

Varying Bandwidth Resource Allocation Problem with Bag Constraints

Venkatesan T. Chakaravarthy, Vinayaka Pandit, Yogish Sabharwal, and Deva P. Seetharam

IBM Research - India,

New Delhi, INDIA

Email: {vechakra, pvinayak, ysabharwal, dseetharam}@in.ibm.com

Abstract—We consider the problem of scheduling jobs on a pool of machines. Each job requires multiple machines on which it executes in parallel. For each job, the input specifies release time, deadline, processing time, profit and the number of machines required. The total number of machines may be different at different points of time. A feasible solution is a subset of jobs and a schedule for them such that at any timeslot, the total number of machines required by the jobs active at the timeslot does not exceed the number of machines available at that timeslot. We present an $O(\log(B_{\max}/B_{\min}))$ -approximation algorithm, where B_{\max} and B_{\min} are the maximum and minimum available bandwidth (maximum and minimum number of machines available over all the timeslots).

Our algorithm and the approximation ratio are applicable for more a general problem that we call the Varying bandwidth resource allocation problem with bag constraints (BAGVBRAP). The BAGVBRAP problem is a generalization of some previously studied scheduling and resource allocation problems.

Keywords-Scheduling; resource allocation; approximation algorithm;

I. INTRODUCTION

We consider scheduling problems in parallel and distributed settings in which we need to schedule jobs on a system offering a certain amount of some resource. Each job requires a particular amount of the resource for its execution. The total amount of the resource offered by the system is different at different points of time. Our goal is to choose a subset of jobs and schedule them such that at any timeslot, the total amount of resource requirement does not exceed the total amount of the resource available at that timeslot. We wish to maximize the profit of the chosen subset of jobs. The problem formulation is motivated by its applications in environments such as cloud computing and bandwidth allocation in networks. Below, we describe a real-life problem encountered in scheduling scientific applications on a massively parallel system.

We now describe a scheduling problem typically faced in the scenario where a number of users are trying to execute scientific applications on either a cluster of machines or a supercomputer. The users have to make reservations for the resources in order to execute their jobs. But, as there are multiple users competing for the same resources, a user may not be allocated all the resources she requested. For the sake of simplicity, let us assume that the resources are processors on the supercomputer or machines on the cluster. Consider

a particular user. The number of processors (or machines) allocated to the user may be different at different points of time (because of reservation policies and the presence of critical jobs) The user gets to know in advance the number of processors allocated to her for each timeslot. The user has a set of jobs that she wishes to execute. Each job of the user has a requirement on the number of processors needed for execution. In addition, each job has a release time, a processing time, a deadline and a profit. The user would like to select a subset of jobs and schedule them in such a way that at any timeslot, the total number of processors required by the jobs active at the timeslot does not exceed the total number processor available to the user at that timeslot. Naturally, the user would wish to choose the subset of jobs having the maximum profit. We would like to highlight that such a scenario is frequently encountered in practice. We assume that a job can be executed on any subset of machines or processors as long as the resource requirement is met (i.e., the machines/processors are identical) and the jobs may not be preempted. In fact, we consider a more general scenario where job can even specify a set of time intervals where it can be scheduled; note that this generalizes the notion of release time and deadline.

Motivated by scheduling and bandwidth allocation scenarios such as the above one, we study an abstract problem that we call the *Varying bandwidth resource allocation problem with bag constraints* (BAGVBRAP). We use *bandwidth* as a generic term to refer to the quantity of the resource under contention. So, the input will specify the bandwidth available at each timeslot, and for each job, its bandwidth requirement and the different time intervals in which it can be scheduled. This kind of interval selection or interval scheduling problems arise naturally in practice. We refer to [1], [2], [3] for real-life applications of interval selection and scheduling in parallel and distributed computing and network management. The BAGVBRAP problem also has applications in smart energy management. Here, we have a set of electrical appliances that need to be scheduled over a period of time, during which the amount of available power may vary, due to the use of different power sources. The BAGVBRAP problem generalizes several previously studied scheduling and resource allocation problems. We next define the problem and then discuss prior work.

A. BAGVBRAP: Problem Definition

The input consists of a set of jobs \mathcal{J} . Each job $J \in \mathcal{J}$ consists of a set of *job instances* of which at most one can be selected for execution. An instance u of a job J is specified by an interval $I_u = [a, b]$, where a and b are the *start time* and the *finish time* of the instance u ; we assume that a and b are integers. The instance u is also associated with a *bandwidth requirement* ρ_u and a profit p_u . Let D be the maximum finish time over all instances so that the interval associated with every job instance is contained in the span $[1, D]$. We refer to each integer $1 \leq t \leq D$ as a *timeslot*. For each timeslot t , the input specifies a number B_t which is the *bandwidth available* at timeslot t .

We use the term *instance* as a shorthand for job instance. Let \mathcal{U} denote the set of all instances over all the jobs in \mathcal{J} . For each instance $u \in \mathcal{U}$, we view each interval $I_u = [a, b]$ as a set of timeslots in the range $[a, b]$. We view each job as a *bag* of its instances. We say that the instance u is *active* at a timeslot t , if $t \in I_u$. For a timeslot t , let $A(t)$ denote the set of all instances active at timeslot t .

A feasible solution is a subset of instances $S \subseteq \mathcal{U}$ such that at every timeslot t , the sum of the bandwidth requirements of the instances from S active at time t is at most B_t , i.e., for every timeslot $1 \leq t \leq D$,

$$\sum_{u \in S \cap A(t)} \rho_u \leq B_t.$$

We call this the *bandwidth constraint*. Furthermore, it is required that at most one instance is picked from each job; we call this the *bag constraint*; we view a job as a bag of instances and hence the terminology. The problem is to find a feasible solution S such that the sum of the profits of the jobs in S is maximized. \square

In our example scheduling problem described earlier, the bandwidth corresponds to the number of machines/processors. The notion of the release time and the deadline is captured by the notion of bags as follows. For a job J in the scheduling problem with release time r and deadline d , and processing time s we create a bag containing $d - r - s + 1$ instances corresponding to the integer intervals of length s lying between r and d . Note that the bag constraint ensures that any chosen job is scheduled in exactly one of the possible time intervals.

The concept of bag constraints is quite powerful. Apart from handling the notion of release time and deadline, it can also work in a more general setting where a job can specify a set of possible time intervals where it can be scheduled. Moreover, BAGVBRAP allows for different instances of the same job to have different bandwidth requirements, processing times and profits.

B. Prior Work

The BAGVBRAP problem is a generalized formulation that captures as special cases many well-studied scheduling

and resource allocation problems. Here we shall describe some important special cases and then present a brief survey of some of the prior work dealing with these problems.

- *Unit bandwidth resource allocation problem (UNITRAP)*: This is the special case of BAGVBRAP problem in which the bandwidth available is 1 across all timeslots and each job has only once instance having a bandwidth requirement of 1.
- *Unit bandwidth resource allocation problem with bag constraints (BAGUNITRAP)*: This is the special case of BAGVBRAP problem, in which bandwidth available is 1 across all timeslots and the bandwidth requirements of all instances is also 1.
- *Uniform bandwidth resource allocation problem (UBRAP)*: This is the special case of the BAGVBRAP problem, where the bandwidth available is uniform across all timeslots, (i.e., for all $1 \leq t \leq D$, $B_t = B$ for some fixed B given as part of the input) and each job has only one instance.
- *Uniform bandwidth resource allocation problem with bag constraints (BAGUBRAP)*: This is the special case of the BAGVBRAP problem, where the bandwidth available is uniform across all timeslots, i.e., for all $1 \leq t \leq D$, $B_t = B$ for some fixed B given as part of the input.
- *Varying bandwidth resource allocation problem (VBRAP)*: This is the special case of the BAGVBRAP problem, where each job has only one instance.

The simplest of the special cases is the UNITRAP problem. Notice that in this problem, the input is simply a set of intervals having profits and goal is to choose a subset of non-overlapping intervals of maximum profit. In other words, this is the same as the classical maximum weight independent set problem on interval graphs. This is well-known to be solvable in polynomial time via dynamic programming [4].

Bar-Noy et al. [5] and Berman and Dasgupta [6] generalized the UNITRAP problem by introducing the notion of bag constraints and defined the BAGUNITRAP problem. They both (independently) gave a 2-approximation algorithm for the BAGUNITRAP problem. Spieksma [7] showed that the BAGUNITRAP problem is APX-hard.

Calinescu et al. [8] considered the UBRAP problem and presented a 3-approximation algorithm, based on LP-rounding technique.

Phillips et al. [9] gave a 6-approximation for the BAGUBRAP problem. Bar-Noy et al. [10] obtained a 4-approximation for the same problem, via the local ratio technique. They also showed how BAGUBRAP generalizes many previously studied problems in scheduling and other scenarios.

The VBRAP problem is equivalent to the unsplitable flow problem (UFP) on lines and has been studied within that context. Chakrabarti et al. [11] presented a $O(\log(\rho_{\max}/\rho_{\min}))$ -approximation algorithm, where ρ_{\max}

and ρ_{\min} are the maximum and minimum bandwidth requirements. They also considered a special case of VBRAP satisfying the *no-bottle assumption*, wherein it is assumed that the maximum bandwidth requirement is less than the minimum bandwidth available; they presented a constant factor randomized approximation algorithm for this special case. For the same special case, Chekuri et al. [12] improved the constant factor. For the general UFP on line (i.e., the VBRAP problem), Bansal et al. [13] presented an $O(\log n)$ -approximation algorithm. In a different paper, Bansal et al. [14] obtained a quasi polynomial time PTAS for the VBRAP problem.

A framework related to the resource allocation problem is the dynamic storage allocation problem, wherein the bandwidth must be allocated to an instance in a contiguous manner. Such a framework applies to resources such as storage and memory. Bar-Noy et al. [10] and Leonardi et al. [15] study these related problems and develop constant factor approximation algorithms.

The BAGVBRAP problem generalizes UBRAP in two dimensions by introducing the notions of varying bandwidth and bag constraints. The prior work have either considered the notion of the bag constraints or the notion of the varying bandwidth. In this paper, we study the most general BAGVBRAP problem and develop an approximation algorithm. To the best of our knowledge, the general BAGVBRAP problem has not been addressed in prior work.

C. Our Results

Our main result is an $O(\log(B_{\max}/B_{\min}))$ -approximation algorithm for the BAGVBRAP problem, where B_{\max} and B_{\min} are the maximum and minimum bandwidths available over all timeslots.

In the second part of the paper, we present a constant factor approximation for a special case of the BAGVBRAP problem that we denote as the LU-BAGVBRAP problem. In this special case, the input satisfies the following property: for each job instance u , the bandwidth available does not change during its interval I_u (i.e., for all instances u , for all timeslots $t_1, t_2 \in I_u$, we have $B_{t_1} = B_{t_2}$). We refer the property as the *local uniformity* property. Our second result is a 5-approximation algorithm for the LU-BAGVBRAP problem. The motivation for studying the LU-BAGVBRAP problem comes from the fact that it captures a natural multi-system generalization of the BAGUBRAP problem. Recall that the BAGUBRAP is a scheduling problem wherein we have a single system offering a uniform bandwidth. In the generalization, we have multiple systems each offering certain bandwidth (which can be different for different systems). A job instance can be scheduled on any one of the systems. We refer to this generalization as MULTIBAGUBRAP. We present a reduction from MULTIBAGUBRAP to LU-BAGVBRAP, there by getting a 5-approximation for MULTIBAGUBRAP.

D. Proof Techniques

Our algorithm and analysis for the BAGVBRAP problem builds on the work of Calinescu et al. [8]. Calinescu et al. [8] consider the UBRAP problem and present a 3-approximation algorithm. They divide the job instances into “large” and “small” instances based on their bandwidth requirement. They find two feasible solutions, one consisting of only large instances and the other consisting of only the small instances. Then, the best of the two solutions is output. They observe that finding an optimal solution restricted to the large instances alone reduces to the problem of finding maximum weight independent set on interval graphs, which is solvable in polynomial time. In the case of small instances, they consider a natural LP relaxation and design a rounding scheme. They show that this LP-based algorithm is a 2-approximation, when only the small instances are considered. Then they argue that the final solution output is a 3-approximation to the original input instance.

We extend their algorithm and analysis to handle the notion of bag constraints and the notion of varying bandwidth. We also divide the instances into “large” and “small” instances. We further classify the “large” instances into logarithmic number of buckets. When only instances from any particular bucket are considered, we argue that finding the optimal solution reduces to the BAGUNITRAP problem (with a factor two loss in approximation). We then invoke the 2-approximation algorithm for BAGUNITRAP [5], [6] on each bucket and obtain a 4-approximate feasible solution with respect to each bucket. The best of these solution is picked and is guaranteed to be a log-factor approximation to the optimum solution restricted to the large instances. We then consider the case of small instances. We extend the LP-rounding scheme and its analysis to get a log-factor approximate feasible solution to the optimum solution restricted to the small instances. Finally, we pick the better of the two solutions. The main crux of our work lies in carefully incorporating the implications of the bag constraints and varying bandwidth within the above framework of Calinescu et al. [8]; this requires introducing additional arguments.

II. AN APPROXIMATION ALGORITHM FOR THE BAGVBRAP PROBLEM

In this section, we present an approximation algorithm for the BAGVBRAP problem. We start by developing some notations.

Let $X \subseteq \mathcal{U}$ be a set of instances. We say that the instances in X *overlap*, if there exists a timeslot in which all the instances are active (i.e., $\cap_{u \in X} I_u \neq \emptyset$). A subset $X \subseteq \mathcal{U}$ is said to be an *independent set*, if no two instances in X are overlapping. For a subset of instances $X \subseteq \mathcal{U}$, let $\text{Profit}(X)$ denote the sum of profits of the instances contained in X , i.e., $\text{Profit}(X) = \sum_{u \in X} p_u$. Let B_{\max} and B_{\min} be the maximum and minimum bandwidth available during the

span of the schedule, i.e., $B_{\max} = \max\{B_t \mid t \in [1, D]\}$ and $B_{\min} = \min\{B_t \mid t \in [1, D]\}$.

Our goal is to develop an $O(\log(B_{\max}/B_{\min}))$ approximation algorithm for BAGVBRAP. We divide the instances into two categories – large instances and small instances based on a fixed constant $\alpha = 3/4$. For an instance u , let B_{\min_u} denote the minimum bandwidth available while it is active, i.e., $B_{\min_u} = \min\{B_t \mid t \in I_u\}$. An instance u is said to be *large* if $\rho_u > \alpha \cdot B_{\min_u}$; it is said to be *small* otherwise. Partition the set of all instances \mathcal{U} into \mathcal{U}_l and \mathcal{U}_s , where \mathcal{U}_l is the set of all large instances and \mathcal{U}_s is the set of all small instances.

Let A^* denote the optimal solution and let $P^* = \text{Profit}(A^*)$. Let A_l^* be the optimal solution when only the large job instances are considered. That is,

$$A_l^* = \operatorname{argmax}_X \{ \text{Profit}(X) \mid X \text{ is a feasible solution and } X \subseteq \mathcal{U}_l \}$$

Similarly, let A_s^* be the optimal solution when only the small job instances are considered. Let $P_l^* = \text{Profit}(A_l^*)$ and $P_s^* = \text{Profit}(A_s^*)$. Notice that $P^* \leq P_l^* + P_s^*$.

In Section II-A, we develop an algorithm that outputs a feasible solution A_l consisting of only large instances having profit $P_l = \text{Profit}(A_l)$ such that for some constant c_1 ,

$$P_l^* \leq c_1 \log(B_{\max}/B_{\min}) P_l \quad (1)$$

Similarly, in Section II-B, we develop an algorithm that outputs a feasible solution A_s consisting of only small instances having profit $P_s = \text{Profit}(A_s)$ such that for some constant c_2 ,

$$P_s^* \leq c_2 \log(B_{\max}/B_{\min}) P_s \quad (2)$$

Of the two solutions A_l and A_s , we output the one with the higher profit as the final solution; we denote the output solution as A . Let $P = \text{Profit}(A)$. It would follow that for some constant c , $P^* \leq c \log(B_{\max}/B_{\min}) P$, establishing the following main theorem.

Theorem 2.1: Our algorithm for the BAGVBRAP problem has $O(\log(B_{\max}/B_{\min}))$ approximation ratio.

A. Handling Large Instances

In this section, we consider only the large instances and find a feasible solution A_l such that

$$P_l^* \leq 4 \left[1 + \frac{\log(B_{\max}/B_{\min})}{1 + \log \alpha} \right] \cdot P_l. \quad (3)$$

This would establish the claim in Equation 1.

In order to achieve this, we partition the set of large instances \mathcal{U}_l into m sets $\mathcal{U}_{l,0}, \mathcal{U}_{l,1}, \dots, \mathcal{U}_{l,m-1}$ based on their bandwidth requirements, where $m = \lfloor 1 + \frac{\log(B_{\max}/B_{\min})}{1 + \log \alpha} \rfloor$. For $0 \leq i \leq m-1$, define

$$\mathcal{U}_{l,i} = \{u \in \mathcal{U}_l \mid \rho_u \in [(2\alpha)^i \cdot \alpha \cdot B_{\min}, (2\alpha)^{i+1} \cdot \alpha \cdot B_{\min}]\}$$

Note that the smallest job is of size at least $\alpha \cdot B_{\min}$. In order to ensure that this is a valid partition of \mathcal{U}_l , α should be at least $1/2$. Our choice of $\alpha = 3/4$ suffices.

Similar to the way A_l^* is defined, for $0 \leq i \leq m-1$, we define $A_{l,i}^*$ to be the optimal feasible solution when only the job instances in $\mathcal{U}_{l,i}$ are considered; let $P_{l,i}^* = \text{Profit}(A_{l,i}^*)$. Note that $P_l^* \leq \sum_{i=0}^{m-1} P_{l,i}^*$.

In the algorithm, for each $0 \leq i \leq m-1$, we will consider the set $\mathcal{U}_{l,i}$ and construct a feasible solution $A_{l,i}$ consisting of only job instances from $\mathcal{U}_{l,i}$ such that $P_{l,i}^* \leq 4P_{l,i}$, where $P_{l,i} = \text{Profit}(A_{l,i})$. Then, the algorithm would pick the best of these m solutions and output it as A_l . It is easy to see that $P_l^* \leq 4mP_l$. Equation 3 would follow.

Our algorithm for computing the feasible solutions $A_{l,i}$ (claimed above) is based on the following crucial claims.

Claim 2.2: Fix any $0 \leq k \leq m-1$. Any feasible solution S cannot contain three or more overlapping instances from $\mathcal{U}_{l,k}$.

Proof: Clearly, it suffices if we prove that S cannot contain three overlapping instances from $\mathcal{U}_{l,k}$. We prove this by contradiction. Suppose there exist three instances $u_1, u_2, u_3 \in S \cap \mathcal{U}_{l,k}$ that are overlapping. It means that there exists a timeslot $t \in I_{u_1} \cap I_{u_2} \cap I_{u_3}$. For $j = 1, 2, 3$, let t_j denote a timeslot in I_{u_j} satisfying $B_{t_j} = B_{\min_{u_j}}$. Without loss of generality, let $t_1 \leq t_2 \leq t_3$. Then clearly at least one of u_1 and u_3 must also be active at t_2 , because otherwise $I_{u_1} \cap I_{u_3} = \emptyset$, contradicting the assumption that $t \in I_{u_1} \cap I_{u_2} \cap I_{u_3}$. Without loss of generality, let u_1 be active at t_2 . Then,

$$\rho_{u_j} \geq (2\alpha)^k \alpha B_{\min} \text{ for } j = 1, 2.$$

On the other hand, $\rho_{u_2} < (2\alpha)^{k+1} \alpha B_{\min}$. Moreover, since we are dealing with large instances, $\rho_{u_2} > \alpha B_{\min_{u_2}}$. It follows that $B_{t_2} = B_{\min_{u_2}} < (2\alpha)^{k+1} \alpha B_{\min}$. Hence, $\rho_{u_1} + \rho_{u_2} \geq 2(2\alpha)^k \alpha B_{\min} > B_{t_2}$. This contradicts the fact that S is a feasible solution. This completes the proof of the claim. \square

Claim 2.3: Fix any $0 \leq k \leq m-1$. For any feasible solution S , the set $S \cap \mathcal{U}_{l,k}$ can be partitioned into X_1 and X_2 such that X_1 and X_2 are independent sets.

Proof: At any timeslot t , at most two jobs from $S \cap \mathcal{U}_{l,k}$ can be active at timeslot t (by Claim 2.2). It is well known that interval graphs are perfect graphs [4]. It follows that the instances in $S \cap \mathcal{U}_{l,k}$ can be colored with two colors such that no two overlapping instances receive the same color. The sets X_1 and X_2 are obtained by partitioning the instances according to their color. \square

For a given $0 \leq i \leq m-1$, let $W_{l,i}^*$ denote the maximum profit independent subset of $\mathcal{U}_{l,i}$ satisfying the bag constraints. The lemma below follows from Claims 2.2 and 2.3.

Lemma 2.4: Fix any $0 \leq k \leq m-1$. Then, $\text{Profit}(W_{l,k}^*) \geq P_{l,k}^*/2$.

Consider any $0 \leq k \leq m-1$. Our algorithm to compute $A_{l,k}$ is as follows. We observe that the problem of computing $W_{l,k}^*$ is the same as the BAGUNITRAP problem over the set $\mathcal{U}_{l,k}$. Recall that Bar-Noy et al. [5] and Berman and Dasgupta [6] presented a 2-approximation algorithm for the BAGUNITRAP problem. We invoke their algorithm with $\mathcal{U}_{l,k}$ as input¹ and obtain a solution; $A_{l,k}$ is taken to be this solution. We have that $P_{l,k} \geq \text{Profit}(W_{l,k}^*)/2$. Now by Lemma 2.4, $P_{l,k} \geq P_{l,k}^*/4$.

B. Handling Small Instances

In this section, we give an algorithm that finds a feasible solution A_s consisting only of instance in \mathcal{U}_s satisfying

$$P_s^* \leq \left[\frac{2 \cdot (1 + \log(B_{\max}/B_{\min}))}{1 - \alpha} + 1 \right] \cdot P_s.$$

Our algorithm follows the general framework introduced by Calinescu et al. [8]. Additionally, we bring in the bag constraints and the varying nature of the bandwidth available. We begin with the following natural IP for computing A_s^* .

$$\text{Maximize} \quad \sum_{u \in \mathcal{U}} p_u x_u \quad (4)$$

$$\text{s.t.} \quad \sum_{u:t \in I_u} \rho_u x_u \leq B_t \quad \text{for all } 1 \leq t \leq D \quad (5)$$

$$\sum_{u \in J} x_u \leq 1 \quad \text{for all } J \in \mathcal{J} \quad (6)$$

$$x_u \in \{0, 1\} \quad \text{for all } u \in \mathcal{U}_s \quad (7)$$

Note that (5) captures the bandwidth constraints and (6) captures the bag constraint. Here $x_u = 1$ if the instance u is selected in the solution and 0 otherwise. The natural relaxation of this IP is obtained by replacing (7) with:

$$0 \leq x_u \leq 1 \quad \text{for all } u \in \mathcal{U}_s \quad (8)$$

For each $u \in \mathcal{U}_s$, let x_u be the fractional value assigned by the LP solution. We assume that instances in \mathcal{U}_s are ordered in the increasing order of their start times.

We shall now discuss a rounding mechanism. We adapt the LIST algorithm and the analysis of Calinescu et al.[8]. However, our analysis requires new arguments and yields an $O(\log B_{\max}/B_{\min})$ -approximation, instead of the constant factor obtained by Calinescu et al.

The algorithm produces a list \mathcal{L} consisting of some m sets of instances S_1, S_2, \dots, S_m together with non-negative real-weights $w(S_1), w(S_2), \dots, w(S_m)$ satisfying the following four properties.

- 1) For each $1 \leq k \leq m$, the set S_k is a feasible solution.
- 2) For each $1 \leq k \leq m$, $0 \leq w(S_k) \leq 1$.
- 3) For each $u \in \mathcal{U}_s$, $\sum_{k:u \in S_k} w(S_k) = x_u$.

¹Ignoring the bandwidths (i.e., taking all bandwidths available and bandwidths required to be 1).

$$4) \sum_{k=1}^m w(S_k) \leq \Delta,$$

where

$$\Delta = 1 + \frac{2 \cdot (1 + \log(B_{\max}/B_{\min}))}{1 - \alpha}.$$

(Recall that α is a parameter set as $\alpha = 3/4$).

Once the list \mathcal{L} is constructed, we output the solution from S_1, S_2, \dots, S_m that has the maximum profit; the output solution is taken to be A_s . Let the profit of A_s be denoted by P_s . The rounding algorithm is shown in Figure 1. Lemma 2.5 shows that the list \mathcal{L} satisfies the four properties. Assuming the lemma, we now argue that $P_s^* \leq \Delta P_s$. (Recall that P_s^* is the profit of the optimal solution when only the small instances are considered). We have

$$\begin{aligned} P_s^* &\leq \sum_{S \in \mathcal{L}} p_u x_u \quad (\text{LP gives an upper bound on } P_s^*) \\ &\leq \sum_{u \in \mathcal{U}_s} p_u \left[\sum_{S \in \mathcal{L} : u \in S} w(S) \right] \quad (\text{By Property 3}) \\ &\leq \sum_{S \in \mathcal{L}} w(S) \left[\sum_{u \in S} p_u \right] \\ &\leq \sum_{S \in \mathcal{L}} w(S) \text{Profit}(S) \\ &\leq \Delta P_s \quad (\text{By Property 4 and the choice of } A_s). \end{aligned}$$

We have shown that the feasible solution A_s satisfies Equation 2.

We now show that the sets in the list \mathcal{L} satisfy the four properties.

Lemma 2.5: The rounding algorithm described above outputs a list \mathcal{L} satisfying the 4 properties.

Proof: It is easy to see that Properties 1 and 2 are satisfied.

In order to show Property 3, for each instance $u \in \mathcal{U}_s$, let \hat{x}_u denote the original value of x_u that was input to the above procedure, i.e. output by the LP. It is easy to verify that the algorithm maintains the following invariant, for any $u \in \mathcal{U}_s$:

$$x_u + \sum_{j:u \in S_j} w(S_j) = \hat{x}_u \quad (9)$$

Property 3 follows from (9) and the fact that after all tasks have been processed $x_u = 0$, for any $u \in \mathcal{U}_s$.

It remains to prove Property 4. We shall show that this property is an invariant of the algorithm. Note that a new set may be added to the list \mathcal{L} either from step 3 or from step 4a. In the latter case, the weight of a set is split amongst itself and the newly created set, leaving the sum of all weights unaffected.

In the former case, the newly created set is a singleton consisting of one instance, say u . Let J be the job to which the instance u belongs. Recall that the instances are processed in the increasing order of their start times. Let Q_u be the set of instances that have so far been processed

1. Consider the instances in order of increasing start times – for instance u :
if $x_u = 0$, proceed to the next instance
(If there are no more instances - goto Step 5)
2. Search \mathcal{L} for a set not containing u to which u can be added without violating Property 1.
3. If no such set exists,
create a new set $V = \{u\}$ with weight $w(V) = x_u$ and add V to \mathcal{L} .
set $x_u = 0$ and return to Step 1.
4. Otherwise suppose $S \in \mathcal{L}$ is such a set
 - 4a. If $x_u < w(S)$, then,
decrease $w(S)$ to $w(S) - x_u$.
create a new set $V = S \cup \{u\}$ with weights $w(V) = x_u$ and add V to \mathcal{L} .
set $x_u = 0$ and return to Step 1.
 - 4b. If $x_u \geq w(S)$, then,
add u to S and decrease x_u to $x_u - w(S)$.
Return to Step 1.
5. Output the set with the maximum profit amongst the sets in \mathcal{L} .

Figure 1. Rounding Algorithm

including u . Consider the set \mathcal{L} immediately after this singleton is added. Partition the list \mathcal{L} into \mathcal{L}_1 and \mathcal{L}_2 , where \mathcal{L}_1 consists of all sets in \mathcal{L} that contain some instance of J (including u itself); \mathcal{L}_2 consists of all the sets in \mathcal{L} that do not contain any instance of J (i.e., $\mathcal{L}_2 = \mathcal{L} - \mathcal{L}_1$).

Claim 2.6: $\sum_{S \in \mathcal{L}_1} w(S) \leq 1$.

Proof: Let J' be the instances in J that have already been processed, including u (i.e., $J' = J \cap Q_u$). For each $u' \in J'$, define $\mathcal{L}_1^{u'}$ to be the sets in \mathcal{L}_1 that contain u' . Notice that this defines a partition of \mathcal{L}_1 . It directly follows from Property 3 that for any $u' \in J'$, $\sum_{S \in \mathcal{L}_1^{u'}} w(S) = \hat{x}_{u'}$. Therefore,

$$\sum_{S \in \mathcal{L}_1} w(S) = \sum_{u' \in J'} \sum_{X \in \mathcal{L}_1^{u'}} w(X) = \sum_{u' \in J'} \hat{x}_{u'} \leq 1,$$

where the last inequality follows from the bag constraint imposed in the LP. This completes the proof of the claim. \square

Now, let us consider the list \mathcal{L}_2 . For a timeslot $t \in I_u$ and a set $S \in \mathcal{L}$, let $\rho_t(S)$ be the sum of the bandwidth requirements of the instances in S that are active at time t , i.e.,

$$\rho_t(S) = \sum_{j \in S \cap A(t)} \rho_j$$

Claim 2.7: For any timeslot $t \in [1, D]$

$$\sum_{S \in \mathcal{L}} \rho_t(S) w(S) \leq B_t$$

Proof: We have

$$\begin{aligned} \sum_{S \in \mathcal{L}} \rho_t(S) w(S) &= \sum_{S \in \mathcal{L}} \left[\sum_{j \in S \cap A(t)} \rho_j w(S) \right] \\ &= \sum_{j \in Q_u \cap A(t)} \rho_j \left[\sum_{S \in \mathcal{L} \mid j \in U} w(S) \right] \\ &= \sum_{j \in Q_u \cap A(t)} \rho_j \hat{x}_j \\ &\leq B_t, \end{aligned}$$

where the last inequality follows from the constraint (5) of the linear program. The claim is proved. \square

Consider any set $S \in \mathcal{L}_2$. We know that u could not be packed in S . The reason could be twofold: either the bag constraint or the bandwidth constraint is violated. However, by the definition of \mathcal{L}_2 , S does not contain any instance from J . This implies that u was not packed in S because of a violation of the bandwidth constraint. It follows that there must exist a timeslot $t \in I_u$ such that $\rho_t(S) + \rho_u > B_t$. Let $\tau(S)$ denote the smallest such timeslot, which we refer to as the *conflict timeslot* of S , i.e., $\tau(S) = \min\{t \in I_u \mid \rho_t(S) + \rho_u > B_t\}$.

We geometrically divide the real interval $[B_{\min}, B_{\max}]$ into r ranges and partition the set \mathcal{L}_2 into r subsets based on the range in which $B_{\tau(S)}$ lies. Let $r = \lceil 1 + \log B_{\max}/B_{\min} \rceil$. For $1 \leq i \leq r$, define

$$\mathcal{L}_2^i = \{S \in \mathcal{L}_2 : 2^{i-1} B_{\min} \leq B_{\tau(S)} < 2^i B_{\min}\}$$

Claim 2.8: For any $1 \leq k \leq r$,

$$\sum_{S \in \mathcal{L}_2^k} w(S) \leq \frac{2}{1 - \alpha}$$

Proof: Define

$$\tau^* = \min_{S \in \mathcal{L}_2^k} \tau(S)$$

Applying Claim 2.7 at τ^* , we get that

$$\sum_{S \in \mathcal{L}} \rho_{\tau^*}(S)w(S) \leq B_{\tau^*}.$$

It follows that

$$\sum_{S \in \mathcal{L}_2^k} \rho_{\tau^*}(S)w(S) \leq B_{\tau^*}.$$

Notice that $B_{\tau^*} \leq 2^k B_{\min}$ and so,

$$\sum_{S \in \mathcal{L}_2^k} \rho_{\tau^*}(S)w(S) \leq 2^k B_{\min} \quad (10)$$

Consider any $S \in \mathcal{L}_2^k$. Since u is a small instance, we have that $\rho_u \leq \alpha B_{\min} \leq \alpha B_{\tau(S)}$. Since $\tau(S)$ is a conflict timeslot of S , we have that $\rho_{\tau(S)}(S) + \rho_u > B_{\tau(S)}$. It follows that

$$\rho_{\tau(S)}(S) \geq (1 - \alpha)B_{\tau(S)} \geq (1 - \alpha)2^{k-1}B_{\min}$$

Recall that $\tau^* \leq \tau(S)$. Since the algorithm considers the instances in increasing start times, all the instances in S that are active at timeslot $\tau(S)$ are also active at τ^* . Therefore, $\rho_{\tau^*}(S) \geq \rho_{\tau(S)}(S)$ and hence,

$$\rho_{\tau^*}(S) \geq (1 - \alpha)2^{k-1}B_{\min} \quad (11)$$

Applying the above bound in (10), we get that

$$\sum_{S \in \mathcal{L}_2^k} [(1 - \alpha)2^{k-1}B_{\min}]w(S) \leq 2^k B_{\min}$$

It follows that

$$\sum_{S \in \mathcal{L}_2^k} w(S) \leq \frac{2}{1 - \alpha}$$

The claim is proved. \square

Summing up over all the ranges shows that

$$\sum_{S \in \mathcal{L}_2} w(S) \leq \frac{2r}{1 - \alpha}$$

Applying the above bound with Claim 2.6, we get that

$$\sum_{S \in \mathcal{L}} w(S) \leq 1 + \frac{2r}{1 - \alpha}$$

Lemma 2.5 is proved. \square

III. LU-BAGVBRAP PROBLEM

In this section, we consider the LU-BAGVBRAP problem, a special case of the BAGVBRAP problem, wherein the input satisfies the local uniformity property. We develop an algorithm with an approximation ratio of 5. The motivation for studying LU-BAGVBRAP comes from the fact that it captures the MULTIBAGUBRAP problem, a multi-system generalization of the BAGUBRAP problem. We will first present the 5-approximation algorithm for the LU-BAGVBRAP problem. Then, we will show how to reduce MULTIBAGUBRAP to LU-BAGVBRAP.

A. A 5-approximation for the LU-BAGVBRAP Problem

We start by recollecting the problem definition of LU-BAGVBRAP.

LU-BAGVBRAP Problem Definition: This is the special case of the BAGVBRAP problem wherein the input satisfies *local uniformity property*: for every job instance $u \in \mathcal{U}$, the bandwidth available during its time interval does not vary (i.e., for any $t_1, t_2 \in I_u$, $B_{t_1} = B_{t_2}$).

Our 5-approximation algorithm for the LU-BAGVBRAP problem and its analysis closely follow the approximation algorithm and analysis presented in Section II. We exploit the local uniformity property to strengthen some crucial claims in the above analysis and obtain the 5-approximation algorithm. We now present the approximation algorithm and highlight the modifications to be made in the algorithm and analysis of Section II.

Theorem 3.1: The LU-BAGVBRAP problem can be approximated within a factor of 5.

We now provide a proof sketch of the above theorem. As in Section II, we divide the instances into two categories – large instances and small instances. Consider an instance u . By the local uniformity property, the bandwidth available is uniform throughout the interval I_u ; let this bandwidth be B . We say that u is a *large* instance, if $\rho_u > B/2$; it is said to be *small* otherwise. Partition the set of all instances \mathcal{U} into \mathcal{U}_l and \mathcal{U}_s , where \mathcal{U}_l is the set of all large instances and \mathcal{U}_s is the set of all small instances.

Let A^* denote the optimal solution and let $P^* = \text{Profit}(A^*)$. Let A_l^* be the optimal solution when only the large job instances are considered. Similarly, let A_s^* be the optimal solution when only the small job instances are considered. Let $P_l^* = \text{Profit}(A_l^*)$ and $P_s^* = \text{Profit}(A_s^*)$. We have $P^* \leq P_l^* + P_s^*$.

Below, we develop two algorithms that deal with large instances and small instances, respectively. The first algorithm outputs a feasible solution A_l consisting of only large instances having profit $P_l = \text{Profit}(A_l)$ such that $P_l^* \leq 2P_l$. The second algorithm outputs a feasible solution A_s consisting of only small instances having profit $P_s = \text{Profit}(A_s)$ such that $P_s^* \leq 3P_s$. Of the two solutions A_l and A_s , we output the one with the higher profit as the final solution; we denote the output solution as A . Let

$P = \text{Profit}(A)$. It follows that $P^* \leq 5P$. This establishes Theorem 3.1.

Handling Large Instances

We now describe the algorithm that deals with large instances. First, we strengthen Claim 2.2 as follows.

Claim 3.2: Any feasible solution S cannot contain two or more overlapping instances from \mathcal{U}_l .

Proof: By contradiction, suppose S contains two overlapping large instances u_1 and u_2 . Let the starting timeslot of u_1 be t_1 and that of u_2 be t_2 . Without loss of generality, assume that $t_1 \leq t_2$. Then, both u_1 and u_2 are active at the timeslot t_2 . Let $B = B_{t_2}$ be the bandwidth available at t_2 . Since both the instances are large, we have that $\rho_{u_1} > B/2$ and $\rho_{u_2} > B/2$. It follows that $\rho_{u_1} + \rho_{u_2} > B$, violating the bandwidth constraint. This contradicts the fact that S is a feasible solution. \square

Our algorithm to compute A_l is as follows. We observe that the problem of computing P_l^* is the same as the BAGUNITRAP problem over the set \mathcal{U}_l . We invoke the 2-approximation algorithm for the BAGUNITRAP problem, due to Bar-Noy et al. [5] and Berman and Dasgupta [6], with \mathcal{U}_l as input and obtain a solution; A_l is taken to be this solution. We conclude that that $P_l^* \leq 2P_l$.

Handling Small Instances:

We now describe the algorithm that deals with the small instances. The algorithm and analysis are similar to those presented in Section II-B. We use the same LP relaxation and the same rounding procedure given in Figure 1. The rounding procedure outputs a list \mathcal{L} of feasible solutions S_1, S_2, \dots, S_m , with weights $w(S_1), w(S_2), \dots, w(S_m)$. Once the list \mathcal{L} is constructed, we output the solution from S_1, S_2, \dots, S_m that has the maximum profit; the output solution is taken to be A_s . Let the profit of A_s be denoted by P_s . As before, we will argue that the list satisfies four properties. The only modification is that we strengthen property four as follows:

$$(4^*) \quad \sum_{k=1}^m w(S_k) \leq 3.$$

We next show that the sets in the list \mathcal{L} satisfy the four properties. By the argument given Section II-B, this would imply that $P_s^* \leq 3P_s$. It is easy to see that Properties 1, 2 and 3 are satisfied by \mathcal{L} . We proceed to show that \mathcal{L} satisfies property 4*. We shall show that this property is an invariant of the algorithm. Note that a new set may be added to the list \mathcal{L} either from step 3 or from step 4a. In the latter case, the weight of a set is split amongst itself and the newly created set, leaving the sum of all weights unaffected.

In the former case, the newly created set is a singleton consisting of one instance, say u . Let J be the job to which the instance u belongs. Recall that the instances are processed in the increasing order of their start times. Consider the set \mathcal{L} immediately after this singleton is added. Partition the list \mathcal{L} into \mathcal{L}_1 and \mathcal{L}_2 , where \mathcal{L}_1 consists of

all sets in \mathcal{L} that contain some instance of J (including u itself); \mathcal{L}_2 consists of all the sets in \mathcal{L} that do not contain any instance of J (i.e., $\mathcal{L}_2 = \mathcal{L} - \mathcal{L}_1$). The following claim is proved similar to Claim 2.6.

Claim 3.3: $\sum_{S \in \mathcal{L}_1} w(S) \leq 1$.

Now, let us consider the list \mathcal{L}_2 . For a timeslot $t \in I_u$ and a set $S \in \mathcal{L}$, let $\rho_t(S)$ be the sum of the bandwidth requirements of the instances in S that are active at time t , i.e.,

$$\rho_t(S) = \sum_{j \in S \cap A(t)} \rho_j$$

The following claim is proved similar to the Claim 2.7.

Claim 3.4: For any timeslot $t \in [1, D]$

$$\sum_{S \in \mathcal{L}} \rho_t(S) w(S) \leq B_t$$

We strengthen Claim 2.8 as follows.

Claim 3.5: For any $1 \leq k \leq r$,

$$\sum_{S \in \mathcal{L}_2} w(S) \leq 2$$

Proof: Let t_1 be the starting timeslot of u . Consider any set $S \in \mathcal{L}_2$. We know that u could not be packed in S . The reason could be twofold: either the bag constraint or the bandwidth constraint is violated. However, by the definition of \mathcal{L}_2 , S does not contain any instance from J . This implies that u was not packed in S because of a violation of the bandwidth constraint. It follows that there must exist a timeslot $t \in I_u$ such that $\rho_t(S) + \rho_u \geq B_t$. Since the algorithm considers the instances in increasing start times, all the instances in S that are active at timeslot t are also active at t_1 . Therefore, $\rho_{t_1}(S) \geq \rho_t(S)$. By the local uniformity property, we have that $B_t = B_{t_1}$. Thus, $\rho_{t_1}(S) + \rho_u \geq B_{t_1}$. Since u is a small instance, we have that $\rho_u \leq B_{t_1}/2$. Hence, $\rho_{t_1}(S) \geq B_{t_1}/2$. Invoking Claim 3.4 at the timeslot t_1 , we get that

$$\sum_{S \in \mathcal{L}_2} \rho_{t_1}(S) w(S) \leq B_{t_1}.$$

It follows that

$$\sum_{S \in \mathcal{L}_2} (B_{t_1}/2) w(S) \leq B_{t_1}.$$

The claim is proved. \square

Combining Claims 3.3 and 3.5, we get that

$$\sum_{S \in \mathcal{L}} w(S) \leq 3.$$

Thus, we have shown that \mathcal{L} satisfies property 4*. This completes the proof sketch of Theorem 3.1.

B. Approximating MULTIBAGUBRAP

We now describe the MULTIBAGUBRAP problem and show how it can be reduced to the LU-BAGVBRAP problem. As a consequence, we shall derive a 5-approximation algorithm for MULTIBAGUBRAP. Recall that the BAGUBRAP problem is the special case of BAGVBRAP in which the available bandwidth is same across all time slots, i.e, $B_t = B$ for all time instances $t \in [1, D]$. Meaning, we have a single system/machine offering a uniform bandwidth of B across the time span $[1, D]$. Now, suppose we have multiple such machines, each offering a uniform bandwidth. The bandwidths offered may vary from machine to machine. Namely, there are m machines and the bandwidth offered by the machine r is $B(r)$ throughout the timespan $[1, D]$. We consider the natural problem of profit maximization when the job instances can be scheduled on any of the machines as long as (i) the bag constraints are satisfied and (ii) for each machine r and at each instant of time t , the sum of bandwidth requirement of job instances active at timeslot t on machine r does not exceed $B(r)$. We call this the MULTIBAGUBRAP problem.

We now present a reduction from MULTIBAGUBRAP to LU-BAGVBRAP. As a result of the reduction, we also get a 5-approximation for the MULTIBAGUBRAP problem. The idea of the reduction is as follows.

In the MULTIBAGUBRAP problem input, suppose the span is $[1, D]$ so that all the instances have finish time at most D . We will create an input of LU-BAGVBRAP having span $[1, mD]$. For $1 \leq r \leq m$, the integer interval $[1 + (r - 1)D, rD]$ will be used to simulate the machine r . Consider a feasible solution to the MULTIBAGUBRAP input. Instead of viewing the jobs as executing concurrently on the m machines in the span $[1, D]$, we adopt the following way of viewing the execution. Suppose a job instance u specified by the interval $I_u = [a, b]$ is scheduled in the solution to be executed on the machine r ; we will view it as executing during the interval $[(i - 1) \cdot D + a, (i - 1) \cdot D + b]$.

Now given an input instance of MULTIBAGUBRAP, we show how to transform it to create an input instance of BAGVBRAP. The span in the new input instance will be $[1, mD]$. For each job instance u with associated interval $I_u = [a, b]$, we create m copies u ; each copy will have the same bandwidth requirement and profit as the instance u ; for $1 \leq i \leq m$, the i^{th} copy of u is declared to have the interval $I_u^i = [(i - 1)D + a, (i - 1)D + b]$; if $J \in \mathcal{J}$ is the job to which the instance u belongs, then all the m copies of u are made part of same job in the transformed input. So, if a job earlier had x instances, it will have mx instances in the transformed input. In the transformed input, the bandwidth available for the timeslots (in the span $[1, mD]$) is declared as follows. For $1 \leq r \leq m$, each timeslot t in the range $[1 + (r - 1)D, rD]$ will have bandwidth available B_t as $B(r)$, where $B(r)$ is the bandwidth offered

by the machine r . It is easy to see that the transformed input satisfies the local uniformity property and there is a one-to-one correspondence between the feasible solutions of the MULTIBAGUBRAP instance and the feasible solutions of the transformed LU-BAGVBRAP instance.

The above reduction combined with 5-approximation algorithm for the LU-BAGVBRAP problem yields a 5-approximation algorithm for the MULTIBAGUBRAP problem.

Theorem 3.6: The MULTIBAGUBRAP problem can be approximated within a factor of 5.

REFERENCES

- [1] T. Erlebach and F. Spieksma, "Interval selection: Applications, algorithms, and lower bounds," *Journal of Algorithms*, vol. 46, no. 1, pp. 27–53, 2003.
- [2] S. Albers, "Resource Management in Large Networks," in *Algorithmics for Large and Complex Networks: Design, Analysis, and Simulation*, J. Lerner, D. Wagner, and K. Zweig, Eds. Springer Berlin/Heidelberg, 2009, pp. 227–246.
- [3] M. Flammini, G. Monaco, G. Moscardelli, H. Shachnai, M. Shalom, T. Tamir, and S. Zaks, "Minimizing total busy time in parallel scheduling with application to optical networks," in *23rd IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2009.
- [4] M. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1980.
- [5] A. Bar-Noy, S. Guha, J. Noar, and B. Schieber, "Approximating the throughput of multiple machines in real-time scheduling," *Siam Journal of Computing*, vol. 31, no. 2, pp. 331–352, 2001.
- [6] P. Berman and B. Dasgupta, "Multi-phase algorithms for throughput maximization for real-time scheduling," *Journal of Combinatorial Optimization*, vol. 4, pp. 307–323, 2000.
- [7] F. Spieksma, "On the approximability of an interval scheduling problem," *Journal of Scheduling*, vol. 2, pp. 215–227, 1999.
- [8] G. Calinescu, A. Chakrabarti, H. Karloff, and Y. Rabani, "Improved approximation algorithms for resource allocation," in *Proceedings of the 9th International Conference on Integer Programming and Combinatorial Optimization*, 2002.
- [9] C. Phillips, R. Uma, and J. Wein, "Off-line admission control for general scheduling problems," in *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2000.
- [10] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Noar, and B. Schieber, "A unified approach to approximating resource allocation and scheduling," *Journal of the ACM*, vol. 48, no. 5, pp. 1069–1090, 2001.
- [11] A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar, "Approximation algorithms for the unsplittable flow problem," *Algorithmica*, vol. 47, no. 1, pp. 53–78, 2007.

- [12] C. Chekuri, M. Mydlarz, and F. Shepherd, "Multicommodity demand flow in a tree and packing integer programs," *ACM Transactions on Algorithms*, vol. 3, no. 3, 2007.
- [13] N. Bansal, Z. Friggstad, R. Khandekar, and M. Salavatipour, "A logarithmic approximation for unsplittable flow on line graphs," in *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2009, pp. 702–709.
- [14] N. Bansal, A. Chakrabarti, A. Epstein, and B. Schieber, "A quasi-ptas for unsplittable flow on line graphs," in *ACM Symposium on Theory of Computing*, 2006, pp. 721–729.
- [15] S. Leonardi, A. Marchetti-Spaccamela, and A. Vitaletti, "Approximation algorithms for bandwidth and storage allocation problems under real time constraints," in *FSTTCS*, 2000.