A logarithmic approximation for unsplittable flow on line graphs

Nikhil Bansal*   Zachary Friggstad†   Rohit Khandekar*   Mohammad R. Salavatipour†

Abstract

We consider the unsplittable flow problem on a line. In this problem, we are given a set of n tasks, each specified by a start time s_i, an end time t_i, a demand $d_i > 0$, and a profit $p_i > 0$. A task, if accepted, requires d_i units of "bandwidth" from time s_i to t_i and accrues a profit of p_i. For every time t, we are also specified the available bandwidth c_t, and the goal is to find a subset of tasks with maximum profit subject to the bandwidth constraints.

In this paper, we present the first polynomial-time $O(\log n)$-approximation algorithm for this problem. No polynomial-time $o(n)$-approximation was known prior to this work. Previous results for this problem were known only in more restrictive settings, in particular, either if the given instance satisfies the so-called "no-bottleneck" assumption: $\max_i d_i \leq \min_t c_t$, or else if the ratio of the maximum to the minimum demands and ratio of the maximum to the minimum capacities are polynomially (or quasi-polynomially) bounded in n. Our result, on the other hand, does not require any of these assumptions.

Our algorithm is based on a combination of dynamic programming and rounding a natural linear programming relaxation for the problem. While there is an $\Omega(n)$ integrality gap known for this LP relaxation, our key idea is to exploit certain structural properties of the problem to show that instances that are bad for the LP can in fact be handled using dynamic programming.

1 Introduction

In the Unsplittable Flow Problem (UFP), we are given an undirected graph $G = (V, E)$ with edge capacities $\{c_e\}_{e \in E}$, a set of n pairs of vertices called demand pairs $T = \{(s_i, t_i)\}_{1 \leq i \leq n}$ where each pair s_i, t_i has a demand value $d_i > 0$ and profit $p_i > 0$. We obtain a profit of p_i if we can route the total demand d_i from s_i to t_i along a *single* path. A subset $S \subseteq \{1, \ldots, n\}$ of the demands is called feasible if all the demands in S can be routed simultaneously without violating any edge capacity, i.e., the total demand flow on each edge e is at most c_e. The goal is to find a feasible set of demand pairs and paths to route the corresponding demands while maximizing the total profit obtained from the demand pairs that are fully routed. The UFP is NP-hard even when restricted

to very special cases. For instance, if the entire graph G is a single edge, the UFP specializes to the KNAPSACK problem. When all the edge capacities as well as all the demands and profits are 1, the UFP specializes to the well-studied maximum edge-disjoint paths problem (EDP). This special case is NP-hard even for restricted classes of graphs, like planar graphs.

There is a large amount of research focused on the study of UFP on line graphs.[1] In such an instance, the input graph G is an undirected path (line). The study of UFP on line graphs is motivated by several applications such as bandwidth allocation of sessions on a shared communication link, job scheduling with known machine requirements and time windows, the general caching problem with varying page sizes and available memory and so on. In fact, UFP on line graphs can be thought of as a scheduling problem called the Resource Allocation Problem (or RAP for short). In this problem, we are given n tasks, each specified by a start time s_i, an end time t_i, a demand d_i, and a profit p_i. The task i, if scheduled, requires d_i units of a resource in the time interval $[s_i, t_i)$, called the *span* of i, and is assumed to accrue a profit of p_i. The resource (e.g., CPU), which is shared among scheduled tasks, is present to an extent c_t at time t. We refer to c_t as the capacity at time t. The problem is to find a subset S of the tasks such that $\sum_{i \in S} p_i$ is maximized while satisfying the resource capacity constraints at all times. It is easy to see the correspondence between the tasks in RAP and demand pairs in UFP on line graphs. This problem has also been studied under names such as "Bandwidth Allocation", "Resource Constrained Scheduling", and "Call Admission Control".

UFP continues to be a difficult problem even when restricted to line graphs and obtaining a reasonable approximation for it has resisted several attempts. No non-trivial approximation for this problem is known previously without any extra assumptions on the parameters of the input. One difficulty is that the natural

*IBM T.J.Watson research center. email: {nikhil,rohitk}@us.ibm.com.

†Department of Computing Science, University of Alberta, Edmonton, Alberta T6G 2E8, Canada. email: {zacharyf,mreza}@cs.ualberta.ca. The second author was supported by NSERC and iCore scholarships and the fourth author was supported by NSERC and an Alberta Ingenuity New Faculty award.

[1] We use the term "line graphs" to refer to graphs that consist of a simple path, following the previous works on UFP. This term may not be confused with a more standard notion of line graphs in graph theory, i.e., a graph obtained from another graph by replacing edges by vertices and making two vertices adjacent if the corresponding edges are incident.

LP relaxation for this problem has an integrality gap of $\Omega(n)$ and obtaining an approximation algorithm with performance ratio $o(n)$ has been an interesting open question. As we discuss below, all previous results require extra assumptions. The most widely used assumption is the so called no-bottleneck assumption, which states that, $\max_i d_i \leq \min_e c_e$. This requires that the demand of every task be no more than the capacity of every edge (and not just those edges that this task spans). The no-bottleneck assumption imposes a rather strong restriction on the instances, and seems to exclude the truly hard cases of the problem. For example, the integrality gap instance mentioned above does not satisfy this assumption.

1.1 Previous work. As stated above, when all the demands, capacities, and profits are one, we obtain a well-studied problem of EDP. This problem is NP-hard in general undirected graphs (with non-constant number of terminal-pairs), and NP-hard even with only two terminal-pairs in directed graphs (see Fortune et al. [11]). Kleinberg [15] proved that EDP is approximable within a factor of $O(\sqrt{|E|})$. This was later generalized to UFP by Srinivasan [19] and Baveja and Srinivasan [6] under no-bottleneck assumption: $\max_i d_i \leq \min_e c_e$. More recently, Chekuri et al. [9] improved these results to $O(\sqrt{|V|})$-approximation. On the other hand, Guruswami et al. [14] proved that EDP on directed graphs is NP-hard to approximate within a factor of $\Omega(|E|^{\frac{1}{2}-\epsilon})$ for any constant $\epsilon > 0$. In the undirected setting, Andrews et al. [1, 2] showed that the problem is quasi-NP-hard to approximate within $\Omega(\log^{\frac{1}{2}-\epsilon}|E|)$ for any $\epsilon > 0$. All these results give the same hardness for UFP even with the no-bottleneck assumption in the corresponding model. Azar and Regev [3] proved that without the no-bottleneck assumption, UFP is NP-hard to approximate within a factor of $\Omega(|E|^{1-\epsilon})$. Garg et al. [13] proved that UFP is APX-hard on trees (and even on stars with unit demands).

Several papers have studied UFP and EDP on graphs with high expansion. For instance, Frieze [12] proved that for (large) constant-degree regular expanders with sufficiently high expansion, there is a constant c such that any $cn/\log n$ pairs, such that no vertex appears in more than $O(1)$ pairs, can be connected via edge-disjoint paths. This implies an $O(\log n)$-approximation for EDP on such expanders. Using earlier works by Kleinberg and Rubinfeld [16], Srinivasan [19] gave an $O(\log^3 n)$-approximation for uniform capacity UFP (referred to as UCUFP) on expanders. Some improvements were obtained by Kolman and Scheideler [17] and Chakrabarti et al. [8].

The special case of the EDP problem on line graphs corresponds to maximum independent set on interval graphs, which can be solved in polynomial time. If we have uniform capacities (i.e., UCUFP), then the problem is NP-hard even on line graphs. This problem is equivalent to a resource allocation problem that has been studied by Bar-Noy et al. [5] and Phillips et al. [18]. The first constant approximation algorithm for UCUFP on line graphs was provided by [18]. The approximation ratio was later improved in a series of papers [5, 7] to $(2 + \epsilon)$.

For the general UFP on line graphs, as mentioned earlier, the problem has not been easy to approximate. Many of the previous works have simplified the problem by making some extra assumptions in order to get reasonable approximations. For UFP on line graphs, with the no-bottleneck assumption, Chakrabarti et al. [8] presented the first constant approximation which was later improved by Chekuri et al. [10] to a $(2 + \epsilon)$-approximation (again under the no-bottleneck assumption). Bansal et al. [4] proved that if all the demands, edge-capacities, and profits are quasi-polynomial in the number of pairs, i.e., at most $O(2^{\text{polylog}(n)})$, then there is a $(1 + \epsilon)$-approximation algorithm that runs in quasi-polynomial time. Chakrabarti et al. [8] also proved that the integrality gap of the natural LP relaxation of the UFP on line graphs is $\Omega(\log(\frac{\max_i d_i}{\min_i d_i}))$ which is $\Omega(n)$ in their example.

1.2 Our result and techniques. In this paper we study the UFP on line graphs, or equivalently, the Resource Allocation Problem (RAP). We present an $O(\log n)$-approximation for this problem *without* any extra assumptions, thus beating the integrality gap for the natural LP relaxation. This also implies an $O(\log n)$-approximation for UFP when the underlying graph is a cycle, also called ring graphs.

The following is a natural LP relaxation of the problem. We associate a variable x_i to denote if task i is picked in the solution.

$$
\begin{array}{llrcll}
(\text{LP}) & \max & \sum_i p_i x_i & & & \\
& \text{s.t.} & \sum_{i: t \in [s_i, t_i)} d_i x_i & \leq & c_t, & 1 \leq t \leq T \\
& & x_i & \in & [0, 1], & 1 \leq i \leq n
\end{array}
$$

It is instructive to consider the following $\Omega(n)$ integrality gap example, that we refer to as the *staircase instance*. This example first seems to have been observed by Chakrabarti et al. [8]. We have n tasks and task i has start time $s_i = 0$ and finish time $t_i = i$, i.e., span $[0, i)$, and $d_i = 2^{-i}$. All the tasks have profit 1 and the capacity c_t during interval $[t-1, t)$ is equal to 2^{-t}, for $1 \leq t \leq n$ (see Figure 1).

Now consider the fractional solution in which $x_i = 1/2$ for all tasks i. It is easy to see that it is feasible
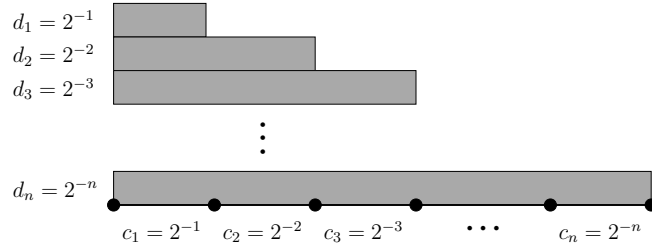
Figure 1: The example showing an integrality gap of $\Omega(n)$. The demands are not drawn to scale as they decrease exponentially.

for the LP and accrues a profit of $n/2 = \Omega(n)$. On the other hand, we claim that any integral solution can have profit of at most 1. To see this, let d_{j^*} be the demand (task) with the smallest index that is selected in the solution. Then this demand saturates time $[j^* - 1, j^*)$ (recall that $d_{j^*} = 2^{-j^*} = c_t$ for $t \in [j^* - 1, j^*)$). So no other task with index $j' > j^*$ can be selected. Note that this example does *not* satisfy the no-bottleneck property that $\max_i d_i \leq \min_t c_t$ and that the demands (and capacities) are exponentially large in n. Therefore, in a sense, the extra assumptions used in earlier works [8, 10, 4] to obtain a constant ratio approximation for UFP on line graphs may actually be excluding the truly hard cases of the problem.

The starting point for our results is the observation that even though the staircase-like instances described above are bad for the LP, they can be well approximated using dynamic programming. In particular, we show that any instance can essentially be decomposed into two parts. The first part can be solved well using LP relaxation, and the second part can be solved well using dynamic programming. The overall algorithm simply chooses the best of these two solutions. The second part requires us to identify some key structural properties such as being "intersecting" and "laminar", that make the instance amenable to dynamic programming.

Outline of our algorithm. Our algorithm has the following steps. First, we show (by a simple argument) that at the loss of an $O(\log n)$ factor, the problem can be reduced to instances where all requests intersect at a common time. We then describe an $O(1)$-approximation for such *intersecting* instances. To do the latter, we partition the tasks into *slack* tasks and *tight* tasks. A task is called *slack* if its demand is a small fraction of the minimum capacity available during its span. A task that is not slack is called *tight*. We show that if all the tasks are slack and intersecting, then there is a randomized rounding based $O(1)$-approximation, building on the ideas of Calinescu et al. [7]. For tight task instances, we show that requiring that each task be tight and the

instance be intersecting, imposes a lot of structure on the instance. Handling tight task instances is perhaps the most interesting contribution of this paper. These instances seem to capture most of the inherent hardness of the problem. For example, note that the staircase instance above satisfies both the intersecting property and each task there is tight.

2 Reduction to Intersecting Instances

We start with some notation. For a task i, we define the span as $\text{span}(i) = [s_i, t_i)$ and the length as $\text{length}(i) = t_i - s_i$. We can assume that for each task i, the start and end times s_i and t_i are integers in the range $\{1, \ldots, 2n\}$, since this leaves the problem combinatorially unchanged. Each edge corresponds to some interval $[i, i + 1)$. Thus task i, spans the edges $[s_i, s_i + 1), \ldots, [t_i - 1, t_i)$. We say that tasks i and j intersect if $\text{span}(i) \cap \text{span}(j) \neq \emptyset$. We call a subset S of tasks "admissible" if it satisfies the capacity constraints: $\sum_{i \in S : t \in \text{span}(i)} d_i \leq c_t$ for all t. Without loss of generality, we assume that each task (by itself) is admissible. Thus, any algorithm can trivially obtain a profit of $p_{\max} = \max_i p_i$. If we are willing to lose a factor of $(1 - \epsilon)$ in the approximation ratio, we can assume that each $p_i \geq \epsilon p_{\max}/n$, by discarding tasks with lower profit if necessary. Moreover, the profits can be assumed to be integers.

We call an instance of RAP "intersecting" if all the tasks intersect at a single time, i.e., there exists a time t_{mode} such that $t_{\text{mode}} \in \text{span}(i)$ for all tasks i. The following lemma shows that RAP can be reduced to the intersecting instances of RAP with a loss of $O(\log n)$ factor in the approximation.

LEMMA 2.1. *If there is a ρ-approximation for the intersecting instances of RAP, then there is an $O(\rho \cdot \log n)$-approximation for RAP.*

Proof. Consider a general instance of RAP. We first partition the tasks into groups according to their lengths $\text{length}(i)$. We say that a task i belongs to group r if $2^r \leq$

length$(i) < 2^{r+1}$. Clearly there are at most $\lceil \log(2n) \rceil$ groups $\{0, 1, \ldots, \lceil \log(2n) \rceil - 1\}$. Here the logarithm is taken to the base 2. Our algorithm computes an $O(\rho)$-approximation for every group taken one at a time, as described below, and outputs the maximum profit solution out of these $\lceil \log(2n) \rceil$ solutions; thereby yielding an overall $O(\rho \cdot \log n)$ approximation. It is, thus, enough to show how to obtain an $O(\rho)$-approximation for a single group.

Now fix some group r. For an integer j, a task i in group r is said to belong to part j if it spans time $j \cdot 2^r$. Since $2^r \leq$ length$(i) < 2^{r+1}$ for every task i in group r, each such task belongs to some part j for some integer j (if it spans two such times, we assign it to both parts). Moreover, a task in part j cannot intersect any task that belongs to part $j + 4$. Now, for $a = 0, 1, 2, 3$, let S_a denote the set of group r tasks in parts $j \equiv a \mod 4$, respectively. Each S_a is a disjoint union of intersecting instances of RAP, and moreover our algorithm can compute these instances.

Given a ρ-approximation algorithm for intersecting instances, our algorithm computes ρ-approximation to each S_a by taking a union of ρ-approximations of parts j for each $j \equiv a \mod 4$. Finally, it outputs the maximum profit solution among the solutions for S_0, S_1, S_2, and S_3 for all possible values of r. It is easy to see that this algorithm is a 4ρ-approximation for group r. Thus the proof is complete. ∎

3 Algorithm for Intersecting Instances

Given Lemma 2.1, we now present a constant factor approximation for intersecting instances of RAP. Recall that for an intersecting instance of RAP, there exists a time t_{mode} such that $t_{\text{mode}} \in$ span(i) for all tasks i. The reason we call this time t_{mode} will be clear from the following lemma.

LEMMA 3.1. *Without loss of generality, we can assume that the capacity profile $\{c_t \mid 1 \leq t \leq 2n\}$ is "unimodal", i.e., $c_t \leq c_{t+1}$ for all $1 \leq t < t_{\text{mode}}$ and $c_t \geq c_{t+1}$ for all $t_{\text{mode}} \leq t < 2n$.*

Proof. Consider any $t \in [1, t_{\text{mode}})$. Note that if task i satisfies $t \in [s_i, t_i)$, then it also satisfies $t + 1 \in [s_i, t_i)$. Therefore setting $c_t := \min\{c_t, c_{t+1}\}$ does not change the set of feasible solutions. Thus we can assume $c_t \leq c_{t+1}$ without loss of generality. A similar argument also works for $t \in [t_{\text{mode}}, 2n)$. ∎

3.1 The laminar instances. We call an instance of RAP "laminar" if the tasks can be ordered $i = 1, \ldots, n$ such that span$(i+1) \subseteq$ span(i) for all $1 \leq i < n$. Before dealing with the general intersecting instances, we first observe that there is an FPTAS for laminar instances.

LEMMA 3.2. *There is an FPTAS for the laminar instances of RAP. For any $\epsilon > 0$, the algorithm obtains a $(1 + \epsilon)$-approximation in time $O(n^2/\epsilon^2 \cdot \log(nP))$ if the profits are integers in the range $[1, P]$.*

Proof. The algorithm is based on dynamic programming similar to the one used for knapsack problems. Given integers i and p, let $D(i, p)$ denote the minimum total demand of an *admissible* subset $S \subseteq \{1, \ldots, i\}$ of the first i tasks such that $\sum_{j \in S} p_j \geq p$. We use the convention $D(i, p) = \infty$ if no such S exists. The values $D(i, p)$ are computed in the order of increasing i. For $i = 1$, we set $D(1, p) = 0$ if $p \leq 0$; or $D(1, p) = d_1$ if $p_1 \geq p$ (recall that $\{1\}$ is admissible); or ∞ otherwise. Now we use the following recurrence:
If $D(i, p - p_{i+1}) + d_{i+1} \leq \min_{t \in \text{span}(i+1)} c_t$, then

$$D(i + 1, p) = \min\{D(i, p), D(i, p - p_{i+1}) + d_{i+1}\},$$

else

$$D(i + 1, p) = D(i, p).$$

The correctness follows as the tasks are laminar, and hence the decision whether task $i + 1$ can be admitted to set $S \subseteq \{1, \ldots, i\}$ or not, depends only on the total demand of S.

The above dynamic program computes the optimum solution in time $O(n \cdot nP)$. Moreover, if we know the value OPT of the optimum solution, the above dynamic program runs in time $O(n \cdot \text{OPT})$. To obtain a $(1 + \epsilon)$ approximation, we can make the running time polynomial in n, $\log P$, and $1/\epsilon$ as follows. We guess the value of OPT in the range $[1, nP]$ within a factor of $(1+\epsilon)$, remove all the tasks with profit less than ϵOPT$/n$, and then round the profits p_i to $\lfloor np_i/(\epsilon$OPT$) \rfloor$. After the rounding, the optimum value is $\lfloor n/\epsilon \rfloor$, and hence the above algorithm computes an $(1 + \epsilon)$-approximation in time $O(n \cdot n/\epsilon)$. Since the algorithm iterates over $O(1/\epsilon \cdot \log(nP))$ guesses of OPT, the overall running time is $O(n^2/\epsilon^2 \cdot \log(nP))$. ∎

3.2 The general intersecting instances. We now consider the general intersecting instances. We partition the tasks into four disjoint types as follows. Let $\epsilon > 0$ be a constant to be fixed later.

1. A task i is called "slack" if $d_i \leq \epsilon \cdot c_t$ for all $t \in [s_i, t_i)$.

2. A task i is called "left-tight" if $d_i > \epsilon \cdot c_t$ for some time $t \in [s_i, t_{\text{mode}})$ and $d_i \leq \epsilon \cdot c_t$ for all $t \in [t_{\text{mode}}, t_i)$.

3. A task i is called "right-tight" if $d_i > \epsilon \cdot c_t$ for some time $t \in [t_{\text{mode}}, t_i)$ and $d_i \leq \epsilon \cdot c_t$ for all $t \in [s_i, t_{\text{mode}})$.

4. A task i is called "tight" if it is not slack, left-tight, or right-tight.

In the following sections, we show how to obtain a constant factor approximation for the case when all the tasks belong to a single type. Clearly, given a general intersecting instance, if we partition the set of tasks into these four types and find a constant factor approximation for each group separately, the maximum profit solution among these four solutions gives a constant approximation for the given instance.

3.2.1 Slack tasks.
For the case of slack tasks, we give an LP based $O(1)$-approximation. Our algorithm is an adaptation of the randomized rounding algorithm of Calinescu et al. [7]. In particular, we need to adapt their algorithm to work for unimodal capacity profiles. We lose an additional factor 2 in the process.[2]

Our algorithm begins by solving the LP relaxation (LP) described in Section 1.2. Let x_i^* be some optimum LP solution. By scaling if necessary, we assume that the smallest capacity is 1. We partition the tasks into two sets: $C_\leq$ be the set of tasks i such that $c_{s_i} \leq c_{(t_i-1)}$; and $C_>$ be the set of tasks i such that $c_{s_i} > c_{(t_i-1)}$. In other words, $C_\leq$ is the set of tasks i for which the capacity at the starting time s_i is at most that at the last time $t_i - 1$ at which task i is active. Clearly, at least one of the sets $C_\leq$ or $C_>$ accrues at least half of the fractional profit, i.e., either (1) $\sum_{i \in C_\leq} p_i x_i^* \geq \frac{1}{2} \sum_i p_i x_i^*$, or (2) $\sum_{i \in C_>} p_i x_i^* \geq \frac{1}{2} \sum_i p_i x_i^*$.

Let us assume that case (1) holds. Below we present how to round the fractional solution for $C_\leq$ to get an admissible integral solution of almost equal cost. An analogous argument holds for the other case (2); and is omitted.

The rounding algorithm proceeds as follows. We ignore the tasks in $C_>$ and order the tasks in $C_\leq$ in the increasing order of their starting times. Let $\delta = \epsilon + \epsilon^{1/4}$. We choose each task $i \in C_\leq$ independently with probability $(1-\delta)x_i^*$. Let R denote the set of chosen tasks. Let these tasks in R be $i_1 < i_2 < \cdots < i_{|R|}$. We construct a sequence of sets $\emptyset = S_0, S_1, \ldots, S_{|R|}$ as follows: let $S_r = S_{r-1} \cup \{i_r\}$ if $S_{r-1} \cup \{i_r\}$ is admissible; or let $S_r = S_{r-1}$ otherwise. The algorithm outputs the set $S = S_{|R|}$.

Note that S is a random set, and the event whether task i lies in S is correlated with the events whether other tasks lie in S. The following theorem states the probability with which a particular task lies in S.

THEOREM 3.1. *For any request $i \in C_\leq$, it lies in S with probability at least $(1 - \sqrt{\epsilon}) \cdot (1 - \epsilon - \epsilon^{1/4})x_i^*$.*

Proof. Define the following random variables: for $i \in C_\leq$, let $X_i = 1$ if $i \in R$, and 0 otherwise; and let $Y_i = 1$ if $i \in S$, and 0 otherwise. Note that X_i's are independent, but Y_i's are not.

Fix $1 \leq r \leq |R|$ and consider the task $i = i_r$. We are interested in $\mathrm{E}[Y_i]$. Since $S \subseteq R$, we have $Y_i \leq X_i$ and hence $\mathrm{E}[Y_i] \leq \mathrm{E}[X_i]$. Consider the event E_r that $[Y_i = 0 \mid X_i = 1]$. If E_r happens, then it must be the case that $S_{r-1} \cup \{i\}$ is not admissible. The lemma below characterizes the reason E_r happens.

LEMMA 3.3. *The event E_r holds if and only if the capacity constraint at the start time s_i of task i is violated by the set of tasks $S_{r-1} \cup \{i\}$.*

Proof. The proof is based on the fact that the capacity profile is unimodal with the maximum capacity at time t_{mode} and the defining property of tasks in $C_\leq$. By definition, E_r happens if and only if the capacity constraint at some time $t \in [s_i, t_i)$ is violated by $S_{r-1} \cup \{i\}$. If $t \leq t_{\mathrm{mode}}$, then from the assumption that capacity profile is unimodal, we have $c_{s_i} \leq c_t$. If $t > t_{\mathrm{mode}}$, then since $i \in C_\leq$, we have $c_{s_i} \leq c_{t_i-1} \leq c_t$. Since all the tasks in $S_{r-1} \cup \{i_r\}$ cross s_i and may or may not cross t, we get that $S_{r-1} \cup \{i_r\}$ must violate the capacity constraint at s_i. ∎

Thus, for E_r to hold, the total demand of tasks in $R \cap \{1, \ldots, i-1\}$ must exceed $c_{s_i} - d_i$. We now use Chebyshev's inequality to bound $\mathrm{PR}[E_r]$. For $j = 1, \ldots, i-1$, consider a random variable $D_j = d_j$ if $j \in R$, and 0 otherwise. Let $D = \sum_{j=1}^{i-1} D_j$.

LEMMA 3.4. $\mathrm{PR}[E_r] \leq \sqrt{\epsilon}$.

Proof. We shall show that $\mathrm{PR}[E_r] \leq \mathrm{PR}[D > c_{s_i} - d_i] \leq \mathrm{PR}[D > c_{s_i} - \epsilon c_{s_i}] \leq \sqrt{\epsilon}$. The first inequality follows from the algorithm and the discussion above, the next inequality follows since all tasks are slack. Below we prove the last inequality.

We have $\mathrm{E}[D] = \sum_{j=1}^{i-1} \mathrm{E}[D_j] = \sum_{j=1}^{i-1} d_j(1-\delta)x_j^* \leq (1-\delta)c_{s_i}$. The last inequality holds since the fractional solution x^* satisfies the capacity constraint at time s_i. Since D_j's are independent,

$$\mathrm{VAR}[D] = \sum_{j=1}^{i-1} \mathrm{VAR}[D_j] \leq \sum_{j=1}^{i-1} \mathrm{E}[D_j^2] = \sum_{j=1}^{i-1} d_j^2 \cdot (1-\delta)x_j^*$$

$$\leq \epsilon c_{s_i} \sum_{j=1}^{i-1} d_j \cdot (1-\delta)x_j^* \leq \epsilon(1-\delta)c_{s_i}^2.$$

---

[2]The original argument of [7] holds only for non-decreasing capacity profiles.

The second last inequality follows since all the tasks are slack. Now we use Chebyshev's inequality:

$$Pr\left[D - \mathrm{E}[D] \geq t\sqrt{\mathrm{VAR}[D]}\right] \leq \frac{1}{t^2}$$

for $t > 0$. Putting $t = \epsilon^{-1/4}$, we get

$$\mathrm{PR}[D > (1-\delta)c_{s_i} + \epsilon^{-1/4} \cdot \sqrt{\epsilon(1-\delta)c_{s_i}^2}] \leq \sqrt{\epsilon}.$$

That is, $\mathrm{PR}[D > c_{s_i} \cdot (1 - \delta + \epsilon^{1/4}\sqrt{1-\delta})] \leq \sqrt{\epsilon}$. Now substituting $\delta = \epsilon + \epsilon^{1/4}$, we get that $\mathrm{PR}[D > c_{s_i} \cdot (1 - \epsilon)] \leq \sqrt{\epsilon}$ as desired. ∎

Now,

$$
\begin{aligned}
\mathrm{E}[Y_i] &= \mathrm{PR}[Y_i = 1 \mid X_i = 1] \cdot \mathrm{PR}[X_i = 1] \\
&\quad + \mathrm{PR}[Y_i = 1 \mid X_i = 0] \cdot \mathrm{PR}[X_i = 0] \\
&= \mathrm{PR}[Y_i = 1 \mid X_i = 1] \cdot \mathrm{PR}[X_i = 1] \\
&= (1 - \mathrm{PR}[E_r]) \cdot (1 - \delta)x_i^* \\
&\geq (1 - \sqrt{\epsilon}) \cdot (1 - \epsilon - \epsilon^{1/4})x_i^*.
\end{aligned}
$$

This completes the proof of Theorem 3.1. ∎

If z^* is the value of the LP solution for the slack tasks, using Theorem 3.1, the expected value of the solution obtained by the algorithm is at least $\frac{1}{2}(1 - \sqrt{\epsilon}) \cdot (1 - \epsilon - \epsilon^{1/4})z^*$. The algorithm and its analysis above only uses second moments and hence can be easily derandomized by using pairwise independent family of random variables.

3.2.2 Tight tasks. Recall that all tasks are admissible by themselves. We partition them into classes based on the demands. A task i belongs to class r if $2^r \leq d_i < 2^{r+1}$. Let $k = \lceil \log(1/\epsilon) \rceil + 1$ and let C_a for $a = 0, 1, \ldots, k-1$ be the collection of tasks of class $r \equiv a \mod k$. It suffices to design a constant factor approximation for any fixed collection C_a. The final algorithm can simply apply this algorithm to each of these collections and choose the best solution, thereby losing a factor of at most k in the approximation.

Fix $a \in \{0, 1, \ldots, k-1\}$. It is easy to see that if $i, j \in C_a$ such that i is of a higher class than j, then $d_j < \epsilon \cdot d_i$. We now argue that $\mathrm{span}(i) \subseteq \mathrm{span}(j)$. Suppose, on the contrary, that this does not hold and that without loss of generality, there exists $t \leq t_{\mathrm{mode}}$ such that $t \in \mathrm{span}(i) \setminus \mathrm{span}(j)$. Since the capacity profile is unimodal, we have $d_j < \epsilon d_i \leq \epsilon c_t \leq \epsilon c_{t'}$ for all $t' \in [s_j, t_{\mathrm{mode}})$. This contradicts the fact that task j is tight and not right-tight.

We next argue that the optimum algorithm can pick at most $2/\epsilon$ tasks from any class. Suppose, on the contrary, that the optimum picks more than $2/\epsilon$ tasks

from some class. Consider the task i among these with largest start time s_i. Since each job in the class of i has demand at least $d_i/2$, it must be that $d_i/2 \cdot 2/\epsilon < c_t$ for all $t \in [s_i, t_{\mathrm{mode}})$. However this contradicts the fact that task i is tight and not right-tight.

We will use the fact that optimum picks at most $2/\epsilon$ tasks from any class in our dynamic program. Our dynamic program is similar to the one presented for the laminar case in Section 3.1 and computes a maximum profit set of tasks among the admissible sets which pick at most $2/\epsilon$ tasks from each class. We then use the standard trick used before to transform this pseudo-polynomial algorithm into a PTAS. The algorithm is essentially the same as for the laminar case, except that at each step, instead of considering the possibility of adding one task to the subproblem, we consider the possibility of adding a subset of size at most $2/\epsilon$ of the tasks in a class. More specifically, we define $D(r, p)$, for $r \equiv a \mod k$, as the minimum total demand of an admissible subset S of tasks of class at most r in C_a such that $\sum_{j \in S} p_j \geq p$ with the condition that no more than $2/\epsilon$ tasks are selected from each class. We populate the table in the increasing order of class r. It is easy to set the initial values for the smallest value of $r \equiv a \mod k$. For computing the remainder of the table, we use the following recurrence. For any subset Q of size at most $2/\epsilon$ of tasks in class $r + k$, let $P_Q = \sum_{j \in Q} p_j$ be the total profit of the tasks in Q and for each time t in the span of some task in Q, let $D_Q(t) = \sum_{j \in Q: t \in \mathrm{span}(j)} d_j$ be the total demand in Q which spans t. We use $D_Q = \sum_{j \in Q} d_j$ to denote the total demands of all the tasks in Q. Note that the tasks in Q form a demand profile, so the total demand $D_Q(t)$ of the tasks in Q over a time t may be different for different times. We compute $D(r + k, p)$ as follows:

$$(3.1) \qquad D(r + k, p) = \min_Q \left(D(r, p - P_Q) + D_Q\right)$$

where the minimum is taken over all subsets Q (including the empty subset) of tasks from class $r + k$ such that

- $|Q| \leq 2/\epsilon$,

- for every time t in the span of some task in Q, the capacity constraints are satisfied: $D(r, p - P_Q) + D_Q(t) \leq c_t$.

Note that each such entry can be computed in time that is exponential in $2/\epsilon$ but polynomial in n. Also, note that here we crucially use the fact that the span of every demand in a class $r + k$ is contained in the span of every demand in any class less than $r + k$.

We can easily transform this into a PTAS as in

the case of laminar instance. Overall we obtain a $(\lceil \log(1/\epsilon)) \rceil + 1)$-approximation for the tight tasks.

3.2.3 Left-tight or right-tight tasks.
We now describe our algorithm for the left-tight tasks. Our algorithm for the right-tight tasks is similar and is omitted. The algorithm for left-tight tasks is similar to that of tight tasks, however there are some differences. Recall that each task is admissible by itself. As before, we partition the tasks into classes based on the demands. A task i is of class r if $2^r \leq d_i < 2^{r+1}$. Let $k = \lceil \log(1/\epsilon) \rceil + 1$ and let C_a for $a = 0, 1, \ldots, k-1$ be the collection of tasks of class $r \equiv a \mod k$. The algorithm computes a constant approximation for each collection C_a and outputs the maximum profit solution out of these k solutions.

Fix $a \in \{0, 1, \ldots, k-1\}$. As in the case of tight tasks, we argue that if $i, j \in C_a$ such that i is of a higher class than j, then $s_i > s_j$. Suppose, on the contrary, that $s_i \leq s_j$. Since the capacity profile is unimodal, we have $d_j < \epsilon d_i \leq \epsilon c_{s_i} \leq \epsilon c_t$ for all $t \in [s_j, t_{\text{mode}})$. This contradicts the fact that task j is left-tight. Note however that unlike the case of tight tasks, span(i) may not be a subset of span(j) (since the right end-points of tasks need not satisfy the above property).

We again argue that the optimum algorithm picks at most $2/\epsilon$ tasks from any class. Suppose, on the contrary, the optimum picks more than $2/\epsilon$ tasks from some class. Consider the task i among these with largest start time s_i. It follows that $d_i/2 \cdot 2/\epsilon < c_t$ for all $t \in [s_i, t_{\text{mode}})$ contradicting the fact that task i is left-tight. So, if we optimally select an admissible subset of tasks with at most $(1 - \epsilon)/\epsilon$ tasks from each class we obtain a $2/(1 - \epsilon)$-approximation for this collection of left-tight tasks. Given that we have a total of $k = \lceil \log(1/\epsilon) \rceil + 1$ collections C_a, the overall approximation for left-tight tasks will be $2/(1 - \epsilon) \cdot (\lceil \log(1/\epsilon) \rceil + 1)$.

Even though we no longer have the property that the span of a task is contained in the span of a task of lower class, we can easily modify the previous dynamic program to compute a maximum profit set of admissible tasks with at most $(1 - \epsilon)/\epsilon$ tasks from each class. The key idea is to ignore the capacity constraints to the right of t_{mode} altogether and consider subsets Q with at most $(1 - \epsilon)/\epsilon$ tasks in the class $r + k$ in the recurrence (3.1). Recall that any left-tight task i satisfies $d_i \leq \epsilon \cdot c_t$ for any $t \in [t_{\text{mode}}, t_i)$. Since we pick at most $(1 - \epsilon)/\epsilon$ tasks from each class, for any $t \geq t_{\text{mode}}$, the total demand from any single class that spans t is at most $(1 - \epsilon)c_t$. As the demands of the tasks in different classes of any fixed collection C_a differ by a factor of at least ϵ, the total demands of all the tasks that span $t \geq t_{\text{mode}}$ can be at most $(1 - \epsilon)c_t + \epsilon(1 - \epsilon)c_t + \epsilon^2(1 - \epsilon)c_t + \cdots < c_t$

for any $\epsilon < 1$. Thus we can never violate the capacity constraints for any $t \geq t_{\text{mode}}$ in our dynamic program. As mentioned earlier, this dynamic program yields a $2/(1 - \epsilon) \cdot (\lceil \log(1/\epsilon) \rceil + 1)$ for left-tight tasks.

4 Ring graphs: when the graph is a cycle

The UFP on cycles can be solved approximately using the algorithm for line graphs. The following approach was observed in [8]. Consider an edge e in the cycle with the smallest c_e value and partition the set of demands routed in the optimal solution OPT into two sets: let OPT$_1$ contain those demands that use edge e and let OPT$_2$ contain those that do not use edge e. Since c_e has the smallest capacity, OPT$_1$ can be approximated to within a factor of $(1 + \epsilon)$ by using the known PTAS [20] for KNAPSACK. The instance defined by considering the demands that do not use edge e is basically an instance on a line graph (obtained by deleting edge e). Thus we can use the algorithm presented in the previous section. Returning the maximum of the two solutions obtained, is an $O(\log n)$-approximation for the problem on cycle graphs.

5 Conclusions

An intriguing open question is whether there is an $O(1)$-approximation, or even an approximation scheme for the problem; we still do not know whether the UFP on line graphs (without any extra constraints) is APX-hard. Another interesting open question is to see if the techniques developed in this paper can be generalized to more families of graphs, e.g., trees. Even for restricted families of trees, such as bounded degree trees, it seems that some new ideas are needed to extend our results.

References

[1] M. Andrews, J. Chuzhoy, S. Khanna, and L. Zhang, *Hardness of the undirected edge-disjoint paths problem with congestion*, In Proc. 46th Annual IEEE Symposium on Foundations of Computer Science, pages 226–244, 2005.

[2] M. Andrews and L. Zhang, *Logarithmic Hardness of the Undirected Edge-Disjoint Paths Problem*, JACM 53(5):745-761, 2006. Earlier version in Proc. 37th Annual ACM Symposium on the Theory of Computing, pages 276-283, 2005.

[3] Y. Azar and O. Regev, *Strongly polynomial algorithms for the unsplittable flow problem*, In Proc. 8th Conference on Integer Programming and Combinatorial Optimization, pages 15-29, 2001.

[4] N. Bansal, A. Chakrabarti, A. Epstein, and B. Schieber, *A Quasi-PTAS for unsplittable flow on line graphs*, In Proc. of STOC 2006.

[5] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, B. Schieber, *A unified approach to approximating resource allocation and scheduling*, J. ACM 48(5): 1069-1090 (2001)

[6] A. Baveja and A. Srinivasan, *Approximation algorithms for disjoint paths and related routing and packing problems*, Math. Oper. Res., 25(2):255-280, 2000.

[7] G. Calinescu, A. Chakrabarti, H. Karloff, and Y. Rabani, *Improved approximation algorithms for resource allocation*, In Proceedings of the 9th Integer Programming and Combinatorial Optimization Conference, volume 2337 of Lecture Notes in Computer Science, pages 439-456, 2001.

[8] A. Chakrabarti, C. Chekuri, A. Kumar, and A. Gupta, *Approximation Algorithms for the Unsplittable Flow Problem*, Algorithmica, 47(1):53–78, 2007. Preliminary version in APPROX, September 2002.

[9] C. Chekuri, S. Khanna, and B. Shepherd, *An $O(\sqrt{n})$ Approximation and Integrality Gap for Disjoint Paths and UFP*, Theory of Computing, Vol 2, 137–146, 2006.

[10] C. Chekuri, M. Mydlarz, and B. Shepherd, *Multicommodity Demand Flow in a Tree and Packing Integer Programs*, ACM Transactions on Algorithms (TALG), 3(3), 2007. Preliminary version appeared in ICALP 2003.

[11] S. Fortune, J. E. Hopcroft, and J. Wylie, *The directed subgraph homeomorphism problem*, Theor. Comput. Sci., 10:111-121, 1980.

[12] A. Frieze, *Edge disjoint paths in expander graphs* SIAM Journal on Computing 30:1790-1801, 2001.

[13] N. Garg, V. V. Vazirani, and M. Yannakakis, *Primal-dual approximation algorithms for integral flow and multicut in trees*, Algorithmica, 18(1):3-20, 1997.

[14] V. Guruswami, S. Khanna, R. Rajaraman, F. B. Shepherd, and M. Yannakakis, *Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems*, J. Comput. Syst. Sci., 67(3):473-496, 2003. Preliminary version in Proc. 31st Annu. ACM Symp. Theory Comput., pages 19-28, 1999.

[15] J. M. Kleinberg, *Approximation Algorithms for Disjoint Paths Problems*, PhD thesis, MIT, 1996.

[16] J.M. Kleinberg and R. Rubinfeld, *Short paths in expander graphs*, In Proc. of the 37th Annual IEEE Symposium on Foundations of Computer Science, pages 86-95, 1996.

[17] P. Kolman and C. Scheideler, *Improved bounds for the unsplittable flow problem*, J. of Algorithms 61(1):20-44, 2006.

[18] A. Phillips, R. N. Uma and J. Wein, *Off-line Admission Control for General Scheduling Problems*, Journal of Scheduling, 3(6):365-381, 2000.

[19] A. Srinivasan, *Improved approximations for edge-disjoint paths, unsplittable flow, and related routing problems*, In Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, pages 416-425, 1997.

[20] V. Vazirani, *Approximation Algorithms*, Springer, 2003.