A Logarithmic Approximation for Unsplittable Flow on Line Graphs

NIKHIL BANSAL, Eindhoven University of Technology
ZACHARY FRIGGSTAD, University of Waterloo
ROHIT KHANDEKAR, IBM T.J. Watson Research Center
MOHAMMAD R. SALAVATIPOUR, University of Alberta

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.

We present the first polynomial-time $O(\log n)$-approximation algorithm for this problem. This significantly advances the state-of-the-art, as no polynomial-time $o(n)$-approximation was known previously. Previous results for this problem were known only in more restrictive settings, in particular, either the instance satisfies the so-called "no-bottleneck" assumption: $\max_i d_i \leq \min_t c_t$, or the ratio of both maximum to minimum demands and maximum to minimum capacities are polynomially (or quasi-polynomially) bounded in n. Our result, on the other hand, does not require 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.

Categories and Subject Descriptors: F.2.2 [**Nonnumerical Algorithms and Problems**]: Computations on Discrete Structures

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Approximation algorithms, resource allocation problem, unsplittable flow, scheduling

ACM Reference Format:
Bansal, N., Friggstad, Z., Khandekar, R., Salavatipour, M.R. 2012. A logarithmic approximation for unsplittable flow on line graphs. ACM Trans. Embedd. Comput. Syst. 9, 4, Article 39 (March 2010), 15 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

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 demand pairs $T = \{(s_i, t_i)\}_{1 \leq i \leq n}$ where each pair $s_i, t_i \in V$ 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 of the pair 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

A preliminary version of this paper appeared in the Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) 2009.
The last author was supported by NSERC and an Alberta Ingenuity new faculty award.
Author's emails: Bansal, N., n.bansal@tue.nl, Friggstad, Z., zfriggstad@uwaterloo.ca, Khandekar, R., rohitk@us.ibm.com, Salavatipour, M.R., mreza@cs.ualberta.ca.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

simultaneously without violating any edge capacity, i.e., the total demand routed on any 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) which 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 networks. In such an instance, the input graph G is an undirected path (line). The study of UFP on line graphs[1] 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 lines is equivalent to the following scheduling problem called Resource Allocation Problem (or RAP for short). In this problem, we are given n tasks, each specified by a start time s_i, end time t_i, demand d_i, and profit p_i. The task i, if scheduled, requires d_i units of a resource in the time interval $[s_i, t_i)$, called 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 lines.

UFP continues to be a difficult problem even when restricted to lines and obtaining a reasonable approximation for it has resisted several attempts. One difficulty is that the natural 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 $d_{\max} \leq c_{\min}$ where $d_{\max} = \max_i d_i$ and $c_{\min} = \min_t c_t$. Note that this requires that 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 the problem of EDP which is very well-studied. This problem is NP-hard in general graphs (with non-constant number of terminal-pairs), and NP-hard even with only two terminal-pairs in directed graphs (see [Fortune et al. 1978]). The first approximation algorithm for EDP was an $O(\sqrt{|E|})$ approximation [Kleinberg 1996] that was later generalized to UFP under the so-called "no-bottleneck" assumption: $\max_i d_i \leq \min_e c_e$ [Srinivasan 1997; Baveja and Srinivasan 2000]. More recently, this was improved these to an $O(\sqrt{|V|})$-approximation [Chekuri et al. 2006] . On the other hand, it is known that EDP on directed graphs is NP-hard to approximate within $\Omega(|E|^{\frac{1}{2}-\epsilon})$ for

[1]We use the term "line graphs" or "line networks" to refer to graphs that consist of a simple path, as done in the previous works on UFP. Our usage of term line graphs should 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.

any constant $\epsilon > 0$ [Guruswami et al. 2003] . In the undirected setting EDP is quasi-NP-hard to approximate within $\Omega(\log^{\frac{1}{2}-\epsilon}|E|)$ for any $\epsilon > 0$ [Andrews and Zhang 2006; Andrews et al. 2005] . All these results give the same hardness for UFP even with the no-bottleneck assumption in the corresponding model. Without the no-bottleneck assumption, UFP is NP-hard to approximate within $\Omega(|E|^{1-\epsilon})$ [Azar and Regev 2001]. For the case of trees, UFP is known to be APX-hard [Garg et al. 1997].

Several papers have studied UFP and EDP on graphs with high expansion, we name a few here. For instance, in (large) constant-degree regular expanders with sufficiently high expansion, there is a constant c such that any $cn/\log n$ pairs for which no vertex appears in more than $O(1)$ pairs, can be connected via edge-disjoint paths [Frieze 2000]. This implies an $O(\log n)$-approximation for EDP on such expanders. Using earlier works by [Kleinberg and Rubinfeld 1996], [Srinivasan 1997] 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 2006] and [Chakrabarti et al. 2002].

The special case of the EDP problem on line networks 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 lines. This problem is equivalent to a resource allocation problem that has been studied by [Bar-Noy et al. 2001] and [Phillips et al. 2000]. The first constant approximation algorithm for UCUFP on lines was provided by [Phillips et al. 2000]. The approximation ratio of the problem was later improved in a series of papers [Bar-Noy et al. 2001; Calinescu et al. 2011] to $(2+\epsilon)$.

For the general UFP on lines, as mentioned earlier, the problem has not been easy to approximate. Therefore, most of the previous works have made some extra assumptions in order to get a reasonable approximation. A typical extra assumption has been to consider instances of UFP on lines with the no-bottleneck assumption. For UFP on line graphs, with the no-bottleneck assumption, [Chakrabarti et al. 2002] presented the first constant approximation which was later improved by [Chekuri et al. 2007] to a $(2+\epsilon)$-approximation (again under the no-bottleneck assumption). Without the no-bottleneck assumption, 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 [Bansal et al. 2006]. Finally, for instances of UFP with bounded "aspect ratio" $d_{\max}/d_{\min}$, the integrality gap of the natural LP relaxation for UFP on line graphs is $\Omega(\log(d_{\max}/d_{\min}))$ [Chakrabarti et al. 2002]. They also gave an example where this can be as bad as $\Omega(n)$.

1.2. Our result and techniques

In this paper we study the UFP on lines, or equivalently, the Resource Allocation Problem (RAP) defined earlier. We present an $O(\log n)$-approximation for RAP (i.e. the UFP on lines) 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 networks.

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.

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

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

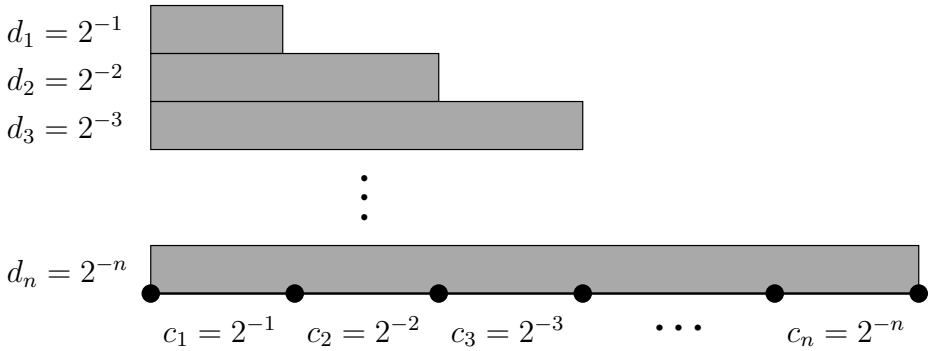

Fig. 1. The example showing an integrality gap of $\Omega(n)$. The demands are not drawn to scale as they decrease exponentially.

by [Chakrabarti et al. 2002]. 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 $d_i = 1/2^i$. All the tasks have profit 1 and the capacity c_t during interval $[t, t+1)$ is equal to $1/2^{t+1}$, for $0 \leq t \leq n-1$ (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 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 $j^* - 1$ (recall that $d_{j^*} = 1/2^{j^*} = c_{j^*-1}$). 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 [Chakrabarti et al. 2002; Chekuri et al. 2007; Bansal et al. 2006] to obtain a constant ratio approximation for UFP on lines 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, can be solved well using LP relaxation, and the second 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 "nested", that make the instance amenable to dynamic programming.

More precisely, 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 some common time. We then describe an $O(1)$-approximation for such intersecting instances. To do this, we partition the tasks into *slack* tasks and *tight* tasks. Slacks tasks are those whose demands are a small fraction of the minimum capacity available during their respective spans. The rest of the tasks are *tight* tasks. It is known that if all the tasks are slack and intersecting, then there is a randomized rounding based $O(1)$-approximation for the problem (see [Chakrabarti et al. 2002; Chekuri et al. 2007; Bansal et al. 2009]). 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 that each task is tight.

Recent developments: Since the initial announcement of this work [Bansal et al. 2009], a stronger LP relaxation for UFP has been studied by [Chekuri et al. 2009]. Specifically, they devise a different LP relaxation that can be solved approximately within a constant factor (they give an approximate separation oracle) and bound the integrality gap of this new LP by $O(\log^2 n)$ for UFP on paths. Additionally, they also present a combinatorial $O(\log^2 n)$-approximation algorithm for UFP on trees.

Even more recently, a polynomial-time $(7+\epsilon)$-approximation has been demonstrated for any constant $\epsilon > 0$ [Bonsma et al. 2011]. They also demonstrate that UFP on line graphs is strongly NP-hard even when the edge capacities are uniform. Before their result, the only known hardness for UFP was the weak NP-hardness it inherits from the Knapsack problem.

2. REDUCTION TO INTERSECTING CASES

We use $\text{span}(i)$ to denote the interval $[s_i, t_i)$. Also define the length of task i by $\text{length}(i) = t_i - s_i$. It is easy to see we may assume that s_i and t_i are positive integers for each task i. Furthermore, we may assume $1 \leq s_i < t_i \leq 2n$ for the following reason. If t is a time that is not the start or end of any task, then we simply subtract one from each s_i and t_i that is greater than t and set the capacity of c_{t-1} to the minimum of the original values c_{t-1} and c_t. There is a clear correspondence between feasible solutions before and after such an update. This also means we may assume $1 \leq \text{length}(i) < 2n$ for each task i.

Tasks i and j are said to *intersect* if they share a common time. That is, i and j intersect if $\text{span}(i) \cap \text{span}(j) \neq \emptyset$. Say a collection of tasks is *intersecting* if all of the tasks share a common time; this is equivalent to the property that the tasks in the collection pairwise intersect. Finally, say that a collection of tasks C is *feasible* if, for all times t we have $\sum_{i \in C : t \in \text{span}(i)} d_i \leq c_t$.

We may assume that each task is admissible by itself. This means we can trivially obtain a profit of $p_{\max} = \max_i p_i$. Now, we perform a preprocessing step via a standard adjustment to the input (*e.g.* see [Vazirani 2003]). Fix a constant $0 < \epsilon' < 1$ and adjust each profit p_i to $p'_i := \lfloor \frac{np_i}{\epsilon' p_{\max}} \rfloor$. Notice the modified profits are integers between 0 and $\lfloor n/\epsilon' \rfloor$. Let OPT and OPT' respectively denote the maximum total profit of a feasible set of tasks before and after adjusting the profits. It is easy to see that a subset of tasks with profit at least $\alpha OPT'$ under the new profits has total profit at least $(1-\epsilon')\alpha OPT$ under the old profits. Therefore:

Assumption: At a loss of $(1-\epsilon')$ in the approximation factor, from now on we assume that all p_i values are integers between 0 and $\lfloor n/\epsilon' \rfloor$.

We now describe a reduction procedure which allows us to focus only on a subset of tasks which is intersecting while losing a factor of only $O(\log n)$ in the approximation guarantee.

LEMMA 2.1. *If there is a ρ-approximation for instances of UFP on a line where all tasks intersect, then there is an $O(\rho \log n)$-approximation for the general instance of UFP on a line.*

PROOF. Consider a general instance of UFP on a line. We first group the tasks according to their lengths. Say a task i belongs to group G_r if $2^r \leq \text{length}(i) < 2^{r+1}$. Since we have $\text{length}(i) < 2n$ for all tasks i, then $r \in \{0, 1, \ldots, \lceil \log_2 2n \rceil - 1\}$. Focus on an optimum set of feasible tasks T with profit OPT. Note that one of the groups G_r must have at least a $\frac{1}{\lceil \log_2 2n \rceil}$-fraction of the total profit of T. That is, if OPT_r is the optimum profit over all feasible subsets of tasks in group G_r, then $OPT_r \geq \frac{OPT}{\lceil \log_2 2n \rceil}$ for some r.

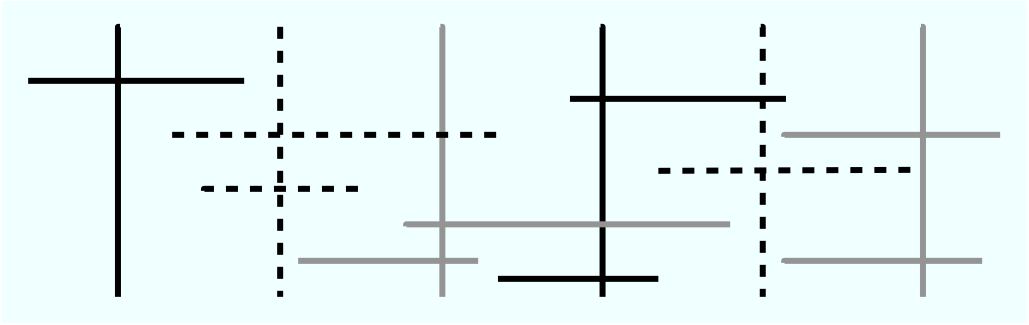

Fig. 2. Grouping the tasks according to the left-most point of the form $k2^r$ for some integer k.

Consider a group G_r, each task $i \in G_r$ must have $k2^r \in \text{span}(i)$ for some integer k. Create groups $H_{r,k}$ for $k \in \mathbb{Z}$ and place $i \in G_r$ in group $H_{r,k}$ if k is the *least* integer for which $k2^r \in \text{span}(i)$. One sees that $H_{r,k}$ is an intersecting collection of tasks (figure 2 helps illustrate this). Observe that for tasks $i \in H_{r,k}$ and $j \in H_{r,l}$ with $k+3 \leq l$ we have $\text{span}(i) \cap \text{span}(j) = \emptyset$. This follows since $\text{length}(i) < 2^{r+1}$ and $s_i \leq k2^r$ imply $t_i < (k+2)2^r$. Furthermore, since l is the least integer for which $s_j \leq l2^r$ then $s_j \geq (k+2)2^r > t_i$.

Now, apply the ρ-approximation to each $H_{r,k}$ and let $C_{r,k}$ denote the collection of tasks chosen by the algorithm. For each $l = 0, 1, 2$, let $C'_{r,l}$ be the union of all $C_{r,l'}$ with $l' \equiv l \pmod 3$. By arguments in the previous paragraph, none of the tasks in $C_{r,k}$ can intersect any task in $C_{r,l}$ if $k+3 \leq l$ so $C'_{r,l}$ is a feasible collection of tasks for each $l = 0, 1, 2$. Furthermore, by looking at the restriction of the optimum solution OPT_r for group G_r to the subgroups $H_{r,k}$, we see that at least one of the three groups $C'_{r,l}$ has profit at least $\frac{\rho OPT_r}{3}$. Thus, for some $r \in \{0, 1, \ldots, \lceil \log_2 2n \rceil - 1\}$ and some $l \in \{0, 1, 2\}$ we have the total profit of tasks in $C'_{r,l}$ is at least $\frac{\rho}{3\lceil \log_2 2n \rceil} \cdot OPT$. □

In the next section, we will develop a constant-factor approximation for instances of UFP on a line where all tasks are intersecting. Combined with the preceding lemma, this yields the following:

THEOREM 2.2. *There is an $O(\log n)$-approximation for UFP on lines.*

3. INTERSECTING CASES

Consider an instance of UFP where all tasks share a time t. Let $0 < \epsilon \leq 1/2$ be some constant. We say time k is a *bottleneck* for task i if $k \in \text{span}(i)$ and $d_i \geq \epsilon c_k$. We classify each task i accordingly:

— if no time is a bottleneck for i then say i is *slack*
— if $k \leq t$ for all bottlenecks k for i then say i is *left-tight*
— if $k > t$ for all bottlenecks k for i then say i is *right-tight*
— if i has bottlenecks on both sides of t then simply say i is *tight*

Partition the tasks into four groups according to the classification above. We describe a constant-factor approximation for each such group. The maximum total profit of these four approximate solutions is then within a constant factor of the optimum solution (since one of these groups has a solution consisting of at least 1/4 of the optimum profit).

There is one further simplification we apply. If all tasks share a common point t, then we may assume the following structure on the capacities. For each $i < j \leq t$, we

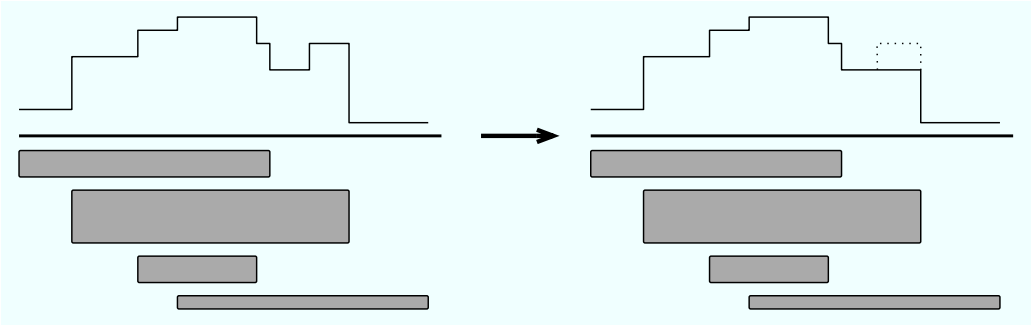

Fig. 3. Illustrating why we may assume the capacity profile is unimodal.

have $c_i \leq c_j$ and for each $t \geq j > i$ we have $c_j \geq c_i$. That is, the capacities increase as we move from the left to t and decrease as we move from t to the right. The reason is this: in any feasible collection of tasks T and any for $i < j \leq t$ we have that the total demand in T at time j is at least the total demand in T at time i. By this reasoning, we may reduce the value c_i to c_j and not worry about affecting feasibility of any solution. If the capacities satisfy this, then we say the capacity profile is *unimodal*. See Figure 3 for an illustration.

3.1. Slack Tasks

For the case of slack tasks, [Chakrabarti et al. 2002] present an LP randomized rounding algorithm that finds a feasible subset of tasks of total profit within a constant factor of the LP optimum for instances satisfying the no bottleneck assumption. Though the slack tasks in our case may not satisfy this assumption, essentially the same rounding algorithm can be seen to provide a constant factor approximation for intersecting cases of slack tasks. For the sake of completeness, we present the full algorithm and proof in the case of intersecting slack tasks. The analysis is simpler in our case because the tasks are intersecting.

We recall the standard LP for UFP.

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

Though it has an $\Omega(n)$ integrality gap in general cases, we will prove it has an $O(1)$-integrality gap for intersecting cases of slack tasks when ϵ is regarded as a constant. From now on, let x^* denote an optimum solution to the above LP.

Consider the following algorithm. Since the capacity profile is unimodal, the minimum capacity of all times spanned by a task i is at either s_i or $t_i - 1$. Let $C_{\leq}$ be the set of tasks with $c_{s_i} \leq c_{t_i-1}$ and let $C_{>}$ be the set of tasks with $c_{s_i} > c_{t_i-1}$. That is, $C_{\leq}$ is the collection of tasks whose most constrained time is the start time and $C_{>}$ is the collection of tasks whose most constrained time is the end time.

The rounding algorithm proceeds as follows. We first ignore the tasks in $C_{>}$ and focus only on tasks in $C_{\leq}$. The algorithm for rounding tasks in $C_{>}$ is similar to what follows so it is omitted. Next, order the tasks in $C_{\leq}$ in the increasing order of their starting times. We choose each task $i \in C_{\leq}$ independently with probability $x_i^*(1-\epsilon)/2$. Let R denote the set of chosen tasks and say these tasks are $i_1 < i_2 < \cdots < i_{|R|}$. We construct a sequence of sets $\emptyset = S_0, S_1, \ldots,$ 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 decision whether task i lies in S or not is correlated to whether other tasks lie in S or not. We will show that:

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

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 $E[Y_i]$. Since $S \subseteq R$, we have $Y_i \leq X_i$ and hence $E[Y_i] \leq E[X_i]$. 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 3.2. *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 and defining property of the 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 $t' \leq t$, then from the assumption that capacity profile is unimodal, we have $c_{s_i} \leq c_{t'}$. If $t' > t$, 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 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 $c_{s_i} - d_i$. 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 3.3. $Pr[E_r] \leq 1/2$

PROOF. We know that $Pr[E_r] \leq Pr[D \geq c_{s_i} - d_i] \leq Pr[D \geq c_{s_i} - \epsilon c_{s_i}]$. The second step follows as all tasks are slack.

We have $E[D] = \sum_{j=1}^{i-1} E[D_j] = \sum_{j=1}^{i-1} x_j^*(1-\epsilon)/2 \leq c_{s_i}(1-\epsilon)/2$. The last inequality holds since the fractional solution x^* satisfies the capacity constraint at time s_i. Thus, by Markov's inequality, $Pr[D \geq c_{s_i}(1-\epsilon)] \leq 1/2$. □

Now,

$$\begin{aligned} E[Y_i] &= Pr[Y_i = 1 \mid X_i = 1] \cdot Pr[X_i = 1] + Pr[Y_i = 1 \mid X_i = 0] \cdot Pr[X_i = 0] \\ &= Pr[Y_i = 1 \mid X_i = 1] \cdot Pr[X_i = 1] \\ &= (1 - Pr[E_r]) \cdot x_i^*(1-\epsilon)/2 \\ &\geq x_i^*(1-\epsilon)/4 \end{aligned}$$

as claimed.

Say that $S_{\leq}$ and $S_{>}$ are, respectively, the subsets of $C_{\leq}$ and $C_{>}$ found through the above rounding algorithm. If z^* is the value of the LP solution for the slack tasks, using Theorem 3.1 and returning the most profitable of $S_{\leq}$ and $S_{>}$, the expected value of the solution obtained is at least $\frac{1-\epsilon}{8} z^*$.

3.2. Tight Tasks

As a warm-up, consider the special case where the tasks form a sequence of nested intervals, called a nested instance. That is, say the tasks can be ordered such that $s_i \leq s_j \leq t_j \leq t_i$ for all $i \leq j$. The following notation will be useful in all cases. For a

task i, let $\mathbf{cap}(i) = \min_{k \in \mathbf{span}(i)} c_k$ denote the minimum capacity over all times in the span of i.

THEOREM 3.4. *There is an FPTAS for nested and tight instances.*

PROOF. The algorithm is based on dynamic programming similar to the one used for knapsack problems. For integers i, p, let $f(i,p)$ be the minimum total demand among feasible subsets $S \subseteq \{1, \ldots, i\}$ that achieve profit exactly p. If it is not possible to obtain profit exactly p using the first i tasks then say $f(i,p) = \infty$. That values of $f(i,p)$ are computed in the order of increasing i. Clearly $f(0,0) = 0$ and $f(0,p) = \infty$ for $p > 0$. We claim the following recurrence is satisfied by the values $f(i,p)$ for $i > 0$:

$$f(i,p) = \begin{cases} \min\{f(i-1,p), f(i-1,p-p_i) + d_i\} & \text{if } p_i \leq p \text{ and } f(i-1,p-p_i) + d_i \leq \mathbf{cap}(i) \\ f(i-1,p) & \text{otherwise} \end{cases}$$

To see this, consider some $i > 0$ and profit p. If $f(i,p) = \infty$ then surely $f(i-1,p) = \infty$. Furthermore, if $p_i \leq p$ and $f(i-1,p-p_i) < \infty$ then we claim that $f(i-1,p-p_i) + d_i > \mathbf{cap}(i)$. If this were not so, then consider some feasible set S' of the first $i-1$ tasks with minimum possible demand with profit exactly $p - p_i$. By definition, all $t \in \mathbf{span}(i)$ have $c_t \geq \mathbf{cap}(i)$. Thus, if $f(i-1,p-p_i) + d_i \leq \mathbf{cap}(i)$ then $S' \cup \{i\}$ is a feasible subset of the first i tasks obtaining profit p which contradicts $f(i,p) = \infty$. Therefore, the recurrence is satisfied for those $(i,p), i > 0$ for which $f(i,p) = \infty$.

On the other hand, suppose $f(i,p) < \infty$. Consider some set S of the first i tasks with minimum possible demand that obtains profit exactly p (*i.e.* the demand of S is $f(i,p)$). If $i \notin S$ then $f(i-1,p) \leq f(i,p)$ since S is also a feasible set of the first $i-1$ tasks. We also have $f(i-1,p) \geq f(i,p)$ since any subset of the first $i-1$ tasks is also a subset of the first i tasks. Combining these inequalities shows $f(i-1,p) = f(i,p)$. If $i \in S$ then $S \setminus \{i\}$ is a feasible subset of the first $i-1$ demands so $f(i-1,p-p_i) \leq f(i,p) - d_i$. If $f(i-1,p-p_i) < f(i,p) - d_i$, then by reasoning in a manner similar to the previous paragraph, any feasible set S' of profit $p - p_i$ of the first $i-1$ tasks with demand $f(i-1,p-p_i)$ can be extended to a feasible set $S' \cup \{i\}$ of the first i elements with profit p and demand $f(i-1,p-p_i) + d < f(i,p)$ which is a contradiction. Therefore, $f(i-1,p-p_i) + d_i = f(i,p)$. In either case of $i \in S$ or $i \notin S$, the recurrence is satisfied.

The value of the optimum solution is then the largest value p for which $f(n,p) < \infty$. By the simplifications made in Section 2, the only values of p which may be finite are integers in the range $[0, np_{\max}]$. Since $p_{\max} \leq \lfloor n/\epsilon' \rfloor$ for some given constant $\epsilon' > 0$ and i ranges from 0 to n (based on the assumption made in Section 2), then the above recurrence can be computed with dynamic programming in time $O(n^3/\epsilon')$. □

Now we go back to the more general case. A collection of tight tasks can be made to look something like a sequence of nested intervals. Assume, by scaling the demands and capacities, that all demands are at least 1. Create groups of demands D_k where $i \in D_k$ if $\epsilon^{-k} \leq d_i < \epsilon^{-k-1}$. We have the following structure between groups D_k's which says if task i has much less demand than task j then task j is nested in task i. The basic idea is that task j, being feasible on its own, cannot cross any bottleneck time for task i since the demand for j is much higher than the demand of i while any bottleneck time for i has capacity close to the demand of i.

LEMMA 3.5. *If $i \in D_k$ and $j \in D_l$ with $k+2 \leq l$ then $s_i < s_j$ and $t_i > t_j$ (*i.e. *$\mathbf{span}(j) \subseteq \mathbf{span}(i)$).*

PROOF. Assume, by way of contradiction, that $s_i \geq s_j$. Since $d_i < \epsilon^{-k-1}$ and $\epsilon^{-l} \leq d_j$, then $d_i < \epsilon d_j$ (by $k+2 \leq l$). Also, since i is tight and the capacity profile is unimodal, then $d_i > \epsilon c_{s_i}$ which shows $d_j > c_{s_i}$. However, since $s_i \geq s_j$ and tasks i and j intersect,

then $s_i \in \text{span}(j)$. This contradicts the fact that task j is feasible by itself. A similar argument shows $t_i > t_j$. □

Let $D'_0 = D_0 \cup D_2 \cup D_4 \cup \ldots$ and $D'_1 = D_1 \cup D_3 \cup D_5 \cup \ldots$ and notice that the entire collection of intersecting tight tasks are partitioned between D'_0 and D'_1. We also have the following observation that bounds the size of a feasible subset of any D_k. Suppose time j is a bottleneck for some task $i \in D_k$. Then since c_j is close to d_i and $d_{i'}$ is close to d_i for all $i' \in D_k$, then only a few task in D_k can fit across j in any feasible solution. Finally, since the tasks in D_k are intersecting and since each has a bottleneck time on either side of the common time t, then in any collection of tasks in D_k there is a task that has one of its bottleneck times spanned by all other tasks. Formally:

LEMMA 3.6. *Let B_k be any feasible subset of D_k. Then $|B_k| \leq \epsilon^{-2}$.*

PROOF. Let $i \in B_k$ be such that $s_i \geq s_j$ for all $j \in B_k$. Notice that $s_i \in \text{span}(j)$ for all $j \in B_k$. Now, by definition of D_k we have $d_j \geq \epsilon d_i$ for all $j \in B_k$. Furthermore, since i is tight we have $d_i \geq \epsilon c_{s_i}$. Therefore, the total demand in B_k at time s_i is at least $|B_k|\epsilon^2 c_{s_i}$. Since B_k is feasible, then the total demand crossing s_i must be at most c_{s_i}. Therefore, $|B_k| \leq \epsilon^{-2}$. □

Lemmas 3.5 and 3.6 lead to a dynamic programming solution. For each $i \in \{0, 1\}$ we will find an optimum feasible collection of tasks in D'_i in the following manner. We build a table $f(k, p)$ that is the minimum total demand using only tasks in groups $D_k, D_{k-2}, D_{k-4}, \ldots$ to obtain profit exactly p. To build the $f(k + 2, p)$ values from the $f(k, p)$ values, we will try adding subsets of D_k of size at most ϵ^{-2}. By Lemma 3.5, each tasks in D_{k+2} is contained in the span of every task in D_k which resembles the property that the tasks form a nested sequence of intervals. To simplify notation, for a set of tasks S let $p_S = \sum_{i \in S} p_i$ and $d_S = \sum_{i \in S} d_i$. Furthermore, we extend the definition of $f(k, p)$ to include $k = -1$ and $k = -2$ which should simply read as $f(k, 0) = 0$ and $f(k, p) = \infty$ for $p > 0$ whenever $k = -1$ or -2 (in other words, one can only obtain a profit of 0 if no tasks are chosen).

The following notation will be helpful. Let $A(k, p)$ be the collection of all subsets S of D_k of size at most ϵ^{-2} with $p_S \leq p$ and the following additional property. For any subset T of $D_{k-2} \cup D_{k-4} \cup \ldots$ with total profit $p - p_S$ and total demand $d_T = f(k - 2, p - p_S)$ we have $d_T + \sum_{i \in S: j \in \text{span}(i)} d_i \leq c_j$ for each time j contained in the common intersection of all tasks in classes $D_l, l < j$. Intuitively, a set S in $A(k, p)$ is one that can extend any optimum set T corresponding to $f(k - 2, p - p_S)$ to a feasible solution $S \cup T$. We only have to verify the capacity constraints are satisfied for those times in the common intersection of all tasks in some lower class $D_l, l + 2 \leq k$. Again, by Lemma 3.5 this is because the span of each task in D_k is completely contained in the span of each task in some $D_l, l + 2 \leq k$.

We can efficiently determine the members of $A(k, p)$ in the following way. While the above definition for $A(k, p)$ may consider exponentially many $T \subseteq D_{k-2} \cup D_{k-4} \cup \ldots$, all such sets T have the same total demand across the common intersection of all tasks in classes $D_l, l < j$ so to determine if $S \in A(k, p)$ it is enough to know the value $f(k-2, p-p_S)$. Lemma 3.6 essentially says we can restrict our attention to small subsets of D_k since any subset larger than ϵ^{-2} is not feasible on its own. Therefore, we can determine the members of $A(k, p)$ in polynomial time by iterating over all subsets S of D_k of size at most ϵ^{-2} and checking that $p_S \leq p$ and $f(k - 2, p - p_S) + \sum_{i \in S: j \in \text{span}(i)} d_i \leq c_j$ for each j contained in the common intersection of all tasks in classes $D_l, l < j$.

Formally, the recurrence for relating the $f(k, p)$ values looks like:

— $f(k, 0) = 0$ for $k \in \{-1, -2\}$

— $f(k,p) = \infty$ for $k \in \{-1,-2\}, p > 0$
— $f(k,p) = \infty$ for $k \geq 0$ if $A(k,p) = \emptyset$
— $f(k,p) = \min_{S \in A(k,p)} f(k-2, p-p_S) + d_S$ for $k \geq 0, A(k,p) \neq \emptyset$

LEMMA 3.7. *The recurrence correctly relates the values of* $f(k,p)$.

PROOF. The base cases with $k < 0$ are clearly correct (when interpreted as suggested above). Now, consider some $k \geq 0$ and profit p. If $f(k,p) < \infty$ then let $S = S' \cup B$ be a subset of $D_k \cup D_{k-2} \cup \ldots$ obtaining profit p with total demand $f(k,p)$ and $S \cap D_k = B$. We first verify that $B \in A(k,p)$. By Lemma 3.6, we know $|B| \leq \epsilon^{-2}$ (in fact, B may be empty). Furthermore, we also clearly have $p_B \leq p$. Finally, since S' is a feasible subset of $D_{k-2} \cup D_{k-4} \cup \ldots$ then $f(k-2, p-p_B) \leq f(k,p) - d_B$. By Lemma 3.5, any optimum set S^* with profit $p - p_B$ and demand $f(k-2, p-p_B)$ places less demand across each time spanned by B than set S'. For such a set S^* we have $S^* \cup B$ being feasible. Therefore, $B \in A(k,p)$.

In fact, the arguments at the end of the last paragraph show that $d_{S^*} = d_{S'}$ for S^* an optimum set corresponding to $f(k-2, p-p_B)$. Indeed, if $d_{S^*} < d_{S'}$ then the feasible set $B \cup S^*$ has demand strictly less than $B \cup S'$ and profit p which contradicts that $d_S = d_{B \cup S'} = f(k,p)$. Therefore, any optimum subsets S of $D_k \cup D_{k-2} \cup \ldots$ with profit p and demand $f(k,p)$ has $S \cap D_k \in A(k,p)$ and $S \setminus D_k$ being an optimum subset of $D_{k-2} \cup D_{k-4} \cup \ldots$ with profit $p - p_B$ and demand $f(k-2, p-p_B)$ so the recurrence correctly determines $f(k,p)$ in this case.

On the other hand, if $f(k,p) = \infty$ then $A(k,p) = \emptyset$. This is because any $B \in A(k,p)$ is such that $f(k-2, p-p_B) + d_B \leq c_j$ for all tasks j in the common intersection of tasks in $D_{k-2} \cup D_{k-4} \cup \ldots$. Thus, by definition we would be able to extend any such set of tasks with demand $f(k-2, p-B)$ to a feasible set of tasks obtaining profit p with demand $f(k,p)$. □

THEOREM 3.8. *There is a polynomial-time 2-approximation for UFP when the tasks are intersecting and tight and all profits are integers bounded by* $O(n/\epsilon')$ *for some constant* $\epsilon' > 0$.

PROOF. As in Theorem 3.4, the highest p for which $f(k,p) \neq \infty$ is then the optimum profit of a feasible subset of tasks in D'_0 or in D'_1. This, in turn, is at least 1/2 of the total profit of an optimum subset of tight tasks. By Lemma 3.7, we can compute the values $f(k,p)$ using dynamic programming (notice the recurrence for a given pair (k,p) only refers to pairs (k',p') for which $k' < k$).

The total profit is $O(n^2/\epsilon')$ and the profit of any subset of tasks is an integer. The number of integers k for which $D_k \neq \emptyset$ is also at most n. Therefore, the total number of $f(k,p)$ entries that need to be considered is $O(n^3/\epsilon')$. For each k and each p, we have $|A(k,p)| \leq n^{\epsilon^{-2}}$ by Lemma 3.6 so the values $f(k,p)$ can be computed in a dynamic programming fashion in polynomial time if ϵ and ϵ' are regarded as fixed constants. □

3.3. Left-Tight and Right-Tight

We describe the algorithm for left-tight tasks. The algorithm for right-tight tasks is essentially identical. As in the case of tight tasks, we group the tasks according to their demand. That is, task i is in group D_k if $\epsilon^{-k} \leq d_i < \epsilon^{-k-1}$. As before, let D'_0 be $D_0 \cup D_2 \cup D_4 \cup \ldots$ and let D'_1 be the remaining tasks. We have the following lemmas whose proofs are readily adapted from the analogous results for tight tasks.

LEMMA 3.9. *If* $i \in D_k$ *and* $j \in D_l$ *with* $k+2 \leq l$, *then* $s_i < s_j$.

LEMMA 3.10. *Let* B_k *be any feasible subset of* D_k. *Then* $|B_k| \leq \epsilon^{-2}$.

Furthermore, we have the following observation. Recall that t is the point on the line which all tasks share.

LEMMA 3.11. *If S is any subset of D'_0 or D'_1 such that $|S \cap D_k| \leq \frac{1-\epsilon}{\epsilon}$ for each k, then the total demand in S at any given time $m > t$ does not exceed c_m.*

PROOF. Note that all the tasks are left-tight. So for any such task d_i and any time $m > t$ we have $d_i \leq \epsilon c_m$. Let $m > t$ and let k be the largest integer such that $\epsilon^{-k-1} \leq \epsilon c_m$. Using the above argument, notice that $D_{k'} \cap S = \emptyset$ for any $k' > k$ since the tasks are only left-tight. Now, for any $k' \leq k$ we have the total demand in $S \cap D_{k'}$ being at most:

$$\frac{1-\epsilon}{\epsilon} \cdot \epsilon^{-k'-1} \leq \frac{1-\epsilon}{\epsilon} \cdot \epsilon^{k-k'+1} c_m$$

since $\epsilon^{-k-2} \leq c_m$ by our choice of k. Summing over all $k' \leq k$ shows the total demand in S across time m is bound by:

$$(1-\epsilon)c_m \sum_{k'=0}^{k} \epsilon^{k-k'} \leq (1-\epsilon)c_m \sum_{k'=0}^{\infty} \epsilon^{k'} = c_m$$

Therefore, the capacity constraint at time m is not violated by S. □

The preceding lemmas indicate that we can use a dynamic programming algorithm similar to the one for tight tasks. The main difference is that we only need to be concerned with the times $m \leq t$ if we ensure we only take subsets of D_k of size at most $\frac{1-\epsilon}{\epsilon}$. Furthermore, since the optimum solution chooses at most ϵ^{-2} tasks from each D_k then the resulting solution found will be close to the optimum. Let $f(k, p)$ denote the minimum total demand of a feasible collection of tasks from groups $D_k, D_{k-2}, D_{k-4}, \ldots$ that has total profit exactly p. The recurrence looks identical to the one for tight tasks except the set $A(k, p)$ is restricted to subsets of D_k of size at most $\frac{1-\epsilon}{\epsilon}$ and the only times we need to check for feasibility those times $j \leq t$ in the common intersection of all tasks in $D_{k-2} \cup D_{k-4} \cup \ldots$.

The proof of correctness is similar to that of the recurrence for tight tasks. The only difference is that Lemma 3.11 assures us that no time to the right of the common time t will be violated by any subset S that represents any finite $f(k, p)$ entry. Furthermore, the recurrence can be computed in polynomial time if ϵ and ϵ' are fixed constants since there are at most n distinct values for k, the maximum total profit p to be considered is $O(n^2/\epsilon')$, and the number of subsets of each D_k that need to be iterated over is at most $n^{\frac{1-\epsilon}{\epsilon}}$.

LEMMA 3.12. *The best of the two solutions found by running the dynamic programming algorithm on D'_0 and D'_1 has total profit at least a $\frac{\epsilon(1-\epsilon)}{2}$-fraction of the optimum solution for these left-tight tasks.*

PROOF. Let T be an optimum collection of left-tight tasks. For each D_k, discard all but the $\frac{1-\epsilon}{\epsilon}$ most profitable tasks in $T \cap D_k$ from T; call this new set T'. Since $|T \cap D_k| \leq \epsilon^{-2}$ then a $\epsilon(1-\epsilon)$-fraction of the total profit of T remains in T'. Now, partition T' into two groups T_1 and T_0 where $T_0 \subseteq D'_0$ and $T_1 \subseteq D'_1$. Surely one of T_1 or T_2 contains at least half of the profit of T'. Therefore, the profit of one of T_1 or T_2 is within a factor $\frac{\epsilon(1-\epsilon)}{2}$ of the optimum profit.

The dynamic programming routine finds the optimum profit subsets S_0 of D'_0 and S_1 of D'_1. So the profit of S_0 is at least the profit of T_0 and the profit of S_1 is at least the profit of T_1. Thus, one of S_0 or S_1 has total profit within a factor of $\frac{2}{\epsilon(1-\epsilon)}$ of the profit of T. □

THEOREM 3.13. *There is a polynomial-time $\frac{2}{\epsilon(1-\epsilon)}$-approximation for intersecting instances of UFP that are either left-tight or right-tight.*

3.4. Bringing it Together

We presented a constant-factor approximation for all four types of tasks in intersecting cases. Let ρ be the worst approximation ratio among these algorithms. Taking the most profitable of the four solutions found for these cases is a 4ρ-approximation for intersecting cases. Since each group G_r can be partitioned into 3 sets, each of which can be partitioned into intersecting cases where no two demands from different intersecting cases share a common time, then we have a 12ρ-approximation for each group G_r.

Since there are $\lceil \log_2(2n) \rceil$ groups, then the overall algorithm finds a solution obtaining at least $\frac{1-\epsilon'}{12\lceil \log_2(2n) \rceil}$-fraction of the optimum solution (recall that we scaled the profits and lost a $(1-\epsilon')$ factor at the start of the algorithm). For fixed constants ϵ, ϵ', the solution found is within an $O(\log n)$ factor of the optimum.

4. RING GRAPHS: WHEN THE GRAPH IS A CYCLE

UFP can be solved approximately using the algorithm for line graphs. The following approach was observed in [Chakrabarti et al. 2002]. Consider an edge e in the cycle with the smallest capacity c_e and partition the tasks used in an optimum solution, say T, into two groups. Group 1 is the collection of tasks that are routed along edge e and group 2 is the collection of tasks which are not routed along edge e. Say the total profit of these groups is, respectively, OPT_1 and OPT_2.

We can approximate OPT_1 within $(1+\epsilon)$ by using the known PTAS for Knapsack [Ibarra and Kim 1975]. For each task i with demand d_i and profit p_i, we create an item for the Knapsack with size d_i and value p_i. Any feasible packing to the Knapsack instance maps directly to a feasible solution for UFP on the cycle by simply routing all tasks whose corresponding Knapsack item is packed. These tasks are routed along the route using edge e. Since all tasks in this solution use edge e and e has the minimum capacity over all edges, then surely any other edge cannot have its capacity constraint violated.

For approximating OPT_2, notice simply that the tasks in group 2 (which are not routed across e) correspond to a feasible solution to the UFP problem on the line obtained by deleting edge e. So, we can approximate OPT_2 within a factor $O(\log n)$ using the UFP approximation algorithm described in this paper. Thus, we get an $O(\log n)$-approximation to UFP on cycles by taking the best of our two approximations to OPT_1 and OPT_2. To summarize,

THEOREM 4.1. *There is an $O(\log n)$-approximation for UFP on rings.*

5. CONCLUDING REMARKS

Obtaining an algorithm with approximation ratio $o(\log n)$ seem to require a more clever way to deal with the interdependencies of the tasks than simply grouping them into intersecting instances. There are instances which require $\Omega(\log n)$ groups to partition the tasks into independent intersecting instances. Consider the following example. Fix a value k and create $2^k - 1$ tasks. There are 2^i tasks of length 2^{k-i} for each $0 \le i < k$. Arrange all of the tasks on a line of length 2^k so that for each fixed $0 \le i < k$, the union of all 2^i tasks of length 2^{k-i} spans the entire line. Call this instance I_k (see figure 4 for an illustration). It is easy to show that at least k partitions are required to group the tasks into independent intersecting instances.

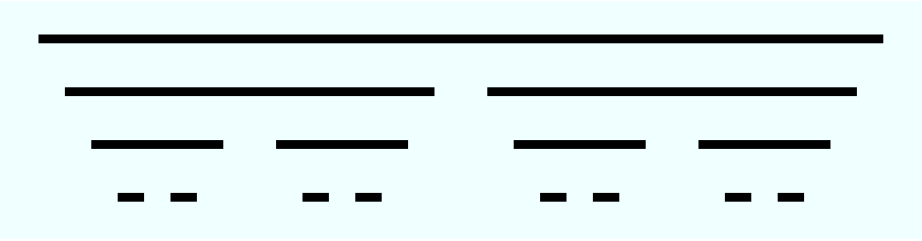

Fig. 4. A sketch of an instance requiring $\Omega(\log n)$ partitions into collections of independent intersecting instances.

LEMMA 5.1. *Any partition of the tasks into collections of independent intersecting instances requires at least k sets.*

PROOF. By induction on k with $k = 1$ being apparent. For the inductive step, consider any partitioning of the tasks in instance I_{k+1} into independent intersecting instances. Let P be the set in the partition that includes the single task of length 2^k. Notice then that P cannot contain two tasks that are on opposite sides of the point 2^k. Say, without loss of generality, that P contains no task to the left of point 2^k. The collection of all tasks to the left of point 2^k are from a copy of instance I_k so at least k groups different than P are required to partition this copy of I_k into a collection of independent intersecting instances. Thus, the total number of groups that partition I_{k+1} is at least $k + 1$. □

The main open problem left over from our work was determining if UFP on line graphs could be approximated within constant factors in polynomial time without any extra assumptions. This has been answered positively with a $(7 + \epsilon)$-approximation for any constant $\epsilon > 0$ [Bonsma et al. 2011]. The next step seems to be determining if the problem admits a PTAS (or even quasi-PTAS) without any further assumptions as the tightest known lower bound is strong NP-hardness [Bonsma et al. 2011]. On the other hand, it could be that general instances of UFP on line graphs is APX-hard.

There is an $O(\log^2 n)$-approximation for trees [Chekuri et al. 2009] which loses one $O(\log n)$ factor essentially due to the same reason we lose $O(\log n)$ in our ratio (a grouping argument). Can the approximation for trees be reduced to $O(\log n)$ to match the ratio we obtain for paths? As noted before, the best hardness for trees is APX-hardness and there is a Quasi-PTAS for trees under the assumptions of a quasi-polynomial bound on the capacities as well as a polylogarithmic bound on the number of leaves [Arackaparambil et al. 2009].

REFERENCES

ANDREWS, M., CHUZHOY, J., KHANNA, S., AND ZHANG, L. 2005. 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, Washington, DC, USA, 226–244.

ANDREWS, M. AND ZHANG, L. 2006. Logarithmic hardness of the undirected edge-disjoint paths problem. *J. ACM 53*, 745–761.

ARACKAPARAMBIL, A., CHAKRABARTI, A., AND HUANG, C. 2009. Approximability of unsplittable flow on trees. Dartmouth Technical Report TR2009-642.

AZAR, Y. AND REGEV, O. 2001. Strongly polynomial algorithms for the unsplittable flow problem. In *In Proceedings of the 8th Conference on Integer Programming and Combinatorial Optimization*. 15–29.

BANSAL, N., CHAKRABARTI, A., EPSTEIN, A., AND SCHIEBER, B. 2006. A quasi-ptas for unsplittable flow on line graphs. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*. STOC '06. ACM, New York, NY, USA, 721–729.

BANSAL, N., FRIGGSTAD, Z., KHANDEKAR, R., AND SALAVATIPOUR, M. 2009. A logarithmic approximation for unsplittable flow on line graphs. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on*

Discrete Algorithms. SODA '09. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 702–709.

BAR-NOY, A., BAR-YEHUDA, R., FREUND, A., NAOR, J., AND SCHIEBER, B. 2001. A unified approach to approximating resource allocation and scheduling. *J. ACM 48*, 1069–1090.

BAVEJA, A. AND SRINIVASAN, A. 2000. Approximation algorithms for disjoint paths and related routing and packing problems. *Mathematics of Operations Research 25*, 2000.

BONSMA, P., SCHULZ, J., AND WIESE, A. 2011. A constant factor approximation algorithm for unsplittable flow on paths. In *FOCS*, R. Ostrovsky, Ed. IEEE, 47–56.

CALINESCU, G., CHAKRABARTI, A., KARLOFF, H., AND RABANI, Y. 2011. An improved approximation algorithm for resource allocation. *ACM Trans. Algorithms 7*, 48:1–48:7.

CHAKRABARTI, A., CHEKURI, C., KUMAR, A., AND GUPTA, A. 2002. Approximation algorithms for the unsplittable flow problem. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*. APPROX '02. Springer-Verlag, London, UK, UK, 51–66.

CHEKURI, C., ENE, A., AND KORULA, N. 2009. Unsplittable flow in paths and trees and column-restricted packing integer programs. In *Proceedings of the 12th International Workshop and 13th International Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. APPROX '09 / RANDOM '09. Springer-Verlag, Berlin, Heidelberg, 42–55.

CHEKURI, C., KHANNA, S., AND SHEPHERD, B. 2006. An o(n) approximation and integrality gap for disjoint paths and unsplittable flow. *Theory of Computing 2*, 2006.

CHEKURI, C., MYDLARZ, M., AND SHEPHERD, F. B. 2007. Multicommodity demand flow in a tree and packing integer programs. *ACM Trans. Algorithms 3*.

FORTUNE, S., HOPCROFT, J., AND WYLLIE, J. 1978. The directed subgraph homeomorphism problem. Tech. rep., Ithaca, NY, USA.

FRIEZE, A. 2000. Edge-disjoint paths in expander graphs. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*. SODA '00. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 717–725.

GARG, N., VAZIRANI, V., AND YANNAKAKIS, M. 1997. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica 18*, 3–20. 10.1007/BF02523685.

GURUSWAMI, V., KHANNA, S., RAJARAMAN, R., SHEPHERD, B., AND YANNAKAKIS, M. 2003. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. *J. Comput. Syst. Sci. 67*, 473–496.

IBARRA, O. AND KIM, C. 1975. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM 22*, 463–468.

KLEINBERG, J. 1996. Approximation algorithms for disjoint paths problems. Ph.D. thesis, MIT.

KLEINBERG, J. AND RUBINFELD, R. 1996. Short paths in expander graphs. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, Washington, DC, USA, 86–.

KOLMAN, P. AND SCHEIDELER, C. 2006. Improved bounds for the unsplittable flow problem. *J. Algorithms 61*, 20–44.

PHILLIPS, A., UMA, R., AND WEIN, J. 2000. Off-line admission control for general scheduling problems. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*. SODA '00. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 879–888.

SRINIVASAN, A. 1997. Improved approximations for edge-disjoint paths, unsplittable flow, and related routing problems. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, Washington, DC, USA, 416–.

VAZIRANI, V. 2003. *Approximation Algorithms*. Springer.

Received February 2007; revised March 2009; accepted June 2009