Scheduling Jobs with Multiple Non-uniform Tasks

Venkatesan T. Chakaravarthy, Anamitra Roy Choudhury, Sambuddha Roy, and Yogish Sabharwal

IBM Research - India, New Delhi
{vechakra,anamchou,sambuddha,ysabharwal}@in.ibm.com

Abstract. This paper considers the problem of maximizing the throughput of jobs wherein each job consists of multiple tasks. Consider a system offering a uniform capacity of a resource (say unit bandwidth). We are given a set of jobs, each consisting of a sequence of at most r tasks. Each task is associated with a window (specified by a release time and a deadline) within which it can be scheduled; each task also has a processing time and a bandwidth requirement. Each job has a profit associated with it. A feasible solution must choose a subset of jobs and schedule all the tasks for these jobs such that at any point of time, the total bandwidth requirement does not exceed the capacity of the resource; furthermore, the schedule must obey the precedence constraints (tasks of a job must be scheduled in order of the input sequence). The goal is to compute the feasible solution having maximum profit.

Prior work has studied the problem without the notion of windows; furthermore, the algorithms presented therein require that the bandwidths of all the tasks of a job are uniform. Under these two restrictions, $O(r)$-approximation algorithms are known. Our main result presents an $O(r)$-approximation algorithm for the general case wherein tasks can have windows and bandwidths of tasks within the same job may be non-uniform.

1 Introduction

Scheduling of jobs arises in diverse areas such as parallel and distributed computing, workforce management and energy management. In particular, consider a compute environment (such as a grid, cloud, etc.) offering resources as a service for executing jobs. The resources offered may be computational nodes, storage, network bandwidth, etc. The aim of the service provider owning the environment is to schedule jobs that maximize its profit subject to the availability of resources. Typically jobs do not require all the resources during their entire execution time and may have different requirements of the resources at different points in time. Suppose that the jobs can specify the time range and the durations during which they require the resources. This enables the service provider to schedule the jobs more optimally, thereby accommodating more jobs as well as increasing their profits. Motivated by this, we consider a setting in which a job is

F. Wolf, B. Mohr, and D. an Mey (Eds.): Euro-Par 2013, LNCS 8097, pp. 90–101, 2013.

decomposed into multiple tasks, where each task specifies the time range, duration and quantity of the resource required. The problem also finds applications in computational biology, multimedia streaming and computational geometry (see Bar-Yehuda and Rawitz [4] for more details). We use the phrase *bandwidth* as a generic term for resources.

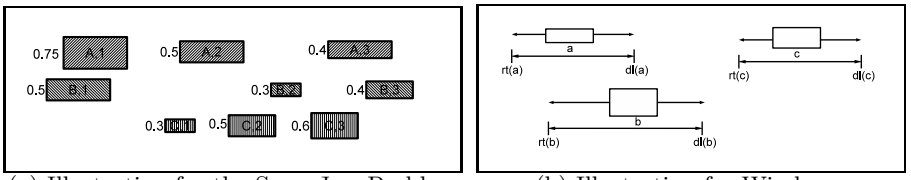
Illustration. Figure 1(a) illustrates the problem. Consider a system offering a uniform bandwidth of one unit. We have three jobs A, B and C, each containing 3 tasks. Each task has a requirement for the bandwidth, as shown in the figure. For example, the three tasks of the job A have requirements $0.75, 0.5$ and 0.4. A feasible solution must select a subset of jobs such that at any point of time, the sum of bandwidth requirements of the scheduled tasks must not exceed the bandwidth offered (i.e., one unit). We see that a feasible solution cannot pick both A and B, because the combined bandwidth requirement of the overlapping tasks $(A, 1)$ and $(B, 1)$ is 1.25. On the other hand, we can see that A and C can be picked together, since the combined bandwidth requirement does not exceed one unit at any point of time.

Problem Statement. Motivated by applications mentioned above, we first define the basic version, the SplitJob problem. Then we discuss a natural generalization of the problem.

Basic SplitJob *Problem:* We assume that time is divided into discrete timeslots $\{1, 2, \ldots, T\}$. Consider a system offering a uniform bandwidth of say 1 unit throughout the span $[1, T]$. The input consists of a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$. Each job $J \in \mathcal{J}$ consists of a sequence of (at most) r tasks; each task is specified by a segment (or interval), given by a starting timeslot and an ending timeslot; Each task $a \in J$ also has a bandwidth requirement or *height*. We require that for any job, the r tasks constituting the job are non-overlapping. Every job $J \in \mathcal{J}$ is associated with a profit $p(J)$. A set of jobs $\mathcal{S} \subseteq \mathcal{J}$ is said to be a *feasible solution*, if at any timeslot $1 \leq t \leq T$, the sum of the heights for all the jobs selected by $\mathcal{S}$ does not exceed 1; we call this the *bandwidth constraint*. The profit of the solution $\mathcal{S}$ is defined to be the sum of profits of the jobs in $\mathcal{S}$. The SplitJob problem is to find a feasible solution $\mathcal{S}$ having the maximum profit.

SplitJob *Problem with Windows:* In the SplitJob problem, each task is specified by a fixed interval where it must be scheduled. However, in realistic applications, a task can specify a window within which it can be executed. To capture these scenarios we define a generalization of the SplitJob problem.

In this setup, each job J is specified by a sequence of tasks $a_1, a_2, \ldots, a_r$. Each input task a is specified by a *window* $[\mathrm{rt}(a), \mathrm{dl}(a)]$, where $\mathrm{rt}(a)$ and $\mathrm{dl}(a)$ are the release time and the deadline for the task, respectively. Each task is also associated with a processing time $\rho(a)$ and a height $h(a)$. The task can be scheduled on any segment of length $\rho(a)$ contained within its window. In addition to choosing a set of jobs, a feasible solution must also decide in which segment to schedule the tasks of the chosen jobs. Apart from satisfying the bandwidth constraint, we also require that the solution must satisfy the precedence constraint: the segment where a_i is scheduled must finish before the segment for a_{i+1} starts



(a) Illustration for the SplitJob Problem (b) Illustration for Windows.

Fig. 1. Illustrations

(in other words, the execution of a_i should end before the execution of a_{i+1} starts). As before, the goal is to choose a feasible solution of maximum profit. We call this the SplitJob problem with windows. Notice that the windows of the tasks of a specific job may overlap, however a feasible solution must choose non-overlapping segments for them.

The SplitJob problem with windows includes as special case the following interesting version of the problem. In the new setup, the windows are associated with jobs, instead of tasks. Each job has a release time and deadline; each task is specified only by a processing time and a height. A task can be scheduled in any segment contained within the window of the job. A feasible solution must respect both the bandwidth constraints and the precedence constraints.

Special Cases. Prior work has addressed the following special cases of the basic SplitJob problem (without windows).

- *Single Task Case:* Here, each job consists of only one task (i.e., $r = 1$).
- *Unit Height Case:* All tasks of all jobs have height 1 (the bandwidth available). In this scenario, no two overlapping jobs can be scheduled.
- *Uniform Height Case:* In this case, for any job $J \in \mathcal{J}$, all the tasks of the job have the same height. Thus, the notion of height can be associated with the job itself, rather than with individual tasks. Note that different jobs are however allowed to have different heights.

All the above special cases also apply to the SplitJob problem with windows.

Prior Work. Bar-Noy et al. [2] studied the case of the single tasks ($r = 1$) and presented a 3-approximation algorithm using the local ratio technique (this algorithm can also handle the concept of windows and the approximation ratio becomes 5). Independently, Calinescu et al. [7] also designed a 3-approximation algorithm, via rounding linear program solutions. The problem has been generalized to the setup where the available bandwidth varies over time and it is known as the unsplittable flow problem on line (UFP), for which constant factor approximation algorithms are known (see [6]).

The unit height case of the basic SplitJob problem has been addressed in the context of finding maximum weight independent sets in r-interval graphs (e.g. [1,5]). Working under this framework, Bar-Yehuda et al. [3] presented a $(2r)$-approximation algorithm; in this context, they introduced the fractional local ratio paradigm. They also proved a hardness result: it is NP-hard to approximate

the problem within a factor of $O(r/\log r)$. Thus, their approximation ratio is near-optimal.

Building on the techniques of Bar-Yehuda et al. [3], Bar-Yehuda and Rawitz [4] studied the uniform case of the basic SplitJob problem (without windows) and derived a $(6r)$-approximation algorithm. Their algorithm also utilizes the fractional local ratio technique.

Our Main Result. To the best of our knowledge, when $r \geq 2$, the prior work does not address two important aspects: (i) the concept of windows; (ii) non-uniform heights (i.e., the tasks of the same job may have different heights). The goal of this paper is to design an algorithm that handles both these aspects.

We present an approximation algorithm for a practically important special case of the problem, where no task requires more than half the bandwidth available; that is for any task a, its height $h(a) \leq 1/2$. Our main result is as follows:

Theorem 1. *There exists a randomized* $(8r)$*-approximation algorithm for the* SplitJob *problem with windows (with non-uniform tasks) when all the input tasks have height at most* $1/2$*. The running time of the algorithm is polynomial in* n*,* T *and* r *(* n *is the number of jobs and* T *is the number of timeslots).*

An interesting question here is whether we can design an algorithm having an constant approximation ratio (independent of the number of tasks r). However, this would imply NP=P, as discussed next. We can show that the basic SplitJob problem (without windows) includes as a special case the multi-dimensional knapsack problem, for which Chekuri and Khanna [9] derived certain hardness results. Using their results, we can prove that it is NP-hard to approximate the SplitJob problem (without windows) within factor of $O(r^{1/3-\epsilon})$, even when all the tasks have height at most $1/2$.

Other Results. We also prove these additional results.

- The main result can be generalized to the case where the task heights are bounded by a fixed constant. For any fixed constant $\alpha < 1$, we derive a randomized algorithm for the case where all the tasks have height at most α and the approximation ratio is $(4r)/(1-\alpha)$ (the main result corresponds to the value $\alpha = 1/2$).
- Our approach can also handle the case of uniform and unit height tasks and the approximation ratios obtained for these cases are $4r$ and $12r$ respectively.
- The algorithms claimed in the main result and the above two scenarios can handle the notion of windows and run in time polynomial in n, T and r. We note that in all the three cases, if we consider the corresponding basic versions without windows, then the algorithm can be made to run in time polynomial in n and r (i.e., the dependency on T can be removed).
- The main result deals with the case where all the tasks have height at most $1/2$. It is an interesting open problem to obtain an $O(r)$-approximation algorithm for the SplitJob problem (with or without windows), where the height of tasks can be arbitrary. In this context, we point out a difficulty in extending our algorithm for this general case. The $(8r)$-approximation

algorithm for the case of small tasks is based on rounding a natural linear program. We prove that this linear program has an integrality gap of $\Omega(2^r)$, even without windows. This shows that solving the open problem must involve a different strategy.

Due to lack of space, we will present only the main result in this paper and defer the details of the other results to the full version of the paper.

Proof Techniques and Discussion. Recall that Bar-Yehuda et al. [3] presented a $(2r)$-approximation algorithm for the scenario where all the tasks have unit height (unit height case). Extending their algorithm, Bar-Yehuda and Rawitz [4] presented a $(6r)$-approximation algorithm for the scenario where all the tasks of a job have the same height (uniform height case). Both these algorithms are based on the fractional local ratio paradigm, which involves rounding a linear program solution using the local ratio technique. Our goal is to design an algorithm than can solve the more general problem having two additional features: (i) the notion of windows; (ii) allow the tasks of the same job to have different heights (non-uniform case). We handle the notion of windows by considering an exponential sized linear program and solving it using a separation oracle. We note that the procedures of Bar-Yehuda et al. [3] and Bar-Yehuda and Rawitz [4] can be extended by incorporating our separation oracle to handle the concept of windows, as long as the tasks have unit or uniform heights, respectively. However, the notion of non-uniform heights poses more interesting challenges. To the best of our efforts, we could not extend their algorithms to handle the non-uniform scenario. In this paper, we overcome the issue by taking a different approach, namely randomized rounding. Thus, at a technical level, the main contribution of this paper is to show that randomized rounding offers an alternative method for dealing with scheduling multi-task jobs and furthermore, this approach can also deal with the case of non-uniform tasks.

Our algorithms are inspired by work of Chakrabarti et al. [8], who study the unsplittable flow problem (UFP) on line. Generalizing the work of Bar-Yehuda et al. [3] suitably for the case of non-uniform heights so as to apply the fractional local ratio technique is left as an interesting open question.

Remark: In the our problems, a job is allowed to have at most r tasks. However, we can assume without loss of generality that every job has exactly r tasks; this can be easily accomplished by introducing dummy tasks. So, in the rest of the paper, we assume that every job has exactly r tasks.

2 Main Result: LP Formulation and Solution

We say that a task a is *small*, if $h(a) \leq 1/2$. Our goal is to establish the main result of the paper, by designing a randomized $(8r)$-approximation algorithm for the special case of the SplitJob problem with windows, wherein all the tasks all small. Meaning, the algorithm outputs a solution $\mathcal{S}$ such that the expected profit of $\mathcal{S}$ is within a factor of $8r$ of the optimum solution $\texttt{Opt}$ (i.e., $\mathbf{E}[p(\mathcal{S})] \geq p(\texttt{Opt})/(8r)$). The algorithm goes via formulating a LP and rounding

the fractional LP optimum solution. In this section, we present the LP formulation and discuss a duality based method for solving it. The rounding procedure is described in the next section. The following notations are useful for this purpose.

Notations. Let $\mathcal{J}$ be the set of n jobs, where each job $J \in \mathcal{J}$ consists of a sequence of r tasks. Each task a is specified by a window $[\mathrm{rt}(a), \mathrm{dl}(a)]$, a processing time $\rho(a)$ and a height $h(a)$. The task a can be scheduled in any *segment* $[s, e]$ of length $\rho(a)$ contained within the window $[\mathrm{rt}(a), \mathrm{dl}(a)]$. For each such segment u, its height is defined to be $h(u) = h(a)$. Such a segment u is said to be *active* at a timeslot t, if $t \in [s, e]$; this is denoted $u \sim t$. Let U be a set of segments (arising from multiple jobs/tasks) and let t be a timeslot. We define $h_t(U)$ to be the sum of heights of all segments from U active at the timeslot t: $h_t(U) = \sum_{u \in U \,:\, u \sim t} h(u)$.

LP Formulation. Let J be a job consisting of a sequence of tasks $a_1, a_2, \ldots, a_r$. For each task a_i with an associated window $[\mathrm{rt}(a_i), \mathrm{dl}(a_i)]$, the number of possible segments is $q(a_i) = \mathrm{dl}(a_i) - \rho(a_i) - \mathrm{rt}(a_i) + 2$. The total number of combinations for choosing segments for all the r tasks of the job J is $q = \Pi_{i=1}^{r} q(a_i)$. For a combination to be valid, it must satisfy the precedence constraint: namely, for $1 \leq i \leq r-1$, the segment chosen for a_i must end before the segment chosen for a_{i+1} starts. Discard the invalid combinations and let $\mathrm{Inst}(J)$ denote the set of remaining valid combinations. The number of valid combinations for the job J is at most T^r, where T is the total number of timeslots. We call each valid combination present in $\mathrm{Inst}(J)$ as a *job instance* of J. Each such job instance consists of a set of r segments each specified by a start time, end time and a height such that the segments are non-overlapping. Let $\mathcal{I}$ denote the union of job instances over all the jobs. For a job J and job instance $I \in \mathrm{Inst}(J)$, we define the profit of I to be $p(I) = p(J)$. We say that a job instance $I \in \mathcal{I}$ is *active* at a timeslot t, if one of its segments is active at the timeslot; we denote this as $I \sim t$. In this case, let $h_t(I)$ denote the height of the (unique) segment of I active at the timeslot (we call this the height of I at the timeslot t).

$$\max \quad \sum_{I \in \mathcal{I}} y(I) \cdot p(I)$$

$$\sum_{I \in \mathcal{I} \,:\, I \sim t} y(I) h_t(I) \leq 1 \qquad \text{for all time-slots } 1 \leq t \leq T \tag{1}$$

$$\sum_{I \in \mathrm{Inst}(J)} y(I) \leq 1 \qquad \text{for all jobs } J \in \mathcal{J} \tag{2}$$

$$y(I) \in \{0, 1\} \qquad \text{for all jobs } J \in \mathcal{J}$$

The integer program (IP) given above arises from the following equivalent formulation of a feasible solution. A feasible solution selects a subset of instances $\mathcal{F} \subseteq \mathcal{I}$ such that the following requirements are satisfied: (i) Bandwidth constraint: for any timeslot t, $\sum_{I \in \mathcal{F} \,:\, I \sim t} h_t(I) \leq 1$; (ii) For any job J, at most one job instance from $\mathrm{Inst}(J)$ is included in $\mathcal{F}$. Our goal is to choose a feasible

solution having the maximum profit. In the IP, for each instance $I \in \mathcal{I}$, we introduce a variable $y(I)$ that denotes whether or not I is chosen in the solution. Constraints (1) and (2) encode the above requirements. We get a linear program by relaxing the integrality constraints as $y(I) \geq 0$, for all $I \in \mathcal{I}$.

The main issue with the above LP is that it has exponential number of variables. The LP has one variable for each job instance and so, the total number of variables is $|\mathcal{I}|$, which can be as large as T^r. In our setup, r is assumed to be an arbitrary input and so, the number of variables could be exponential. Hence, a polynomial time algorithm cannot even afford to explicitly write down the above LP and directly solve it. However, notice that number of constraints in the above LP is $T+n$, which is polynomial in the input length. This means that an optimal basic feasible solution (BFS) will set at most $T+n$ variables to non-zero values. Our goal is to find these non-zero variables and their values in time polynomial in T, n and r. We achieve this above goal by constructing a separation oracle for the dual LP, as discussed next.

Solving the LP. Consider the dual LP. We introduce dual variables $\alpha(t)$ corresponding to the set of constraints (1) and $\beta(J)$ corresponding to the set of constraints (2). The dual includes a constraint corresponding to each primal variable $y(I)$. For an instance $I \in \mathcal{I}$, let J_I denote the job to which it belongs. Then, the dual LP is as follows:

$$\min \qquad \sum_{t \in [1,T]} \alpha(t) + \sum_{J \in \mathcal{J}} \beta(J)$$

$$\beta(J_I) + \sum_{t \,:\, I \sim t} \alpha(t) \cdot h_t(I) \geq p(I) \qquad \text{for all job instances } I \in \mathcal{I}$$

The dual also includes non-negativity constraints: $\alpha(t) \geq 0$ and $\beta(J) \geq 0$. The dual has $T+n$ variables and $|\mathcal{I}|$ constraints (excluding the trivial non-negativity constraints); the number of variables is polynomial, whereas the number of constraints is exponential.

We shall construct a separation oracle for the dual. Recall that such a procedure takes as input a vector specifying values for all the dual variables and outputs whether or not the vector is a feasible solution; moreover, if the vector is not feasible, then the procedure must also output a constraint which is violated. Given such an oracle, the ellipsoid algorithm can solve the dual LP and find the optimum solution in polynomial time, even though the number of constraints is exponential. Our separation oracle procedure works using a dynamic programming approach.

The separation oracle is described next. Let $\alpha(\cdot)$ and $\beta(\cdot)$ be the input vectors specifying values assigned to the variables. We say that a job instance I is violated, if the dual constraint corresponding to I is violated by the input vectors. The goal is to find if there exists a job instance I which is violated. Towards that goal, we consider the jobs in $\mathcal{J}$ iteratively and for each job J, we determine if one of the job instances of J is violated.

Fix a job $J \in \mathcal{J}$. For a job instance $I \in \text{Inst}(J)$, let $\lambda(I)$ denote the sum $\sum_{t\,:\,I\sim t} \alpha(t)h_t(I)$. Let I^* be the job instance having the minimum value of $\lambda(I)$, among all the job instances in $\text{Inst}(J)$. All the instances $I \in \text{Inst}(J)$ have identical value $\beta(J_I)$ and $p(I)$. So, if there exists a instance $I \in \text{Inst}(J)$ which is violated, then the job instance I^* will also violated. Thus, it suffices if we find the job instance I^* and the value $\lambda(I^*)$.

We shall find I^* and $\lambda(I^*)$ using dynamic programming. Let $a_1, a_2, \ldots, a_r$ be the sequence of tasks contained in the job J. Fix an integer $1 \leq k \leq r$. By a *k-partial job instance* of J, we mean a sequence of segments $u_1, u_2, \ldots, u_k$ such that u_i is a segment of a_i and u_i finishes before u_{i+1} starts. The notion of $\lambda(\cdot)$ can be naturally extended to k-partial job instances P. Namely, P is said to be active at a timeslot t, if one of the segments of P is active at t and in this case, $h_t(P)$ is defined to be the height of the segment of P active at t; then, $\lambda(P) = \sum_{t\,:\,P\sim t} \alpha(t)h_t(P)$. For a timeslot $t \in [1, T]$ and an integer $1 \leq k \leq r$, let $M[t, k]$ denote the minimum value $\lambda(P)$ achieved by any k-partial job instance of J satisfying the property that all the segments of P are contained within $[1, t]$. Notice that due to the release time and deadline constraints of the tasks, no such k-partial job instance may exist; in this case, we define $M[t, k]$ be ∞. The value $\lambda(I^*)$ that we wish to compute is given by the entry $M[T, r]$.

The table $M[\cdot, \cdot]$ can be computed using the recurrence relation described below. We consider all possible segments of the task a_k which are contained within $[1, t]$ and for each such possibility, we consider the best way of selecting segments for $a_1, a_2, \ldots, a_{k-1}$. Then, among these possibilities we choose the one yielding the minimum $\lambda(\cdot)$ value. The recurrence relation is as follows:

$$M[t,k] = \min_{\text{rt}(a_k)\leq \widetilde{t} \leq t-\rho(a_k)+1} \left(M[\widetilde{t}-1, k-1] + \sum_{i=\widetilde{t}}^{\widetilde{t}+\rho(a_k)-1} \alpha(i) \cdot h(a_k). \right).$$

For the base case, we define $M[t, 0] = 0$, for all timeslots $t \in [1, T]$. Using the above recurrence relation, we can compute all the entries of M. In particular, we can find I^* and $\lambda(I^*)$.

The separation oracle runs in time polynomial in n, T and r. Given the oracle, the ellipsoid algorithm can compute the optimum solution to the dual LP. Furthermore, it can also output the optimum solution to the primal LP. As mentioned earlier, only $n + T$ primal variables will have non-zero value in the primal (basic feasible) optimum solution. Using the ellipsoid algorithm, in conjunction with the separation oracle, we can find these non-zero variables and their values in time polynomial in n, T and r. We refer to the book by Grötschel et al. [10] for more details.

3 Rounding the LP Solution

The algorithm discussed in the previous section yields the optimum fractional solution to the primal LP, denoted by $\mathbf{y}$. In this section, we describe a procedure for rounding the solution. Let $\widetilde{\mathcal{I}}$ be the set of job instances that receive non-zero

value under $\mathbf{y}$. Recall that only $n+T$ primal variables will have non-zero value in the primal (basic feasible) optimum solution. Thus, the number of instances in $\widetilde{\mathcal{I}}$ is at most $n+T$.

For a job J, let $\mathbf{x}(J)$ denote the sum of $\mathbf{y}(I)$ over all job instances of J that receive non-zero value under $\mathbf{y}$. Intuitively, this is the value assigned by the LP solution to the job J. Let $\widetilde{\mathcal{J}}$ denote the set of jobs having non-zero value for $\mathbf{x}(J)$. The profit of the LP solution is then given by $p(\mathbf{y}) = \sum_{J \in \widetilde{\mathcal{J}}} \mathbf{x}(J)p(J)$. Clearly, the optimum integral solution satisfies $p(\texttt{Opt}) \leq p(\mathbf{y})$. We shall present a randomized rounding procedure which outputs a (integral) feasible solution $\mathcal{S}$ such that the expected profit satisfies $\mathbf{E}[p(\mathcal{S})] \geq p(\mathbf{y})/(8r)$.

The basic idea behind the rounding procedure is as follows. A natural rounding strategy is to select each job with probability $\mathbf{x}(J)$. But, it is difficult to argue that such a procedure will output a feasible solution with high profit. However, we shall show that if we "scale down" the selection probability by a factor $1/(cr)$, then we can get a solution with high profit (where c is a suitable constant). We note that the above idea of scaling down the probabilities has been successfully used in other contexts in prior work (see for example, [7], [8]). The rounding procedure is explained in detail next.

The rounding procedure proceeds in four phases:

- *Job Selection Phase:* Consider each job $J \in \widetilde{\mathcal{J}}$ and select it with probability $\mathbf{x}(J)/(4r)$. The jobs are selected independently at random. Let $\mathcal{J}_{\text{sel}}$ denote the set of selected jobs.
- *Job Instance Selection Phase:* Consider each selected job $J \in \mathcal{J}_{\text{sel}}$. Select exactly one job instance from $\text{Inst}(J)$, where an instance $I \in \text{Inst}(J)$ is chosen with probability $\mathbf{y}(I)/\mathbf{x}(J)$. Let $\mathcal{I}_{\text{sel}}$ be the set of job instances selected.
- *Segment Selection Phase:* Consider the set of all the segments belonging to the selected job instances. Arrange all these segments in the increasing order of their starting timeslots. Let $U = \emptyset$. For each segment u in the above ordering , select u if u can be added to U without violating the bandwidth constraint (i.e., $h(u) + h_t(U) \leq 1$, for all timeslots t in the span of u). Let $\mathcal{U}_{\text{sel}}$ denote the set of selected segments.
- *Output Phase:* Construct a set $\mathcal{I}_{\text{out}}$ as follows. For each job instance $I \in \mathcal{I}_{\text{sel}}$, include I in $\mathcal{I}_{\text{out}}$, if all the r segments of I are found in $\mathcal{U}_{\text{sel}}$. Consider a job $J \in \mathcal{J}_{\text{sel}}$ and let I be the unique job instance selected for J. Output the job J, if I is included in $\mathcal{I}_{\text{out}}$. Let $\mathcal{S}$ be the set of all jobs output. The set $\mathcal{S}$ is the solution output by the procedure.

Regarding the second phase, for any job $J \in \mathcal{J}_{\text{sel}}$, $\sum_{I \in \text{Inst}(J)} \mathbf{y}(I)/\mathbf{x}(J) = 1$. So, we will select exactly one job instance from $\text{Inst}(J)$. In the fourth phase, for any job $J \in \mathcal{S}$, the corresponding job instance I included in $\mathcal{I}_{\text{out}}$ specifies where the tasks of J must be scheduled. Thus, $\mathcal{S}$ constitutes the full description of a feasible solution. We next analyze the rounding procedure.

Lemma 1. *Suppose all the input tasks are small. Then,* $\mathbf{E}[p(\mathcal{S})] \geq p(\mathbf{y})/(8r)$.

Proof: Consider any job instance $I \in \widetilde{\mathcal{I}}$. The probability that I is output is :

$$\Pr[I \in \mathcal{I}_{\text{out}}] = \Pr[I \in \mathcal{I}_{\text{sel}}] \cdot \Pr[I \in \mathcal{I}_{\text{out}} \mid I \in \mathcal{I}_{\text{sel}}]. \tag{3}$$

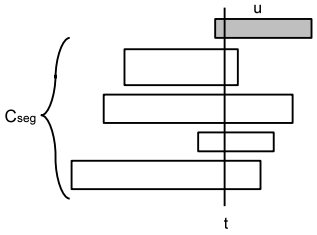


Fig. 2. Illustration for Proof of Lemma 1

Consider the first term in the RHS. Let J be the job to which I belongs. Then,

$$\Pr[I \in \mathcal{I}_{\text{sel}}] = \Pr[J \in \mathcal{J}_{\text{sel}}] \cdot \Pr[I \in \mathcal{I}_{\text{sel}} \mid J \in \mathcal{J}_{\text{sel}}] = \frac{\mathbf{x}(J)}{4r} \times \frac{\mathbf{y}(I)}{\mathbf{x}(J)} = \frac{\mathbf{y}(I)}{4r}. \quad (4)$$

Now consider the second term in the RHS of (3). Let the segments contained in I be $u_1, u_2, \ldots, u_r$. Then,

$$\begin{aligned}\Pr[I \in \mathcal{I}_{\text{out}} \mid I \in \mathcal{I}_{\text{sel}}] &= \Pr[\forall u \in I, u \in \mathcal{U}_{\text{sel}} \mid I \in \mathcal{I}_{\text{sel}}] \\ &= 1 - \Pr[\exists u \in I, u \notin \mathcal{U}_{\text{sel}} \mid I \in \mathcal{I}_{\text{sel}}] \geq 1 - \sum_{u \in I} \Pr[u \notin \mathcal{U}_{\text{sel}} \mid I \in \mathcal{I}_{\text{sel}}], \quad (5)\end{aligned}$$

where the last statement follows from the union bound. Let us derive a bound on each term of the summation in the last line.

We refer to Figure 2(a). Consider any segment $u \in I$. Let t be the starting timeslot of u. Let U be the set of segments that have already been selected when u was considered in the segment selection phase. Suppose u is not selected to be included in $\mathcal{U}_{\text{sel}}$. This implies that inclusion of u violates the bandwidth constraint at some timeslot t' in the span of u, meaning $h_{t'}(U) + h(u) > 1$. Recall that all segments are assumed to be small. In particular, $h(u) \leq 1/2$ and so $h_{t'}(U) \geq 1/2$. The segments are considered in the increasing order of their starting timeslots. Thus all segments of U active at the timeslot t' must also be active at the timeslot t. It follows that $h_t(U) \geq h_{t'}(U) \geq 1/2$. Hence,

$$\Pr[u \notin \mathcal{U}_{\text{sel}} \mid I \in \mathcal{I}_{\text{sel}}] \quad \leq \quad \Pr[h_t(U) \geq 1/2 \mid I \in \mathcal{I}_{\text{sel}}]. \quad (6)$$

We next derive a bound on the random variable $h_t(U)$.

Let $\mathcal{U}$ be the union of all segments[1]. For a segment $v \in \mathcal{U}$, let I_v be the job instance to which v belongs. Let C_{seg} be the set of all segments from $\mathcal{U}$ which are active at the timeslot t and considered earlier than u in the ordering considered in the segment selection phase (excluding u); we call C_{seg} as the *conflict segment set* of u. The expectation of the random variable $h_t(U)$ can be expressed as:

$$\mathbf{E}[h_t(U)] = \sum_{v \in C_{\text{seg}}} \Pr[v \in \mathcal{U}_{\text{sel}}] h(v) \leq \sum_{v \in C_{\text{seg}}} \Pr[I_v \in \mathcal{I}_{\text{sel}}] h(v),$$

[1] Notice that $\mathcal{U}$ is multi-set, since a segment v belonging to a task a of a job J' may be added multiple times in $\mathcal{U}$ by the different job instances of J'.

where the second statement follows from the fact that a segment v can belong to $\mathcal{U}_{\text{sel}}$, only if I_v belongs to $\mathcal{I}_{\text{sel}}$. A similar argument shows that

$$\mathbf{E}[h_t(U) \mid I \in \mathcal{I}_{\text{sel}}] \leq \sum_{v \in C_{\text{seg}}} \Pr[I_v \in \mathcal{I}_{\text{sel}} \mid I \in \mathcal{I}_{\text{sel}}] h(v).$$

For any job instance I' belong to the same job as I, $\Pr[I' \in \mathcal{I}_{\text{sel}} \mid I \in \mathcal{I}_{\text{sel}}] = 0$. On the other hand for any job instance I' belonging to a different job than that of I, the two events "$I' \in \mathcal{I}_{\text{sel}}$" and "$I \in \mathcal{I}_{\text{sel}}$" are independent (since jobs are includes in $\mathcal{J}_{\text{sel}}$ independently at random). It follows that

$$\mathbf{E}[h_t(U) \mid I \in \mathcal{I}_{\text{sel}}] \leq \sum_{v \in C_{\text{seg}}} \Pr[I_v \in \mathcal{I}_{\text{sel}}] h(v).$$

We can now appeal to Equation (4):

$$\begin{aligned} \mathbf{E}[h_t(U) \mid I \in \mathcal{I}_{\text{sel}}] = \sum_{v \in C_{\text{seg}}} \frac{\mathbf{y}(I_v)}{4r} h(v) \quad &= \quad \sum_{v \in C_{\text{seg}}} \frac{\mathbf{y}(I_v)}{4r} h_t(I_v) \\ \leq \sum_{I' \,:\, I' \sim t} \frac{\mathbf{y}(I')}{4r} h_t(I') \quad &\leq \quad 1/(4r). \end{aligned}$$

The first statement follows from Equation (4); the third statement follows from the fact that all the segments in C_{seg} are active at timeslot t; the last statement follows from the bandwidth constraint of the primal LP. By Markov's inequality, $\Pr[h_t(U) \geq 1/2 \mid I \in \mathcal{I}_{\text{sel}}] \leq (1/2r)$. Substituting in (6), we get that $\Pr[u \notin \mathcal{U}_{\text{sel}} \mid I \in \mathcal{I}_{\text{sel}}] \leq 1/(2r)$. Substituting in (5), we have that $\Pr[I \in \mathcal{I}_{\text{out}} \mid I \in \mathcal{I}_{\text{sel}}] \geq 1/2$. (since each job instance has r segments). It now follows from (3) and (4) that $\Pr[I \in \mathcal{I}_{\text{out}}] \geq \mathbf{y}(I)/(8r)$.

Consider any job J. The job J will be included in $\mathcal{S}$, if the job instance chosen for J is included in $\mathcal{I}_{\text{out}}$. We see that

$$\Pr[J \in \mathcal{S}] \quad = \quad \sum_{I \in \text{Inst}(J)} \Pr[I \in \mathcal{I}_{\text{out}}] \quad \geq \quad \left(\frac{1}{8r}\right) \sum_{I \in \text{Inst}(J)} \mathbf{y}(I) \quad = \quad \frac{\mathbf{x}(J)}{8r}.$$

We can now compute $\mathbf{E}[p(S)]$, by appealing to linearity of expectation.

$$\mathbf{E}[p(\mathcal{S})] \quad = \quad \sum_{J \in \mathcal{J}} \Pr[J \in \mathcal{S}] p(J) \quad \geq \quad \left(\frac{1}{8r}\right) \sum_{J \in \mathcal{J}} \mathbf{x}(J) p(J) \quad = \quad \frac{p(\mathbf{y})}{8r}.$$

This completes the proof of the lemma. □

We have established the main result of the paper (Theorem 1).

4 Conclusions and Open Problems

We presented a randomized $O(r)$-approximation algorithm for the SplitJob problem with windows, when all the tasks have height at most 1/2. We showed

that when the tasks can have arbitrary heights, the natural LP has an integrality gap of $\Omega(2^r)$. Overcoming this issue and designing an $O(r)$-approximation algorithm is an interesting open problem.

Recall that in the introduction, we identified an interesting special case of the SplitJob problem with windows, wherein the windows are associated with jobs, rather than tasks. Clearly, our results imply $O(r)$-approximation algorithms for this problem. Designing an algorithm with better approximation ratio is an interesting avenue of research. We note that a constant factor approximation algorithm is not ruled out for this problem.

Recall that it is NP-hard to approximate the basic SplitJob problem (without windows) within a factor of $O(r/\log r)$, for the unit height case [3]. This hardness result also holds for the uniform height case. For the case of small tasks, we showed that it is NP-hard to approximate within $r^{1/3}$. Improving the hardness result to $O(r/\log r)$ would be of interest.

References

1. Bafna, V., Narayanan, B., Ravi, R.: Nonoverlapping local alignments (weighted independent sets of axis-parallel rectangles). Discrete Applied Math. 71(1-3), 41–53 (1996)
2. Bar-Noy, A., Bar-Yehuda, R., Freund, A., Naor, J., Schieber, B.: A unified approach to approximating resource allocation and scheduling. J. of the ACM 48(5), 1069–1090 (2001)
3. Bar-Yehuda, R., Halldórsson, M., Naor, J., Shachnai, H., Shapira, I.: Scheduling split intervals. SIAM Journal of Computing 36(1), 1–15 (2006)
4. Bar-Yehuda, R., Rawitz, D.: Using fractional primal-dual to schedule split intervals with demands. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 714–725. Springer, Heidelberg (2005)
5. Berman, P., DasGupta, B., Muthukrishnan, S.: Simple approximation algorithm for nonoverlapping local alignments. In: SODA (2002)
6. Bonsma, P., Schulz, J., Wiese, A.: A constant factor approximation algorithm for unsplittable flow on paths. In: FOCS (2011)
7. 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)
8. Chakrabarti, A., Chekuri, C., Gupta, A., Kumar, A.: Approximation algorithms for the unsplittable flow problem. Algorithmica 47(1), 53–78 (2007)
9. Chekuri, C., Khanna, S.: On multidimensional packing problems. SIAM J. Comput. 33(4), 837–851 (2004)
10. Grötschel, M., Lovász, L., Schrijver, A.: Geometric Algorithms And Combinatorial Optimization. Springer (1993)