Improved Algorithms for Resource Allocation under Varying Capacity*

Venkatesan T. Chakaravarthy[1], Anamitra R. Choudhury[1], Shalmoli Gupta[2,**], Sambuddha Roy[3,**], and Yogish Sabharwal[1]

[1] IBM Research - India
{vechakra,anamchou,ysabharwal}@in.ibm.com
[2] University of Illinois at Urbana-Champaign, USA
shalmoli@gmail.com
[3] Amazon, Bangalore, India
shombuddha@gmail.com

Abstract. We consider the problem of scheduling a set of jobs on a system that offers certain resource, wherein the amount of resource offered varies over time. For each job, the input specifies a set of possible scheduling instances, where each instance is given by starting time, ending time, profit and resource requirement. A feasible solution selects a subset of job instances such that at any timeslot, the total requirement by the chosen instances does not exceed the resource available at that timeslot, and at most one instance is chosen for each job. The above problem falls under the well-studied framework of unsplittable flow problem (UFP) on line. The generalized notion of scheduling possibilities captures the standard setting concerned with release times and deadlines. We present improved algorithms based on the primal-dual paradigm, where the improvements are in terms of approximation ratio, running time and simplicity.

1 Introduction

We study the classical scheduling setting of *unsplittable flow problem on line* (UFP). Consider a system offering a certain resource as a service for executing jobs. The total amount of the resource offered by the system may be different at different points of time. Each job is specified as an interval consisting of a starting time and an ending time, and requires a particular amount of the resource for its execution. A feasible solution selects a subset of jobs for execution such that at any point of time, the total amount of resource requirement does not exceed the total amount of the resource available at that time point. Each job is associated with a profit and the objective is to maximize the aggregate profit of the scheduled jobs.

The problem is applicable in a variety of settings based on the resource under consideration, examples of which include computational nodes, storage, electricity and network bandwidth. We refer to prior work for real-life applications of

* Full version of the paper is available as Arxiv preprint.
** Work was done while the author was at IBM Research - India

A. Schulz and D. Wagner (Eds.): ESA 2014, LNCS 8737, pp. 222–234, 2014.

the above job selection problem in parallel/distributed computing and network management [22]. The terminology "unsplittable flow" arises from a more generic graph theoretic framework and the above scheduling problem corresponds to the case, wherein the graph is simply a path. We refer to the survey by Kolliopoulos [24] for a discussion on the general graph theoretic framework.

In the setting considered so far, each job is specified by a single interval where it must be scheduled. Consider a more general scenario where each job specifies a set of possible time intervals and the job may be scheduled in any one of those intervals. In other words, each job can be a viewed as a set (or *bag*) of *job instances* of which at most one can be selected for execution. We allow different instances of the same job to have different resource requirements, processing times (interval length) and profits. The above generalization captures a variety of scenarios. For example, consider the standard setting where time is divided into discrete timeslots and each job is specified by a processing time and a window consisting of release time and deadline. A job with release time r and deadline d, and processing time p can be modelled as a bag containing $(d - r - p + 1)$ instances corresponding to the integer intervals of length p lying between r and d. Motivated by such applications, the `UFP` *problem with bag constraints* (`BagUFP`) has been well-studied. The problem is formally defined next.

`BagUFP` **- Problem Definition:** We assume that time is divided into discrete timeslots $1, 2, \ldots, T$ and let $\mathcal{T} = \{1, 2, \ldots, T\}$ denote the set of all timeslots. For each timeslot t, the input specifies an integer $c(t)$, which is the *capacity* available at the timeslot t. The input consists of a set of jobs $\mathcal{J}$. Each job $a \in \mathcal{J}$ consists of a set of *job instances* of which at most one can be selected for execution. Each job instance u is associated with a starting timeslot $s(u)$, an ending timeslot $e(u)$, a *demand* $h(u)$ and a profit $p(u)$. The interval $[s(u), e(u)]$ is called the *span* of u.

Let $\mathcal{U}$ denote the set of all job instances (over all jobs) and let n be the total number of job instances. Given a set of job instances $X \subseteq \mathcal{U}$, let $p(X)$ denote the cumulative profit $\sum_{u \in X} p(u)$. A job instance $u \in \mathcal{U}$ is said to be *active* at a timeslot t, if t belongs to the range $[s(u), e(u)]$; this is denoted using the notation $u \sim t$. A feasible solution is a set of job instances $S \subseteq \mathcal{U}$ such that the following two constraints are satisfied. The first constraint (called the *capacity constraint*) enforces that for any timeslot t, the cumulative demand of job instances in S active at the timeslot t is at most the capacity $c(t)$ available at t: $\sum_{u \in S \,:\, u \sim t} h(u) \leq c(t)$. The second constraint (called the *bag constraint*) requires that at most one instance is picked from each job. The problem is to find a feasible solution S having the maximum profit $p(S)$. □

Remark. Using preprocessing, we can modify the input capacities suitably so that only the timeslots wherein some job instance starts or finishes is of relevance and the other timeslots can be ignored (see [6]). In the rest of the paper, without loss of generality, we assume that the number of timeslots is at most $2n$.

Special Cases of `BagUFP`**:** Prior work has addressed two important restrictions of the `BagUFP` problem.

- *Single job instance:* In this setting, each job has exactly one job instance; namely, the setting does not involve the bag constraint.
- *No-bottleneck Assumption (NBA):* In this setting, we assume that the maximum demand of any job instance is at most the minimum capacity available, i.e., $h_{\max} \leq c_{\min}$, where $h_{\max} = \max_{u \in \mathcal{U}} h(u)$ and $c_{\min} = \min_{t \in \mathcal{T}} c(t)$.

The NBA setting is well-studied and arises in scenarios wherein the system capacity is larger than the demand of any individual job instance.

By considering the combinations of the two restrictions, we get four different special cases of the problem: (1) `BagUFP` - the most general case, where neither restriction applies; (2) `UFP` - assumes that every job has only one job instance, but does not impose NBA; (3) `BagNbaUFP` - requires NBA, but allows each job to have arbitrary number of job instances; (4) `NbaUFP` - the most specialized case that imposes both the restrictions.

Prior Work: Consider the simplest special case, where each job has only one instance and furthermore, all the job instances have unit demand and all the timeslots offer unit capacity. This is the same as the classical maximum weight independent set problem on interval graphs, which can easily be solved optimally via dynamic programming.

Spieksma [26] considered the above problem along with bag constraints, under the name *weighted job interval selection problem* (`WJISP`). He showed that the problem is NP-hard and APX-hard. Bar-Noy et al. [8] and independently, Berman and Dasgupta [9] presented 2-approximation algorithms. Both these algorithms are based on the local ratio technique. For the unweighted version (wherein all the job instances have unit profit), Chuzhoy et al. [20] presented an algorithm with an approximation ratio of 1.582.

Calinescu et al. [11] studied the case where each job has only one instance and the capacity offered is uniform across all timeslots (however, the instances can have arbitrary demands). They presented a 3-approximation algorithm via the LP-rounding technique. For the above setting with bag constraints, Bar-Noy et al. [7] designed a 5-approximation algorithm using the local ratio technique.

Let us now look at non-uniform capacity setting, viz `NbaUFP`, `UFP`, `BagNbaUFP` and `BagUFP`. For the simplest case of `NbaUFP`, Chakrabarti et al. [16] provided the first constant factor approximation algorithm. Subsequently, Chekuri et al. [19] improved the ratio to $2 + \epsilon$ (here and in the rest of the paper, ϵ would refer to a constant $\epsilon > 0$). Relaxing the NBA assumption, Chakrabarti et al. [16] also presented an algorithm for `UFP`, with an approximation ratio of $O(\log \frac{h_{\max}}{h_{\min}})$, where $h_{\max}$ and $h_{\min}$ are the maximum and minimum demands. For the same problem, Bansal et al. [6] presented an $O(\log n)$-approximation algorithm. In a different paper, Bansal et al. [5] obtained a QPTAS. Recently Bonsma et al. [10] gave the first constant factor polynomial time algorithm; their algorithm achieves an approximation factor of $7 + \epsilon$. Subsequently, the ratio was improved to $2 + \epsilon$

by Anagnostopoulos et al. [4]. The above algorithms are based on sophisticated LP-rounding and dynamic programming strategies.

Chakaravarthy et al. [14] studied the notion of bag constraints and devised an algorithm for BagUFP with an approximation ratio of $O(\log \frac{c_{\max}}{c_{\min}})$, where $c_{\max}$ and $c_{\min}$ are the maximum and minimum capacities, respectively. It remains an interesting open question to design a constant factor approximation algorithm for the BagUFP problem. However, this has been achieved under the NBA assumption. Chakaravarthy et al. [13] presented a 120-approximation algorithm for the BagNbaUFP problem, via a reduction from the non-uniform capacity setting to the uniform capacity setting. Subsequently, Elbassioni et al. [21] improved the factor to 65 using LP-rounding techniques.

The BagUFP problem has also been studied under distributed models and constant factor approximation algorithms are known in the uniform capacity setting, and logarithmic factor approximations in the non-uniform capacity setting [25,15,12]. These algorithms are based on the primal-dual paradigm and they also apply to the parallel setting.

Our Results: In this paper, we present improved algorithms for the BagUFP and its special cases. The main tool used in our work is the primal-dual paradigm (or equivalently the local ratio method), leading to simpler algorithms which provide improvements in terms of approximation ratio and running time. In contrast, prior work on the non-uniform setting predominantly use sophisticated LP rounding and dynamic programming approaches. Furthermore, prior work [12,15,25] has shown that the primal-dual method is more suitable for the parallel/distributed models, and so, the procedures developed in this paper may be adaptable for these environments. We next state the main results of the paper.

- A 17-approximation algorithm for BagNbaUFP problem.
- An $O(\log n)$-approximation algorithm for the BagUFP problem.

Both the algorithms are based on the primal-dual method and run in time $O(n^2)$. The previously best known approximation ratios for the above two problems are 65 [21] and $O(\log \frac{c_{max}}{c_{min}})$ [14], respectively. The second ratio can be as high as $O(n)$ in the worst case. Furthermore, our algorithms are also more efficient in terms of running time; both the previous algorithms go via LP-rounding and need to solve linear programs.

The above two main results deal with the versions having the bag constraint. The procedures developed as part of these results also provide an interesting improvement for the versions devoid of the constraint. Recall that for the UFP problem, Bonsma et al. [10] presented a $(7+\epsilon)$-approximation algorithm, which was subsequently improved to $(2+\epsilon)$ by Anagnostopoulos [4]. Both these algorithms run in polynomial time, but the exponent of the polynomial is very high. Bonsma et al. addressed the issue by presenting another algorithm having a faster running time of $O(n^4)$, but with an increased approximation ratio of $(25+\epsilon)$. The above algorithm has two components based on LP-rounding and dynamic programming, respectively. Of these, the first component can be replaced by one of our procedures yielding a simpler algorithm with the same running time,

but with a better approximation ratio of 13. We present a 13-approximation algorithm for the UFP problem with a running time of $O(n^4)$.

Finally, consider NbaUFP, the most restricted special case. Chakrabarti et al. [16] designed the first constant factor approximation algorithm for this problem via rounding a natural LP. In that context, they raised the question of devising such an algorithm based on the primal-dual method. Our algorithm for BagNbaUFP answers this question affirmatively.

2 Overview

In this section, we provide an overview of our algorithms, highlighting the main components and place them in the context of prior work. Most prior work on BagUFP and its variants go via classifying the job instances into two categories based on their demands. Consider any job instance $u \in \mathcal{U}$. Among all timeslots in the span of u, let t be any timeslot having the minimum capacity (breaking ties arbitrarily). The timeslot t is called the *bottleneck timeslot* for u (denoted by $\mathtt{bt}(u)$) and its capacity *bottleneck capacity* for u (denoted by $\mathtt{bc}(u)$). Fix any constant $0 < \gamma \leqslant 1$. We say that the job instance u is γ-*small*, if $h(u) \leqslant \gamma \mathtt{bc}(u)$; otherwise, u is said to be γ-*large*. For the case where $\gamma = 1/2$, we shall drop the prefix and simply write "small" and "large" to mean 1/2-small and 1/2-large job instances, respectively.

Let Opt denote the optimal solution. Let $\mathcal{U}_s$ and $\mathcal{U}_l$ denote the set of all small and large job instances, respectively. Let Opt_s and Opt_l denote the optimal solution considering only the small and large job instances, respectively. We shall design two procedures that would produce solutions S_s and S_l such that S_s is an f_1-approximation to Opt_s and S_l is an f_2-approximation to Opt_l, for some $f_1, f_2 \geqslant 1$. The best of the two solutions is taken to be the final solution S. It is easy to see that S is an $(f_1 + f_2)$-approximation to Opt.

Given the above aggregation result, we consider the small and the large job instances separately. The core technical component of the paper is a simple and fast primal-dual procedure for handling the small job instances, while guaranteeing a good approximation ratio.

Lemma 1 (PD-Small)**.** *Consider the* BagUFP *problem. There exists a procedure that considers only the small job instances and outputs a solution $S \subseteq \mathcal{U}_s$ such that $p(\mathrm{Opt}_s) \leqslant 9 \cdot p(S)$. The running time is $O(n^2)$.*

The PD-Small lemma is proved in Section 3. Here we highlight certain key aspects of the procedure given by the lemma. For the sake of clarity, we have stated the lemma for the case of $\gamma = 1/2$. However, it can be extended for any $\gamma > 0$, to derive an algorithm for handling γ-small job instances having an approximation ratio of $1 + \frac{4}{1-\gamma}$. Prior work on BagNbaUFP [21] and NbaUFP [19,18] also provide procedures for handling γ-small job instances. However, these procedures yield a good approximation ratio only when γ is set to a small value. In contrast, the PD-Small lemma achieves good approximation factors even for large values of γ (such as $\gamma = 1/2$). The advantage is that the complementary

problem of handling γ-large job instances can be solved more efficiently and with better approximation ratios, leading to improved algorithms for BagUFP.

We note that the PD-Small lemma applies to the general BagUFP problem and does not require the NBA assumption. Our next goal is to handle the large job instances. For this purpose, we shall employ two different procedures, one for the general case and a second one for the special case where NBA applies. The lemma below deals with the general case.

Lemma 2. *Consider the* BagUFP *problem. There exists a procedure that considers only the large job instances and outputs a solution* $S \subseteq \mathcal{U}_l$ *such that* $p(\mathrm{Opt}_l) \leqslant 16 \lceil \log 2n \rceil p(S)$. *The procedure runs in time* $O(n^2)$.

The above procedure exploits a combinatorial lemma regarding large job instances, due to Bonsma et al. [10], that establishes a connection to the *Maximum Weight Independent Set of Rectangles* (MWISR) problem: given a set of rectangles with associated profits, find the maximum profit subset of non-overlapping rectangles [2,23,17,1]. For our purposes, we consider a generalization involving bag constraints and present a $(4\lceil \log 2n \rceil)$-approximation algorithm running in time $O(n^2)$, which may be of independent interest. Our algorithm goes via the notion of sequential k-independent graphs, studied by Akcoglu et al. [3], and Ye and Borodin [27]. Lemma 2 is proved in the full version.

Combining Lemma 2 with PD-Small lemma, we can get an $(9 + 16\lceil \log 2n \rceil)$-approximation to the overall optimal solution, establishing the following result.

Theorem 1. *There exists an* $O(\log n)$*-approximation algorithm for the* BagUFP *problem having running time of* $O(n^2)$.

The above result improves the previously best known approximation ratio of $O(\log \frac{c_{\max}}{c_{\min}})$ [14]. Obtaining a constant factor approximation algorithm for BagUFP remains an open question. The main issue arises in the handling of large job instances. However, prior work has shown that in the NBA setting, the large job instances can be handled via a simple reduction to the WJISP problem (see [16,19,21]). The WJISP problem can be approximated with a factor of 2 via the primal-dual method [7,9].

Lemma 3 ([21,7]). *There exists a procedure for* BagNbaUFP *that considers only the* γ*-large job instances and outputs a solution* $S \subseteq \mathcal{U}_l$ *such that* $p(\mathrm{Opt}_l) \leqslant f \cdot p(S)$, *where* $f = \frac{4}{\gamma}(\frac{1}{\gamma} - 1)$. *The procedure runs in time* $O(n \log n)$.

For the setting of $\gamma = 1/2$, the above lemma yields an 8-approximation procedure. Combining this with PD-Small lemma, we get an overall approximation ratio of 17 for the BagNbaUFP problem, improving upon the previously best known approximation ratio of 65 [21].

Theorem 2. *There exists an algorithm for the* BagNbaUFP *problem having an approximation ratio of* 17. *The algorithm runs in time* $O(n^2)$.

The PD-Small lemma provides an interesting corollary for the UFP problem. Bonsma et al. [10] devised a $(7+\epsilon)$-approximation algorithm running in polynomial time, albeit with a prohibitively large exponent in the polynomial. However, they showed that the running time can be improved to $O(n^4)$, at the cost of increasing the approximation ratio to $(25+\epsilon)$. Their algorithm also employs the strategy of classifying the input into small and large job instances, of which the small job instances are handled via a complex procedure based on randomized rounding. For the case of large job instances, they present a procedure that achieves an approximation ratio of $2/\gamma$, where γ is the largeness parameter; the procedure runs in time $O(n^4)$. We can obtain an alternative algorithm for UFP by employing PD-Small lemma in place of the former procedure. Setting $\gamma = 1/2$, we get an approximation ratio of 13.

Theorem 3. *There exists a* 13*-approximation algorithm for* UFP *running in time* $O(n^4)$*.*

3 Small Job Instances

Here, we establish Lemma 1 by presenting a 9-approximation algorithm for BagUFP on small job instances. We ignore all the large job instances and assume that the input set $\mathcal{U}$ consists only of small job instances. The algorithm is based on the primal-dual paradigm and builds on prior work on on distributed algorithms for the UFP problem [25,15,12]. However, the prior algorithms either deal with the simpler uniform capacity setting (wherein the capacity across all the timeslots is assumed to be the same) or provide logarithmic approximation factor. All the above primal-dual algorithms consider the job instance in a particular order and the main feature of our approach is to employ a more appropriate ordering. Our analysis exploits the new ordering in a crucial manner leading to constant factor approximations for the generic non-uniform setting.

The LP relaxation and its dual are presented next.

$$\max \sum_{u \in \mathcal{U}} x(u)p(u)$$

$$(\forall t \in \mathcal{T}) \quad \sum_{u \,:\, u \sim t} h(u)x(u) \leqslant c(t)$$

$$(\forall J \in \mathcal{J}) \quad \sum_{u \in J} x(u) \leqslant 1$$

$$\min \sum_{J \in \mathcal{J}} \alpha(J) + \sum_{t \in \mathcal{T}} c(t)\beta(t)$$

$$\alpha(J_u) + h(u) \sum_{t \,:\, u \sim t} \beta(t) \geqslant p(u) \qquad (\forall u \in \mathcal{U})$$

The primal includes a variable $x(u)$ for each job instance $u \in \mathcal{U}$. The capacity and the bag constraints are enforced next. The dual includes a variable $\alpha(J)$ corresponding to the bag constraint of J, for each job J. Moreover, for each timeslot $t \in \mathcal{T}$, the dual includes variable $\beta(t)$ corresponding to the capacity constraint at t. For each job instance u, we include a constraint corresponding to the primal variable $x(u)$, which we call the *dual constraint of* u. For a job

instance u, let J_u denote the job to which the instance u belongs. All the primal and dual variables are non-negative.

3.1 Algorithm

Our primal-dual algorithm uses a two-phase framework consisting of a forward phase and a reverse phase. The forward phase would construct a set of job instances $R \subseteq \mathcal{U}$ and a dual feasible solution $\alpha(\cdot)$ and $\beta(\cdot)$. The set R may not be a feasible solution. The reverse phase would delete certain job instances from R and construct a feasible solution $S \subseteq R$.

Forward Phase: We start by initializing all the dual variables to be zero and taking R to be the empty set. The algorithm would process the job instances in particular order and raise the dual variables in an appropriate manner. The ordering is cardinal to our algorithm in that it dictates the performance guarantee. We order the job instances in the decreasing order of their bottleneck capacities $\mathtt{bc}(u)$ and among the job instances having the same bottleneck capacity, the ordering is determined in the increasing order of ending timeslots (breaking ties arbitrarily). We denote the above ordering as σ.

The forward phase works iteratively, where the ith iteration would process the ith job instance in the ordering σ. Consider an iteration and let u be the job instance under processing. We check if the dual constraint of u is already satisfied and if so, we simply proceed to the next iteration. Otherwise, we shall raise certain dual variables suitably so that the constraint is satisfied, as described below. We first determine the *slackness* of the constraint, which is the difference between the RHS and the LHS of the constraint:

$$\text{slack}(u) = p(u) - \left(\alpha(J_u) + h(u) \sum_{t \,:\, u \sim t} \beta(t) \right). \tag{1}$$

We next select two specific timeslots t_ℓ and t_r from the span of u, as follows. Consider all the timeslots in the span of u having capacity at most $2\mathtt{bc}(u)$ and let t_ℓ be the left-most timeslot among them. Similarly, let t_r be the right-most timeslot among the timeslots satisfying the above property. Intuitively the span in between the timeslots t_ℓ and t_r is the essential span of the job instance u; beyond these timeslots, there is enough capacity. Call t_ℓ and t_r as the *left* and *right critical* timeslots of u, respectively.

We shall suitably raise the dual variables $\alpha(J_u)$, $\beta(t_\ell)$ and $\beta(t_r)$ so that the dual constraint is satisfied. Intuitively, we would like to satisfy two goals: (a) The dual objective value is not raised by much (since the dual is a minimization problem); (b) All the critical timeslots contribute an equal amount of increase in the dual objective value. With the above goals in mind, the dual variables for the critical timeslots are raised inversely proportional to the capacities at those timeslots, conforming to the intuition that timeslots with higher capacities are *less* critical. We choose a suitable value $\delta(u)$ and raise $\alpha(J_u)$ by $\delta(u)$, $\beta(t_\ell)$ by $4\frac{\delta(u)}{c(t_\ell)}$ and $\beta(t_r)$ by $4\frac{\delta(u)}{c(t_r)}$. The amount $\delta(u)$ is calculated so that the slack

vanishes and the constraint becomes satisfied tightly i.e., LHS becomes equal to RHS. Namely, compute $\delta(u)$ satisfying the following equation:

$$\delta(u) \cdot \left(1 + 4h(u)\left[\frac{1}{c(t_\ell)} + \frac{1}{c(t_r)}\right]\right) = \text{slack}(u). \tag{2}$$

The job instance u is added to the set R. This completes an iteration of the first phase. We say that all the job instances in R are *raised*.

Reverse Phase: We consider the job instances in reverse order in which they were inserted into R and construct the solution S as follows. In any iteration of this phase, we look at the next job instance u (in the reverse order) and add u to S if doing so does not violate the capacity or the bag constraints. This phase continues until all of the job instances in R have been considered. The algorithm outputs the (feasible) solution S. This completes the description of the algorithm. A pseudocode can be found in the full version.

3.2 Analysis

Let us calculate the objective value of the dual solution constructed by the forward phase, denoted $val(\alpha, \beta)$ in terms of $\delta(\cdot)$. For any job instance $u \in R$, the dual variable $\alpha(J_u)$ is raised by $\delta(u)$ and this increases the dual objective value by $\delta(u)$. Similarly, we raise the dual variables corresponding to the two critical timeslots of u; namely, $\beta(t_\ell)$ is raised by $\frac{4\delta(u)}{c(t_\ell)}$ and $\beta(t_r)$ is raised by $\frac{4\delta(u)}{c(t_r)}$. Therefore, for each job instance $u \in R$, the dual objective value raises by an amount $9\delta(u)$. It follows that $val(\alpha, \beta) = 9 \sum_{u \in R} \delta(u)$ The lemma below provides a comparable lowerbound on the profit of the output solution S.

Lemma 4. *We have* $p(S) \geqslant \sum_{u \in R} \delta(u)$.

The lemma implies that $val(\alpha, \beta) \leq 9 \cdot p(S)$. Coupled with the weak duality theorem, we get that S is a 9-approximation to the optimal solution.

We proceed to Lemma 4. We shall associate a suitable quantity $\pi(u)$ with each job instance $u \in R$ such that $\pi(u) \geq \delta(u)$, and their overall sum satisfies $\sum_{u \in R} \pi(u) = p(S)$. Intuitively, $\pi(u)$ is the contribution made by u towards $p(S)$ (irrespective of whether or not u got picked in the final solution S).

For two job instances $u_1, u_2 \in R$, we say that u_1 is a *predecessor* of u_2, if u_1 appears before u_2 in the ordering σ; in this case, u_2 is said to be a *successor* of u_1. For a job instance $u \in R$, let pred(u) and succ(u) denote the set of all predecessors and successors of u, respectively. We consider a job instance as both predecessor and successor of itself.

Consider a job instance $u \in S$. Let LHS(u) be the variable denoting the LHS of the dual constraint of u:

$$\text{LHS}(u) = \alpha(J_u) + h(u) \sum_{t \,:\, u \sim t} \beta(t).$$

The variable LHS(u) would be zero in the beginning of the forward phase and it would keep increasing as the algorithm proceeds. At the end iteration in which u is raised, LHS(u) would be equal to $p(u)$ (since we ensured that the dual constraint of u is satisfied tightly). Thus, by tracking the variable LHS(u), we can derive a formula for $p(u)$ in terms of $\delta(\cdot)$ values of predecessors of u.

Consider a predecessor $u' \in \text{pred}(u)$. When u' is raised, three dual variables are increased, $\alpha(J_{u'})$, $\beta(t_\ell)$ and $\beta(t_r)$, where $J_{u'}$ is the job to which u' belongs, and t_ℓ and t_r are the left and right critical timeslots of u'. These increments will reflect as an increase in LHS(u), if the dual constraint of u also shares one or more of these variables. The increment in LHS(u) corresponding to the above three types are as follows:

- Type 1: If both u' and u belong to the same job, then the increment is $\delta(u')$.
- Type 2: If u is active at t_ℓ, the increment is $\frac{4h(u)\delta(u')}{c(t_\ell)}$.
- Type 3: If u is active at t_r, the increment is $\frac{4h(u)\delta(u')}{c(t_r)}$.

Notice that LHS(u) may increase via more than one type, in which case the total increment would be given by corresponding sum; if none of the cases occur, then the sum would be zero. We call the above sum as the *contribution of u' towards u* and denote it as $\lambda(u', u)$. The sum of contributions made by the predecessors of u yields the value of LHS(u) (as it stood at the end of the iteration in which u was raised), which is the same as $p(u)$.

The above discussion focuses on a job instance and analyzes the contributions made by the predecessors towards the instance. Conversely, we can fix a job instance u and consider the aggregate contribution that u makes towards its successors found in the solution S. We call the above aggregate quantity as the *total contribution* of u and denote it as $\pi(u)$: $\pi(u) = \sum_{u' \in \text{succ}(u) \cap S} \lambda(u, u')$. Notice that the profit $p(S)$ is given by the sum $\sum_{u \in R} \pi(u)$.

The quantity $\pi(u)$ can be computed by considering the three types of contributions discussed earlier. Let J_u be the job to which u belongs, and let t_ℓ and t_r be its left and right critical timeslots. Let $X = S \cap \text{succ}(u)$. Then,

$$\pi(u) = \sum_{u' \in X \,:\, u' \in J_u} \delta(u) + \sum_{u' \in X \,:\, u' \sim t_\ell} \frac{4h(u')\delta(u)}{c(t_\ell)} + \sum_{u' \in X \,:\, u' \sim t_r} \frac{4h(u')\delta(u)}{c(t_r)}$$

We next establish a lowerbound on total contribution of any raised job instance.

Lemma 5. *For any $u \in R$, $\pi(u) \geq \delta(u)$.*

The lowerbound implies Lemma 4. To prove the lowerbound, we fix a job instance $u \in R$ and analyze three cases. The first case is where u is picked in the solution S. In this case, u would contribute $\delta(u)$ towards itself (Type 1) and hence, $\pi(u) \geq \delta(u)$. So, assume that the reverse phase did not pick u for inclusion in S. This means that u could not be added to X, where $X \subseteq S$ is the set of successors of u found in S. The reason is that a bag constraint or a capacity constraint (or both) gets violated when u is added to X. Consider the first scenario, wherein

X contains some job instance u' that belongs to the same job as u. In this case, u would contribute $\delta(u)$ towards u' (Type 1) and hence, $\pi(u) \geq \delta(u)$.

We next analyze the last and the most interesting scenario, wherein the capacity constraint is violated at some timeslot $\widehat{t}$ found in the span of u, i.e.,

$$h(u) + \sum_{u' \in X \,:\, u' \sim \widehat{t}} h(u') > c(\widehat{t}). \tag{3}$$

If there are multiple such timeslots, choose the one having the minimum capacity (breaking ties arbitrarily). This is denoted by $\widehat{t}$ and is called the *conflict timeslot*.

Let $C \subseteq X$ be the set of job instances from X active at $\widehat{t}$; intuitively, C is the set of job instances that conflict with X at $\widehat{t}$ and prevent it from being included in X. Let t_ℓ and t_r be the left and right critical timeslots of u. We next make an important claim regarding any job instance $u' \in C$.

Lemma 6. *Any job instance $u' \in C$ must be active at t_ℓ or t_r (or both).*

The lemma is proved by exploiting the properties of the ordering σ. Intuitively, the argument is that the timeslots in the span of u outside of the range $[t_\ell, t_r]$ have too high a capacity to cause capacity constraint violation and hence, $\widehat{t}$ must lie within the range. Moreover, the ordering also implies that any job instance in C must start before u or end after u. The above two statements put together would imply the lemma. The lemma is proved in the full version.

Here, we assume the lemma and complete the proof of Lemma 5. Let A and B be the job instances in C that are active at t_ℓ and t_r, respectively. The lemma implies that every job instance in C is included in at least one of the two sets. Let us consider the quantity $\pi(u)$ and focus only on the terms corresponding to the job instances found in the two sets:

$$\pi(u) \geq \left(4\delta(u) \sum_{u' \in A} \frac{h(u')}{c(t_\ell)} \right) + \left(4\delta(u) \sum_{u' \in B} \frac{h(u')}{c(t_r)} \right)$$

The capacity at t_ℓ and t_r is at most twice the bottleneck capacity $\mathtt{bc}(u)$. So,

$$\pi(u) \geq \frac{2\delta(u)}{\mathtt{bc}(u)} \sum_{u' \in C} h(u'). \tag{4}$$

Since we are dealing with small job instances, $h(u) \leq \mathtt{bc}(u)/2$, which implies that $h(u) \leq c(\widehat{t})/2$. From (3), we get that

$$\sum_{u' \in C} h(u') \quad \geq \quad c(\widehat{t})/2 \quad \geq \quad \mathtt{bc}(u)/2.$$

Substituting in (4), we get that $\pi(u) \geq \delta(u)$. The proof of Lemma 5 is completed.

We conclude that the algorithm achieves an approximation guarantee of 9. It is not difficult to see that the algorithm can be implemented in time $O(n^2)$. This completes the proof of PD-Small lemma.

References

1. Adamaszek, A., Wiese, A.: Approximation schemes for maximum weight independent set of rectangles. In: FOCS (2013)
2. Agarwal, P., Kreveld, M., Suri, S.: Label placement by maximum independent set in rectangles. Computational Geometry 11(3-4), 209–218 (1998)
3. Akcoglu, K., Aspnes, J., Dasgupta, B., Kao, M.: Opportunity cost algorithms for combinatorial auctions. In: Kontoghiorghes, E., Rustem, B., Siokos, S. (eds.) Applied Optimization: Computational Methods in Decision-Making (2000)
4. Anagnostopoulos, A., Grandoni, F., Leonardi, S., Wiese, A.: A mazing $2 + \epsilon$ approximation for Unsplittable Flow on a Path. In: Proceedings of the Symposium on Discrete Algorithms, SODA 2014 (2014)
5. Bansal, N., Chakrabarti, A., Epstein, A., Schieber, B.: A quasi-PTAS for unsplittable flow on line graphs. In: STOC (2006)
6. Bansal, N., Friggstad, Z., Khandekar, R., Salavatipour, M.: A logarithmic approximation for unsplittable flow on line graphs. In: SODA (2009)
7. Bar-Noy, A., Bar-Yehuda, R., Freund, A., Naor, J., Schieber, B.: A unified approach to approximating resource allocation and scheduling. Journal of the ACM 48(5), 1069–1090 (2001)
8. Bar-Noy, A., Guha, S., Noar, J., Schieber, B.: Approximating the throughput of multiple machines in real-time scheduling. SICOMP 31(2), 331–352 (2001)
9. Berman, P., Dasgupta, B.: Multi-phase algorithms for throughput maximization for real-time scheduling. J. of Comb. Opt. 4, 307–323 (2000)
10. Bonsma, P., Schulz, J., Wiese, A.: A constant factor approximation algorithm for unsplittable flow on paths. In: FOCS (2011)
11. Calinescu, G., Chakrabarti, A., Karloff, H., Rabani, Y.: Improved approximation algorithms for resource allocation. In: Cook, W.J., Schulz, A.S. (eds.) IPCO 2002. LNCS, vol. 2337, pp. 401–414. Springer, Heidelberg (2002)
12. Chakaravarthy, V., Choudhury, A., Roy, S., Sabharwal, Y.: Distributed algorithms for scheduling on line and tree networks with non-uniform bandwidths. In: IPDPS (2013)
13. Chakaravarthy, V., Choudhury, A.R., Sabharwal, Y.: A near-linear time constant factor algorithm for unsplittable flow problem on line with bag constraints. In: FSTTCS (2010)
14. Chakaravarthy, V., Pandit, V., Sabharwal, Y., Seetharam, D.: Varying bandwidth resource allocation problem with bag constraints. In: IPDPS (2010)
15. Chakaravarthy, V., Roy, S., Sabharwal, Y.: Distributed algorithms for scheduling on line and tree networks. In: PODC (2012)
16. Chakrabarti, A., Chekuri, C., Gupta, A., Kumar, A.: Approximation algorithms for the unsplittable flow problem. Algorithmica 47(1), 53–78 (2007)
17. Chalermsook, P., Chuzhoy, J.: Maximum independent set of rectangles. In: SODA (2009)
18. Chekuri, C., Ene, A., Korula, N.: Unsplittable flow in paths and trees and column-restricted packing integer programs. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. LNCS, vol. 5687, pp. 42–55. Springer, Heidelberg (2009)
19. Chekuri, C., Mydlarz, M., Shepherd, F.: Multicommodity demand flow in a tree and packing integer programs. ACM Trans. on Algorithms 3(3) (2007)

20. Chuzhoy, J., Ostrovsky, R., Rabani, Y.: Approximation algorithms for the job interval selection problem and related scheduling problems. In: FOCS (2001)
21. Elbassioni, K., Garg, N., Gupta, D., Kumar, A., Narula, V., Pal, A.: Approximation Algorithms for the Unsplittable Flow Problem on Paths and Trees. In: FSTTCS (2012)
22. Erlebach, T., Spieksma, F.: Interval selection: Applications, algorithms, and lower bounds. J. Algorithms 46(1), 27–53 (2003)
23. Khanna, S., Muthukrishnan, S., Paterson, M.: On approximating rectangle tiling and packing. In: SODA (1998)
24. Kolliopoulos, S.: Edge-disjoint paths and unsplittable flow. In: Gonzalez, T. (ed.) Handbook of Approximation Algorithms and Metaheuristics, Chapman and Hall/CRC (2007)
25. Panconesi, A., Sozio, M.: Fast primal-dual distributed algorithms for scheduling and matching problems. Distributed Computing 22(4), 269–283 (2010)
26. Spieksma, F.: On the approximability of an interval scheduling problem. J. of Scheduling 2, 215–227 (1999)
27. Ye, Y., Borodin, A.: Elimination graphs. ACM Transactions on Algorithms 8(2), 14 (2012)