} y_i \text{ s.t. } \sum_{T_i \in \mathcal{T}_{\mathrm{large}}:\, p \in T_i} y_i \leq 1, \forall p \in P;\; y_i \geq 0, \forall T_i \in \mathcal{T}_{\mathrm{large}}\}$$

The above LP is the canonical LP for MISR on rectangles $\mathcal{T}_{\mathrm{large}}$. By the definition of the heights of the rectangles, it is easy to see that any independent set of rectangles $\mathcal{T}' \subseteq \mathcal{T}_{\mathrm{large}}$ induces a feasible UFPP-solution. This yields the following lemma.

Lemma 1. *Assume that* $\mathrm{LP}_{\mathrm{MISR}}$ *has an integrality gap of* α *for instances that stem from* UFPP *instances and that* $\mathrm{LP}_{\mathrm{UFPP}}$ *has an integrality gap of* β *for instances with only* $1/2$*-small tasks. Then* $\mathrm{LP}^{+}_{\mathrm{UFPP}}$ *has an integrality gap of at most* $4\alpha + \beta$*.*

2.1 A Combinatorial Algorithm for Large Tasks

It remains to show that $\mathrm{LP}_{\mathrm{MISR}}$ has an integrality gap of $\alpha = O(1)$. To this aim, we describe a combinatorial algorithm that computes an independent set of rectangles whose cardinality is at most by a constant factor smaller than the value of the optimal LP solution. Suppose we are given a set of rectangles $\mathcal{T}$ stemming from (the large tasks of) a UFPP instance. Our algorithm runs in phases where in each phase k we either compute a maximal set $\mathcal{T}_{k+1}$ based on the set $\mathcal{T}_k$ computed in the previous iteration such that $|\mathcal{T}_{k+1}| > |\mathcal{T}_k|$ or assert that $|\mathcal{T}_k|$ is large in comparison with the LP optimum. Because the optimal solution can contain at most n rectangles, there can be at most n phases. We start with any maximal independent set of rectangles $\mathcal{T}_0 \subseteq \mathcal{T}$, which can be trivially computed (say, using a greedy algorithm). Now suppose that we have computed a set $\mathcal{T}_k$. For each rectangle $T_i \in \mathcal{T}_k$ we identify at most ten points Q_i in the plane (see Figure 1(b)). The first four points in Q_i are the four corners of T_i (points C in Figure 1(b)). The other six points are obtained as follows:

- Take the bottom-left (the bottom-right) corner and move down until you hit the boundary of another rectangle in $\mathcal{T}_k$ (points T in Figure 1(b)), if any.
- The points (points L in Figure 1(b)) which are defined by the following process: Start from the top-right (also bottom-right) corner; call this point (x, y). Iteratively execute the following step until you hit the boundary of another rectangle in $\mathcal{T}_k$, if any: If $y \leq u_{\{x,x+1\}}$ then set $(x, y) \leftarrow (x + 1, y)$. Otherwise, set $(x, y) \leftarrow (x, y - 1)$.
- Similarly, going leftwards starting from the top-left and bottom-left points (points R in Figure 1(b)).

We denote by $\mathcal{T}_{\mathrm{points}} \subseteq \mathcal{T}$ all rectangles in the instance that overlap some point in $Q := \bigcup_{T_i \in \mathcal{T}_k} Q_i$. We show later in Lemma 4 that those tasks have bounded LP-weight. Consider the remaining rectangles $\mathcal{T} \setminus \mathcal{T}_{\mathrm{points}}$. First observe that because $\mathcal{T}_k$ is maximal, every rectangle in $\mathcal{T} \setminus \mathcal{T}_{\mathrm{points}}$ must intersect a rectangle of $\mathcal{T}_k$. We classify $\mathcal{T} \setminus \mathcal{T}_{\mathrm{points}}$ as *top-intersecting* rectangles $\mathcal{T}_{\mathrm{top}}$, *left-intersecting* rectangles $\mathcal{T}_{\mathrm{left}}$, and *right-intersecting* rectangles $\mathcal{T}_{\mathrm{right}}$. We call a rectangle T_i top intersecting if there exists a rectangle $T_j \in \mathcal{T}_k$ such that $s_j < s_i < t_i < t_j$, and $\ell_i < b_j < b_i$. We call a rectangle T_i left (resp., right) intersecting if there exists a rectangle $T_j \in \mathcal{T}_k$ such that $\ell_j < \ell_i < b_i < b_j$, and $s_i < s_j < t_i$ (resp., $s_i < t_j < t_i$) (see Figure 1(a)). We can then prove the following lemma:

Lemma 2. *All the rectangles in* $\mathcal{T} \setminus \mathcal{T}_{\mathrm{points}}$ *are either top-intersecting, left-intersecting, or right-intersecting.*

In our algorithm we now take each set $\mathcal{T}_{\text{top}}$, $\mathcal{T}_{\text{left}}$, and $\mathcal{T}_{\text{right}}$ separately and compute an optimal solution for it. The crucial observation is now that this problem is equivalent to the maximum independent set problem in interval graphs. To this end, construct a graph $G_{\text{top}} = (V_{\text{top}}, E_{\text{top}})$ where V_{top} consists of one vertex v_j for each rectangle $T_j \in \mathcal{T}_{\text{top}}$ and an edge $\{v_j, v'_j\}$ exists if and only if T_j and $T_{j'}$ overlap. Define the graphs G_{left} and G_{right} similarly.

Lemma 3. *The graphs G_{top}, G_{left}, and G_{right} are interval graphs.*

The proof is very technical, so we only provide an intuition for the top-intersecting rectangles—the argument for the other two cases is similar albeit somewhat more complicated. It relies highly on the definition of the points Q. Consider two top-intersecting rectangles T_1 and T_4 (see Figure 1(a)). To each of them corresponds an interval I_1 and I_4, defined by the intersection of the rectangle with the rectangle to which they intersect. Note that I_1 and I_4 intersect if and only if T_1 and T_4 overlap. To compute a maximum independent set, it suffices to preorder the rectangles according to their values t_i and at iteration k, for each interval, select rectangles greedily by increasing values of t_i [17].

Denote by OPT_{top}, OPT_{left}, and $\text{OPT}_{\text{right}}$ the optimal independent sets for G_{top}, G_{left}, and G_{right}, respectively. Now there are two cases. If $|\text{OPT}_{\text{top}}| \leq |\mathcal{T}_k|$ and $|\text{OPT}_{\text{left}}| \leq |\mathcal{T}_k|$ and $|\text{OPT}_{\text{right}}| \leq |\mathcal{T}_k|$, then we output $\mathcal{T}_k$ and halt. Otherwise we define $\mathcal{T}_{k+1}$ to be the set of maximum cardinality among OPT_{top}, OPT_{left}, and $\text{OPT}_{\text{right}}$, and we proceed with the next iteration.

Suppose now that our algorithm runs for k iterations and finally outputs the set $\mathcal{T}_k$. We want to bound its cardinality in comparison with the optimal fractional solution to LP_{MISR}. By Lemma 2 the union of the sets $\mathcal{T}_{\text{points}}, \mathcal{T}_{\text{top}}, \mathcal{T}_{\text{left}}, \mathcal{T}_{\text{right}}$ equals $\mathcal{T}$. We bound the LP profit for each of these sets separately. Note that the next lemma is the only part of our reasoning where we use that the rectangles are unweighted.

Lemma 4. *In any feasible solution y to LP_{MISR} the profit of the rectangles in $\mathcal{T}_{\text{points}}$ is bounded by $\sum_{p \in Q} \sum_{T_i \colon p \in T_i} y_i \leq 10 \cdot |\mathcal{T}_k|$.*

Proof (sketch). Let $Q_{\text{TL}} \subseteq Q$ be the set of top-left corners of all the rectangles in $\mathcal{T}_k$. Notice that $|Q_{\text{TL}}| = |\mathcal{T}_k|$. We have $\sum_{p \in Q_{\text{TL}}} \sum_{T_i \colon p \in T_i} y_i \leq \sum_{p \in Q_{\text{TL}}} 1 \leq |\mathcal{T}_k|$, where the first inequality follows from the constraints of LP_{MISR}. By performing the same approach for all the 10 families of points that constitute the set Q, we obtain the lemma.

Lemma 5. *For any feasible solution y to LP_{MISR} and any set $\mathcal{T}' \in \{\mathcal{T}_{\text{top}}, \mathcal{T}_{\text{left}}, \mathcal{T}_{\text{right}}\}$ it holds that $\sum_{T_i \colon \mathcal{T}'} y_i \leq OPT(\mathcal{T}')$, where $OPT(\mathcal{T}')$ stands for OPT_{top}, OPT_{left}, or $\text{OPT}_{\text{right}}$.*

Proof. By Lemma 3 the graphs G_{top}, G_{left}, and G_{right} are interval graphs and hence in particular perfect graphs. Therefore, for the maximum independent set problem the following LP formulation is exact: introduce a variable $x_v \geq 0$ for every vertex $v \in V$ and the clique inequality $\sum_{v \in C} x_v \leq 1$ for all maximal cliques $C \subseteq V$ (see, for example, [17]). In MISR, for every maximal clique $C \subseteq \mathcal{T}$ we

can find a point in the plane that is covered by all the rectangles in C. Hence, $\mathrm{LP}_{\mathrm{MISR}}$ contains a clique inequality for each maximal clique in the graphs G_{top}, G_{left}, and G_{right}. Thus, $\mathrm{LP}_{\mathrm{MISR}}$ cannot gain more profit than the respective optimal integral solution for these subproblems.

Theorem 1. *Consider the set of rectangles $\mathcal{T}$ in the plane that stem from a* UFPP*-instance. There is a polynomial-time algorithm that computes a set $\mathcal{T}' \subseteq \mathcal{T}$ such that $\sum_{T_i \in \mathcal{T}} y_i \leq 13 \cdot |\mathcal{T}'|$ for any feasible solution y of* $\mathrm{LP}_{\mathrm{MISR}}$.

Proof. By Lemma 4 and 5 we have $\sum_{T_i \in \mathcal{T}} y_i = \sum_{T_i \in \mathcal{T}_{\mathrm{points}}} y_i + \sum_{T_i \in \mathcal{T}_{\mathrm{top}}} y_i + \sum_{T_i \in \mathcal{T}_{\mathrm{left}}} y_i + \sum_{T_i \in \mathcal{T}_{\mathrm{right}}} y_i \leq 10 \cdot |\mathcal{T}_k| + |\mathrm{OPT}_{\mathrm{top}}| + |\mathrm{OPT}_{\mathrm{left}}| + |\mathrm{OPT}_{\mathrm{right}}| \leq 13 \cdot |\mathcal{T}_k|$.

Combining this theorem with Lemma 1, we obtain the following theorem.

Theorem 2. *The integrality gap of* $\mathrm{LP}^{+}_{\mathrm{UFPP}}$ *for unweighted* UFPP *is constant.*

Finally, observe that if we define a task to be in $\mathcal{T}_{\mathrm{large}}$ if it is 3/4-large, then the resulting LP still has constant integrality gap by the same reasoning. In particular, then our added constraints would be a proper (polynomial size) subset of the (exponentially many) *rank constraints* introduced in [10]: for each edge e and for each subset $\mathcal{T}'$ of large tasks using e, there is a rank constraint bounding the maximum number of tasks in $\mathcal{T}'$ which can be in a feasible solution. In [10] it was left as an open question whether the integrality gap of the standard LP together with these constraints is $O(1)$, and an upper bound of $O(\log^2 n)$ was shown. Hence, we answered this question affirmatively for the unweighted case.

3 Embedding Dynamic Programs into Linear Programs

Let us start with a formal definition of a *standard* dynamic program $\mathcal{DP}$, which seems to capture most natural dynamic programs. For the sake of simplicity, let us focus on maximization problems, the case of minimization problems being symmetric. Consider some instance of the problem. This instance induces a polynomial-size set of possible *states* $\mathcal{S}$. The dynamic program fills in a table $t(\cdot)$, indexed by the states. There is a collection $\mathcal{S}_{\mathrm{base}} \subseteq \mathcal{S}$ of *base states*, whose profits can be computed with some trivial procedure (for example, they have profit zero). The remaining table entries $t(S)$, $S \notin \mathcal{S}_{\mathrm{base}}$, are filled in as follows. There is a set $\mathcal{C}_S$ of possible choices associated to S. We let $\mathcal{C} := \cup_{S \in \mathcal{S}, C \in \mathcal{C}_S} \{C\}$ be the set of all the possible choices. For notational convenience, we assume that choices of distinct states are distinct, and we let S^C denote the (only) state associated to C. Each choice $C \in \mathcal{C}_S$ is characterized by a profit w^C and by a collection of distinct states $S^C_1, \ldots, S^C_{k^C}$. If $\mathcal{C}_S = \emptyset$, we set $t(S) = -\infty$. Otherwise $t(S)$ is computed by exploiting the following type of recurrence, for proper coefficients $\alpha^C_i > 0$ (which can be assumed to be integral w.l.o.g.)[4]:

$$t(S) := \max_{C \in \mathcal{C}_S} \{w^C + \sum_{i=1}^{k^C} \alpha^C_i \cdot t(S^C_i)\}.$$

[4] Often in practice all α^C_i are 1 and k^C is a small number like 1 or 2.

Each state S_i^C must be a predecessor of S according to a proper partial order defined on the states (where the minimal states are the base ones). This partial order guarantees that $t(\cdot)$ can be filled in a bottom up fashion (without cycling). At the end of the process $t(S_{\text{start}})$ contains the value of the desired solution, for a proper special state S_{start}. By keeping track of the choices C which give the maximum in each recurrence, one also obtains the corresponding solution. For notational convenience we next assume that $t(S) = 0$ for every $S \in \mathcal{S}_{\text{base}}$. This can be enforced by introducing a dummy choice node C' with weight $t(S)$, and an associated dummy child state node S' with $t(S') = 0$. This way the profit can be expressed as the sum of the weights of the selected choices. We will use $\mathcal{I}$ to denote the input instance, excluding the part which is needed to define the weights w^C in the recurrences. In particular, $\mathcal{I}$ defines the states, the feasible choices for each state, the states associated to each choice, and the corresponding coefficients.

Next we describe an LP whose basic solutions describe the execution of $\mathcal{DP}$ on a given $\mathcal{I}$ for all the possible weights w^C. In particular, the weights will not appear in the set of constraints. Let us define a digraph $G = (V, E)$, with *state nodes* $\mathcal{S}$ and *choice nodes* $\mathcal{C}$. For every $C \in \mathcal{C}$, we add edges (S^C, C) and (C, S_i^C) for all $1 \leq i \leq k^C$. Observe that G is a DAG (i.e., there are no directed cycles) due to the partial order on the states. W.lo.g. we can assume that S_{start} has no ancestors. We let $\delta^{in}(v)$ and $\delta^{out}(v)$ denote the set of edges ending and starting at v, respectively. We associate a variable y_e to each edge e. The value of y_e in a fractional solution will be interpreted as a directed flow crossing e. For each state node $S \in S_{int} := \mathcal{S} - (\mathcal{S}_{\text{base}} \cup \{S_{\text{start}}\})$, we introduce a *flow conservation* constraint: $\sum_{\delta^{in}(S)} y_e = \sum_{\delta^{out}(S)} y_e$. We remark that it might be $\delta^{out}(S) = \emptyset$, in which case we assume that the corresponding sum has value zero. Recall that, in this case, $t(S) = -\infty$. We also force S_{start} to be the source of one unit of flow: $\sum_{\delta^{out}(S_{\text{start}})} y_e = 1$. This flow will end in nodes of $\mathcal{S}_{\text{base}}$. For each choice node $C \in \mathcal{C}$, we add a *flow duplication* constraint which guarantees that the flow entering C from its only ingoing edge $e = (S^C, C)$ is duplicated on all its outgoing edges according to the integral coefficients α_i^C: $x_{(C,S_i^C)} = \alpha_i^C \cdot x_{(S^C,C)}$ for all $1 \leq i \leq k^C$. We remark that, due to flow duplication, the flow entering a given node might be larger than 1. For a state node S this means that $t(S)$ contributes multiple times to the objective function. Altogether, the LP is defined as follows:

$$
\begin{aligned}
LP_{\mathcal{DP},\mathcal{I}} = \{\max \sum_{C \in \mathcal{C}} w^C \cdot y_{(S^C,C)} \text{ s.t. } & \sum_{\delta^{in}(S)} y_e = \sum_{\delta^{out}(S)} y_e, \forall S \in \mathcal{S}_{int}; \\
& \sum_{\delta^{out}(S_{\text{start}})} y_e = 1; \\
& y_{(C,S_i^C)} = \alpha_i^C \cdot y_{(S^C,C)}, \forall C \in \mathcal{C}, 1 \leq i \leq k^C; \\
& y_e \geq 0, \forall e \in E\}
\end{aligned}
$$

Let $CLP_{\mathcal{DP},\mathcal{I}} = CLP_{\mathcal{DP},\mathcal{I}}(y)$ be the set of constraints of $LP_{\mathcal{DP},\mathcal{I}}$. Let also $CH_{\mathcal{DP},\mathcal{I}}$ denote the collection of set of choices made by $\mathcal{DP}$ on any feasible input $\mathcal{I}$ for any possible choice of the weights w^C.

Theorem 3. (DP-embedding) *The vertices of $CLP_{\mathcal{DP},\mathcal{I}}$ are integral and in one to one correspondence with $CH_{\mathcal{DP},\mathcal{I}}$. Furthermore, $t(S_{start})$ is $-\infty$ iff $CLP_{\mathcal{DP},\mathcal{I}}$ is infeasible, and in all the other cases $t(S_{start})$ equals the optimal value of $LP_{\mathcal{DP},\mathcal{I}}$ (for a given choice of the weights).*

4 Constant Integrality Gap for Weighted UFPP

In this section we present an extended LP formulations for UFPP with constant integrality gap. For reasons of space, we present here a weaker LP with $O(1)$ integrality gap. For the claimed LP with integrality gap $7+\varepsilon$ we refer to the full version of this paper.

Recall that T_i denotes either a task or the corresponding rectangle. For brevity we call *large* (resp., *small*) the tasks that are 1/2-large (resp., 1/2-small), and denote the corresponding set by $\mathcal{T}_{\text{large}}$ (resp., $\mathcal{T}_{\text{small}}$). W.l.o.g., we can assume that $\mathcal{T}_{\text{large}}$ is given by the first n' tasks. We will crucially need the following two lemmas.

Lemma 6 ([5]). *Let $\mathcal{T}' \subseteq \mathcal{T}_{\text{large}}$ be a feasible solution to* UFPP. *There exists a partition of $\mathcal{T}'$ into 4 (disjoint) subsets, where each subset is an independent set of rectangles.*

Lemma 7 ([5]). *There is a dynamic program $\mathcal{DP}'$ which computes a maximum weight independent set of rectangles in $\mathcal{T}_{\text{large}}$.*

Maximum weight independent set of rectangles is a *subset problem*, where we are given a collection of n items $\{1,\ldots,n\}$ where item i has profit w_i, and we need to select a maximum profit subset of items satisfying given constraints. A solution to these problems can be defined as a binary vector $z = (z_1,\ldots,z_n) \in \{0,1\}^n$, where $z_i = 1$ iff item i (task T_i in our case) is selected. We remark that each choice of the dynamic program $\mathcal{DP}'$ from the previous lemma corresponds to selecting one or more items, and the structure of $\mathcal{DP}'$ guarantees that no item is selected more than once. Let $\mathcal{C}_i$ denote the choices of $\mathcal{DP}'$ that select item i. Consider the following LP:

$$EXT_{\mathcal{DP}',\mathcal{I}} := \{\max \sum_{i=1}^{n} w_i \cdot z_i \text{ s.t. } CLP_{\mathcal{DP}',\mathcal{I}}(y); z_i = \sum_{C \in \mathcal{C}_i} y_{(S^C,C)}, i = 1,\ldots,n\}.$$

By Theorem 3, if we project the basic solutions of $EXT_{\mathcal{DP}',\mathcal{I}}$ on variables z_i we obtain the set of solutions that $\mathcal{DP}'$ might compute on instance $\mathcal{I}$ for some choice of the weights w_i. In other terms, $EXT_{\mathcal{DP}',\mathcal{I}}$ is an integral extended LP formulation of the problem solved by $\mathcal{DP}'$ on instance $\mathcal{I}'$ for given weights. Also in this case we let $CEXT_{\mathcal{DP}',\mathcal{I}} = CEXT_{\mathcal{DP}',\mathcal{I}}(z)$ denote the set of constraints of $EXT_{\mathcal{DP}',\mathcal{I}}$.

We remark that there exists a choice of (possibly negative[5]) weights of the tasks that forces $\mathcal{DP}'$ to compute any given feasible solution $\mathcal{T}'$: we need this property for technical reasons.

We consider the following LP formulation for UFPP:

$$\begin{aligned}\mathrm{LP}^{+}_{\mathrm{UFPP}} := \{\max \sum_{i=1}^{n} w_i \cdot x_i \text{ s.t. } & CEXT_{\mathcal{DP}',\mathcal{T}_{\mathrm{large}}}(z^j), j = 1,2,3,4;\\ & x_i \leq z_i^1 + z_i^2 + z_i^3 + z_i^4, i = 1,\ldots,n';\\ & \sum_{T_i \in \mathcal{T}_e} d_i \cdot x_i \leq u_e, \forall e \in E;\\ & 0 \leq x_i \leq 1, i = 1,\ldots,n\}\end{aligned}$$

Let us argue that every feasible solution $\mathcal{T}'$ induces a feasible integral solution $(\tilde{x}, \tilde{z})$ (of the same profit) to $\mathrm{LP}^{+}_{\mathrm{UFPP}}$. Set $\tilde{x}_i = 1$ if $T_i \in \mathcal{T}'$, and $\tilde{x}_i = 0$ otherwise. Let $\mathcal{T}'_{\mathrm{large}} := \mathcal{T}' \cap \mathcal{T}_{\mathrm{large}}$, and $(\mathcal{T}^1, \mathcal{T}^2, \mathcal{T}^3, \mathcal{T}^4)$ be the partition of $\mathcal{T}'_{\mathrm{large}}$ given by Lemma 6. Fix $\tilde{z}_i^j = 1$ if $T_i \in \mathcal{T}^j$ and $\tilde{z}_i^j = 0$ otherwise. The resulting integral solution trivially satisfies the last three constraints. For the first constraint, we observe that $\mathcal{T}^j$ is a feasible independent set of rectangles: consequently, there is a choice of the weights that forces $\mathcal{DP}'$ to compute that solution. Thus $\tilde{z}^j$ must be a feasible (indeed basic) solution of $EXT_{\mathcal{DP}',\mathcal{T}_{\mathrm{large}}}(z^j)$.

Consider the standard linear program $\mathrm{LP}_{\mathrm{UFPP}}$. Even though $\mathrm{LP}_{\mathrm{UFPP}}$ has unbounded integrality gap in general, its integrality gap is bounded when there are only small tasks.

Lemma 8 ([10]). *Let $\delta > 0$. For instances of* UFPP *with only $(1-\delta)$-small tasks, the integrality gap of $\mathrm{LP}_{\mathrm{UFPP}}$ is bounded by $O(\log(1/\delta)/\delta^3)$.*

We are ready to bound the integrality gap of $\mathrm{LP}^{+}_{\mathrm{UFPP}}$.

Theorem 4. *The integrality gap of $\mathrm{LP}^{+}_{\mathrm{UFPP}}$ is in $O(1)$.*

We can strengthen the linear program presented here even further such that its integrality gap is bounded by $7+\epsilon$, matching the ratio of the best known approximation algorithm for UFPP [5]. The latter algorithm works as follows: for the 1/2-large tasks it uses the DP described in the previous section. For δ-small tasks (for a sufficiently small value of δ depending on ϵ) it uses LP-based methods together with a framework to combine solutions for suitable subproblems. (In fact, already in [11, Corollary 3.4] it was shown that in that setting $\mathrm{LP}_{\mathrm{UFPP}}$ has an integrality gap of only $1+\epsilon$, if δ is sufficiently small.) For the remaining tasks, that is, tasks that are δ-large but 1/2-small, the algorithm in [5] employs $O(n)$ dynamic programs for suitably chosen subproblems. We can embed these dynamic programs into $\mathrm{LP}^{+}_{\mathrm{UFPP}}$. Using similar ideas as for the $(7+\epsilon)$-approximation algorithm in [5] one can show that the resulting LP has an integrality gap of at most $7+\epsilon$. We leave the details to the full version of this paper.

[5] Non-negative weights are sufficient, provided that ties in the computation of the maxima are broken in a proper way.

Theorem 5. *For every $\epsilon > 0$ there is a linear programming formulation of UFPP with an integrality gap of at most $7 + \epsilon$ whose complexity is bounded by a polynomial in the input.*

References

1. N. Bansal, A. Chakrabarti, A. Epstein, and B. Schieber. A quasi-PTAS for unsplittable flow on line graphs. In *STOC*, pages 721–729, 2006.
2. N. Bansal, Z. Friggstad, R. Khandekar, and R. Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. In *SODA*, pages 702–709, 2009.
3. A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. In *STOC*, pages 735–744, 2000.
4. P. Berman, B. DasGupta, S. Muthukrishnan, and S. Ramaswami. Improved approximation algorithms for rectangle tiling and packing. In *SODA*, pages 427–436, 2001.
5. P. Bonsma, J. Schulz, and A. Wiese. A constant factor approximation algorithm for unsplittable flow on paths. In *FOCS*, pages 47–56, 2011.
6. G. Calinescu, A. Chakrabarti, H. Karloff, and Y. Rabani. Improved approximation algorithms for resource allocation. In *IPCO*, pages 401–414, 2002.
7. A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar. Approximation algorithms for the unsplittable flow problem. *Algorithmica*, pages 53–78, 2007.
8. P. Chalermsook and J. Chuzhoy. Maximum independent set of rectangles. In *SODA*, pages 892–901, 2009.
9. C. Chekuri, A. Ene, and N. Korula. Unsplittable flow in paths and trees and column-restricted packing integer programs. Unpublished. Available at http://web.engr.illinois.edu/ ene1/papers/ufp-full.pdf.
10. C. Chekuri, A. Ene, and N. Korula. Unsplittable flow in paths and trees and column-restricted packing integer programs. In *APPROX*, pages 42–55, 2009.
11. C. Chekuri, M. Mydlarz, and F. Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Trans. on Algorithms*, 3, 2007.
12. M. Chrobak, G. J. Woeginger, K. Makino, and H. Xu. Caching is hard: even in the fault model. In *ESA*, pages 195–206, 2010.
13. A. Darmann, U. Pferschy, and J. Schauer. Resource allocation with time intervals. *Theor. Comp. Sc.*, 411:4217–4234, 2010.
14. 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.*, 15:100–109, 1984.
15. S. Khanna, S. Muthukrishnan, and M. Paterson. On approximating rectangle tiling and packing. In *SODA*, pages 384–393, 1998.
16. R. Kipp Martin, Ronald L. Rardin, and Brian A. Campbell. Polyhedral characterization of discrete dynamic programming. *Oper. Res.*, 38(1):127–138, 1990.
17. A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency.* Springer, Berlin, 2003.