Real-time Scheduling with a Budget*

Joseph (Seffi) Naor† Hadas Shachnai‡ Tami Tamir§

January 19, 2006

Abstract

Suppose that we are given a set of jobs, where each job has a processing time, a non-negative weight, and a set of possible time intervals in which it can be processed. In addition, each job has a processing cost. Our goal is to schedule a feasible subset of the jobs on a single machine, such that the total weight is maximized, and the cost of the schedule is within a given budget. We refer to this problem as *budgeted real-time scheduling (BRS)*. Indeed, the special case where the budget is *unbounded* is the well-known real-time scheduling problem. The second problem that we consider is *budgeted real-time scheduling with overlaps (BRSO)*, in which several jobs may be processed simultaneously, and the goal is to maximize the time in which the machine is utilized. Our two variants of this real-time scheduling problem have important applications, in vehicle scheduling, linear combinatorial auctions, and Quality-of-Service management for Internet connections. These problems are the focus of this paper.

Both BRS and BRSO are strongly NP-hard, even with unbounded budget. Our main results are $(2+\varepsilon)$-approximation algorithms for these problems. This ratio coincides with the best known approximation factor for the (unbudgeted) real-time scheduling problem, and is slightly weaker than the best known approximation factor of $e/(e-1)$ for the (unbudgeted) real-time scheduling with overlaps, presented in this paper. We show that better ratios (or simpler approximation algorithms) can be derived for some special cases, including instances with unit-costs and the budgeted *job interval selection problem (JISP)*. Budgeted JISP is shown to be APX-hard even when overlaps are allowed and with unbounded budget. Finally, our results can be extended to instances with multiple machines.

Keywords: Real-time scheduling, job interval selection, approximation algorithms.

*A preliminary version of this paper appeared in the Proceedings of the 30th International Colloquium on Automata, Languages, and Programming, Eindhoven, The Netherlands, 2003.

†Computer Science Dept., Technion, Haifa 32000, Israel. E-mail: naor@cs.technion.ac.il. This research is supported in part by a foundational and strategical research grant from the Israeli Ministry of Science, and by a US-Israel BSF Grant 2002276.

‡Computer Science Dept., Technion, Haifa 32000, Israel. E-mail: hadas@cs.technion.ac.il. On leave at Bell Laboratories, Lucent Technologies, 600 Mountain Ave., Murray Hill, NJ 07974.

§School of Computer Science, The Interdisciplinary Center, Herzliya, Israel. E-mail: tami@idc.ac.il. Work done while the author was at the University of Washington.

1 Introduction

In the well-known *real-time scheduling* problem (also known as the *throughput maximization* problem), we are given a set of n jobs; each job J_j has a processing time p_j, a non-negative weight w_j, and a set of time intervals in which it can be processed (given as either a window with release and due-dates or as a discrete set of possible processing intervals). The goal is to schedule a feasible subset of the jobs on a single machine, such that the overall weight of the scheduled jobs is maximized. In this paper we consider two variants of this problem.

> In the *budgeted real-time scheduling (BRS)* problem, each job J_j has a processing cost c_j. A budget B is given, and the goal is to find a maximum weight schedule, among the feasible schedules whose total processing cost is at most B.
>
> In *real-time scheduling with overlaps (RSO)*, the jobs are scheduled on a single *non-bottleneck* machine, which can process simultaneously several jobs. The goal is to maximize the overall time in which the machine is utilized (i.e., processes at least one job).[1] In the budgeted case ($BRSO$), each job J_j has a processing cost c_j. The goal is to maximize the time in which the machine is utilized, among the schedules with total processing cost at most B.

In our study of BRS, RSO and BRSO, we distinguish between *discrete* and *continuous* instances. In the discrete case, each job J_j can be scheduled to run in one of a given set of n_j intervals $I_{j,\ell}$ ($\ell = 1, \ldots, n_j$). If each job has at most k possible intervals, that is, $\forall j, n_j \leq k$, then this version is called $JISP_k$. In the continuous case, job J_j has release date r_j, due date d_j, and a processing time p_j. It is possible to schedule J_j in any interval $[s_j, e_j]$ such that $s_j \geq r_j$, $e_j \leq d_j$, and $e_j = s_j + p_j$. We consider also the special case, denoted $JISP_1$, where each job can be processed only in the single interval $I_{j,1} = [r_j, d_j]$, and $p_j = d_j - r_j$.

We consider general (discrete and continuous) instances, where each job has processing time p_j, a weight w_j, and a processing cost c_j. For some variants we study also classes of instances in which (i) jobs have unit-costs (that is, $c_j = 1\ \forall j$), or (ii) for all the jobs $w_j = p_j$.

The BRS and BRSO problems extend the classic real-time scheduling problem to model the natural goal of gaining the maximum available service for a given budget. In particular, the following practical scenarios yield instances of our problems.

Multi-Vehicle Scheduling on a Path The vehicle scheduling problem arises in many applications, including robot handling in manufacturing systems and secondary storage management in computer systems (see e.g. [KN-01]). Suppose that a fleet of vehicles needs to service requests on a path. There is an operation cost to each vehicle, and a segment on

[1] Note that job weights have no effect on the objective function.

the line in which the vehicle can provide service. Our objective is to assign the vehicles to service requests on line segments such that the total length of the union of the line segments, i.e., the part of the line which is covered, is maximized, yet the overall cost is within some budget constraints. This yields an instance of the continuous BRS problem.

Transmission of Continuous-media Data In multimedia-on-demand applications, as well in scheduling programs in cable TV, a company that wishes to broadcast a program needs to rent a channel for the duration of the program. Consider a company that wishes to assign a set of programs/advertisements in a set of possible time slots from a given time interval $[0, T]$. Each program is targeted at specific audience and has a *revenue* associated with each time slot, defined to be the percentage of the viewers (from the targeted audience) in this time slot. There is also a cost associated with the assignment of a program to a certain slot which is non-uniform, e.g., it increases in peak hours. The company needs to schedule a subset of its programs such that the overall cost for renting the channel is within a given budget. Since each time slot is assigned to a single company, this is an instance of BRS.

Crew Scheduling Consider a system that operates in shifts, where each shift requires the assignment of a crew. Any crew member provides several possible shifts in which he/she can be assigned. Also, there is an individual payment associated with each crew member per shift. The system has a certain budget, and the goal is to operate a maximum number of shifts within the given time interval, under the budget constraints.

Combinatorial Auctions In auctions used in e-commerce, a buyer needs to complete an order for a given set of goods. There is a collection of *sellers*, each offers a subset (or *bundle*) of the goods at some cost. Each of the goods g_i is associated with a weight w_i, which indicates its priority in the order. The buyer needs to satisfy a fraction of the order of maximum weight, by selecting a subset of the offers, such that the total cost is bounded by the buyer's budget, B. In auctions for *linear* goods (see, e.g., in [T-00]), we have an ordered list of m goods $g_1, \ldots, g_m$, and the offers should refer to bundles of the form $g_i, g_{i+1}, \ldots, g_{j-1}, g_j$. Note that, while selecting a subset of the offers overlaps are allowed, i.e., the buyer may acquire more than the needed amount from some good; however, this does not decrease the cost of any of the offers. Thus, this is an instance of the *BRSO* problem, where any job J_j can be processed in one possible time interval.

Recovery from Power Outage Suppose that a service line needs to be fixed, following a power outage, within a given time interval $[0, T]$. Several companies can send workers to fix various parts of the line. The team of company i can handle within T time units the recovery of a path of length l_i, at the cost c_i. The team can work on any segment between two endpoints s_i, f_i (that depend on the location of the company's service station). The

electricity company has a certain budget and the goal is to assign parts of the line to a subset of the companies, such that the overall length that is fixed by the time T is maximized. In this instance of our problem, two companies whose areas overlap may be selected, so that the recovery in the area is handled more efficiently; thus, this is an instance of continuous $BRSO$.

QoS Upgrade in a Network Consider an end-to-end connection between s and t that uses several Internet service providers (ISP). Each ISP provides a basic service (for free) and to upgrade the service one needs to pay; that is, an ISP can decrease the delay in its part of the path for a certain cost. (See, e.g., [LORS-00, LO-02].) The end-to-end delay is additive (over all ISP-s). The limited budget needs to be distributed between the ISP-s. In certain scenarios, an ISP may need to choose to upgrade only a portion of the part of the $s-t$ path that it controls, however, it has the freedom to choose which portion. In this problem instance, "jobs" (upgraded segments) are allowed to overlap, therefore this is an instance of BRSO.

1.1 Our Results

We give hardness results and approximation algorithms for BRS, RSO, and $BRSO$.

Hardness results We show that continuous RSO is strongly NP-hard.[2] In the discrete case, both BRS and $BRSO$ are shown to be APX-hard, already for instances where $\forall j, n_j \leq k$ ($JISP_k$), and where all the intervals corresponding to a job have the same length, for any $k \geq 3$.

Approximating BRS We present a $(2+\varepsilon)$-approximation algorithm for BRS (both discrete and continuous). We build on the framework of Jain and Vazirani [Va-01] for using Lagrangian relaxation in developing approximation algorithms. Our algorithm is based on a novel combination of Lagrangian relaxation with efficient search on the set of feasible solutions. We show that a simple Greedy algorithm yields a 4-approximation for discrete BRS with unit costs, where $w_j = p_j$ for all j; the same algorithm achieves the ratio $(4+\varepsilon)$ in the continuous case. We demonstrate the applicability of our techniques to the budgeted maximum weight matching problem and present a fully polynomial time approximation scheme for it.

Approximating RSO and BRSO We give a $(2+\varepsilon)$-approximation algorithm for continuous inputs of $BRSO$, and a $(3+\varepsilon)$-approximation for discrete inputs, using the Lagrangian relaxation technique. For RSO we present a greedy algorithm that achieves the ratio of

[2] The continuous real-time scheduling problem (with no overlaps) is known to be strongly NP-hard [GJ-79].

2. An improved ratio of $e/(e-1)$ is obtained by a randomized algorithm (where e denotes the base of the natural logarithm). For $JISP_1$, we obtain an optimal solution for instances of $BRSO$ with unit costs, and a *fully polynomial time approximation scheme (FPTAS)* for arbitrary costs. Note that $JISP_1$ is weakly NP-hard. This can be shown by reduction from knapsack [GJ-79].

Extensions Our results are shown to extend to instances of BRS and $BRSO$ in which the jobs can be scheduled on multiple machines.

The approximation technique that we use for deriving our $(2+\varepsilon)$-approximation results for BRS and $BRSO$ (see in Section 2) is shown to apply to a fundamental class of budgeted maximization problems, including *throughput maximization in a system of dependent jobs*, which generalizes the BRS problem. The input is a set of jobs whose dependencies can be modeled by a general conflict graph; the goal is to schedule a feasible subset of the jobs of maximum weight, such that the overall processing cost is within a given budget. The BRS problem is the special case where the conflicts among the jobs can be modeled as an interval graph. We show that, using the technique, any problem in the class which has an LP based ρ-approximation with unbounded budget, can be approximated within factor $\rho+\varepsilon$ in the budgeted case, for any $B \geq 1$.

1.2 Related Work

To the best of our knowledge, the *budgeted* real-time scheduling problem is studied here for the first time. There has been extensive work on real-time scheduling, both in the discrete and the continuous models. Garey and Johnson (cf. [GJ-79]) showed that the continuous case is strongly NP-hard, while the discrete case, JISP, was shown by Spieksma [S-99] to be APX-hard, already for instances of $JISP_k$, where $k \geq 2$. Bar-Noy et al. [BG$^+$99, BB$^+$01] and independently Berman and DasGupta [BD-00] presented 2-approximation algorithms for the discrete case,[3] and $(2+\varepsilon)$ ratio in the continuous case (also known as the *throughput maximization* problem). As shown in [BB$^+$01], this ratio holds for arbitrary number of machines. While none of the existing techniques has been able to improve upon the 2 and $(2+\varepsilon)$ ratios for general instances of the real-time scheduling problem, improved bounds were obtained for some special cases. In particular, Chuzhoy et al. [COR-01] considered the unweighted version, for which they gave an $(e/(e-1)+\varepsilon)$-approximation algorithm, where ε is any constant. For other special cases, they developed polynomial time approximation schemes. Finally, some special cases of JISP were shown to be polynomially solvable (see, e.g., in [AS-87, B-99]).

We are not aware of previous work on the RSO and $BRSO$ problems. Since overlaps are allowed, and the goal is to maximize the overall time in which the machine is utilized, these problems can be viewed as maximum coverage problems. In previous work on budgeted

[3] A 2-approximation for *unweighted* JISP is obtained by a Greedy algorithm, as shown in [S-99].

covering (see, e.g., [KMN-99]), the covering items are *sets*; once a set is selected, the covered elements are uniquely defined. In contrast, in *RSO* (and *BRSO*) the covering items are *jobs*, and we can choose the time segments (= elements) that will be covered by a job, by determining the time interval in which this job is processed.

Recent developments In a recent paper [L-05], Levin considered the discrete variants of *BRS* and *BRSO*. The paper shows that by applying a *parametric* search on the set of feasible solutions, one can eliminate the additive factor of ε from the approximation ratios, thus obtaining an improved bound of 2 for discrete BRS, and of 3 for discrete *BRSO*.

2 Approximation via Lagrangian Relaxation

2.1 The Technique

We describe below the general approximation technique that we use for deriving our results for *BRS* and *BRSO*. Our approach builds on the framework developed by Jain and Vazirani [Va-01][pp. 250-251] (see also [Ga-96]), for using Lagrangian relaxations in approximation algorithms. Our approach applies to the following class of *subset selection* problems. The input for any problem in the class consists of a set of elements $A = \{a_1, \ldots, a_n\}$; each element $a_j \in A$ is associated with a weight, w_j, and the cost of adding a_j to the solution set is $c_j \geq 1$. We have a budget $B \geq 1$. The goal is to find a subset of the elements $A' \subseteq A$ satisfying a given set of constraints $C_1, \ldots, C_r$, together with a budget constraint, such that the total weight is maximized. We assume that any problem Π in the class satisfies the following property:

(P1) Let A' be a feasible solution for Π; then, any subset $A'' \subseteq A'$ is also a feasible solution.

Denote by $x_j \in \{0, 1\}$ the indicator variable for the selection of a_j. The integer program for Π has the following form.

$$(\Pi) \qquad \text{maximize} \qquad \sum_{a_j \in A} w_j x_j$$

subject to:

$$\text{Constraints} : C_1, \ldots, C_r$$

$$\sum_j c_j x_j \leq B.$$

In the linear relaxation, $x_j \in [0, 1]$. The Lagrangian relaxation of the budget constraint is:

$$(L - \Pi(\lambda)) \qquad \text{maximize} \qquad \lambda \cdot B \; + \sum_{a_j \in A} (w_j - c_j \lambda) x_j$$

subject to:

$$\text{Constraints} : C_1, \ldots, C_r$$

Assume that $\mathcal{A}_\pi$ is a ρ-approximation algorithm with respect to the optimal integral solution for $L - \Pi(\lambda)$, for any value of $\lambda > 0$. Thus, there exist values $\lambda_1 < \lambda_2$ such that $\mathcal{A}_\pi$ finds integral ρ-approximate solutions $\mathbf{x}_1, \mathbf{x}_2$ for $L - \Pi(\lambda_1)$, $L - \Pi(\lambda_2)$, respectively, and the budgets used in these solutions are B_1, B_2, where

$$B_2 < B < B_1. \tag{1}$$

Let W_1, W_2 denote the weights of the solutions $\mathbf{x}_1, \mathbf{x}_2$, then $W_i = \lambda_i B + \sum_{a_j \in A}(w_j - c_j\lambda_i)\mathbf{x}_{ij}$, $i \in \{1, 2\}$, $1 \leq j \leq n$. Without loss of generality, we can assume that $W_1, W_2 \geq 1$.

Following the framework of [Va-01], we require that $\mathcal{A}_\pi$ satisfies the following property. Let $\alpha = (B - B_2)/(B_1 - B_2)$, then the convex combination of the solutions $\mathbf{x}_1, \mathbf{x}_2$, namely, $\mathbf{x} = \alpha\mathbf{x}_1 + (1-\alpha)\mathbf{x}_2$ is a (fractional) ρ-approximate solution that does not exceed the budget B. This is indeed the case if, for example, the solutions $\mathbf{x}_1, \mathbf{x}_2$ are obtained from a primal-dual algorithm. In this case, a convex combination of the dual solutions corresponding to $\mathbf{x}_1$ and $\mathbf{x}_2$ can be used to prove this property. This will be heavily used in our algorithms for the BRS and $BRSO$ problems.

Our goal is to find a feasible integral solution whose weight is close to the weight of $\mathbf{x}$. We show that for the class of subset selection problem that we consider here, by finding 'good' values of λ_1, λ_2, we obtain an integral solution that is within factor $\rho + \varepsilon$ from the optimal. The running time of our algorithm is dominated by the complexity of the search for λ_1, λ_2 and the running time of $\mathcal{A}_\pi$.

We now summarize the steps of the algorithm, $\mathcal{A}_L$, which gets as input the set of elements $a_1, \ldots, a_n$, an accuracy parameter $\varepsilon > 0$ and a budget $B \geq 1$.

1. Given $\varepsilon > 0$, let $\varepsilon' \in (0, \varepsilon)$ (to be determined).
2. Define the *modified weight* of an element a_j to be $w'_j = w_j/c_j$.
   Let $\omega_1 \leq \cdots \leq \omega_R$ be the set of R distinct values of modified weights.
3. Find in $(0, \omega_R)$ the values of $\lambda_1 < \lambda_2$
   satisfying (1), such that $\lambda_2 - \lambda_1 \leq \varepsilon'$.
4. Output the (feasible) integral solution found by $\mathcal{A}_\pi$ for $L - \Pi(\lambda_2)$.

Analysis The heart of our approximation technique is the following theorem.

Theorem 2.1 *For any $\varepsilon' > 0$ and λ_1, λ_2 satisfying (1), if $0 < \lambda_2 - \lambda_1 < \varepsilon'$, then $W_2 \geq W_1 - \varepsilon' c$, where $c = \sum_j c_j$ is the total cost of the instance.*

Proof: We note that, for a fixed value of λ, we can omit from the input elements a_j for which $w'_j = w_j/c_j \leq \lambda$. Denote by S_i the *feasible* set of modified weights for λ_i, i.e., the set of values ω_ℓ satisfying $\omega_\ell \geq \lambda_i$; then $S_2 \subseteq S_1$. Let A_i be the set of elements selected by $\mathcal{A}_\pi$ for the solution, for the given value λ_i, $i = 1, 2$. Clearly, for any element $a_j \in A_i$, $w'_j \in S_i$, $i = 1, 2$. We can write $W_1 = \lambda_1 B + \sum_{A_1}(w_j - c_j\lambda_1)$ and $W_2 = \lambda_2 B + \sum_{A_2}(w_j - c_j\lambda_2)$. We handle two cases separately.

(i) The feasible sets for λ_1, λ_2 are identical, that is, $S_1 = S_2$. Note that, for any $a_j \in A_1$, $w'_j = w_j/c_j \geq \lambda_2$. Assume, w.l.o.g., that

$$W_2 \geq \lambda_2 B + \sum_{A_1}(w_j - c_j\lambda_2)$$

(else, $\mathcal{A}_\pi$ can select A_1 for λ_2, without harming the approximation ratio). And since $\lambda_1 < \lambda_2 < \lambda_1 + \varepsilon'$, it follows that

$$W_2 > \lambda_1 B + \sum_{A_1}(w_j - c_j(\lambda_1 + \varepsilon')) = W_1 - \varepsilon' \sum_{A_1} c_j \geq W_1 - \varepsilon' c.$$

(ii) The feasible set for λ_1 contains some modified weights that are not in S_2, that is, $S_2 \subset S_1$. For simplicity, we assume that $S_1 = \{\omega_{\ell+1}, \omega_{\ell+2}, \ldots, \omega_R\}$, while $S_2 = \{\omega_{\ell+2}, \ldots, \omega_R\}$, i.e., for some $1 \leq \ell < R$, $\omega_{\ell+1} \in S_1$ and $\omega_{\ell+1} \notin S_2$. In general, several modified weight values may be contained in S_1 but not in S_2. A similar argument can be used in this case. Denote by $\hat{A}_1$ the subset of elements in A_1 whose modified weights are equal to $\omega_{\ell+1}$. Then, w.l.o.g., we may assume that

$$W_2 \geq \lambda_2 B + \sum_{A_1 \setminus \hat{A}_1} (w_j - c_j\lambda_2)$$

(else, A_2 can be replaced by $A_1 \setminus \hat{A}_1$, without harming the approximation ratio).

Hence, we get that

$$\begin{aligned} W_2 &\geq \lambda_1 B + \textstyle\sum_{A_1 \setminus \hat{A}_1}(w_j - c_j(\lambda_1 + \varepsilon')) \\ &= \lambda_1 B + \textstyle\sum_{A_1}(w_j - \lambda_1 c_j) - \sum_{\hat{A}_1}(w_j - \lambda_1 c_j) - \varepsilon' \sum_{A_1 \setminus \hat{A}_1} c_j \\ &= W_1 - \textstyle\sum_{\hat{A}_1} c_j(\omega_{\ell+1} - \lambda_1) - \varepsilon' \sum_{A_1 \setminus \hat{A}_1} c_j > W_1 - \varepsilon' \sum_{A_1} c_j \geq W_1 - \varepsilon' c \end{aligned}$$

The first inequality follows from the difference $(\lambda_2 - \lambda_1)$ being bounded by ε'; the second inequality follows from the fact that $\lambda_1 < \omega_{\ell+1} \leq \lambda_2$, i.e., $(\omega_{\ell+1} - \lambda_1) < \varepsilon'$. ∎

From Theorem 2.1 we get that

$$\begin{aligned} W_2 &\geq (W_1 - \varepsilon' c)\alpha + W_2(1 - \alpha) \geq (W_1\alpha + W_2(1 - \alpha)) - \varepsilon' c \\ &\geq (W_1\alpha + W_2(1 - \alpha))(1 - \varepsilon' c). \end{aligned}$$

The last inequality follows from the fact that $W_1, W_2 \geq 1$.

Let OPT denote the value of an optimal solution. Then, since the value $W_1\alpha + W_2(1-\alpha)$ is within factor of ρ from the optimal, we have that

$$W_2 \geq \frac{OPT(1-\varepsilon' c)}{\rho}.$$

Hence, given the input parameter $\varepsilon > 0$, by taking $\varepsilon' = \varepsilon/(c(\rho+\varepsilon))$, we get our main result.

Theorem 2.2 *Algorithm $\mathcal{A}_L$ achieves an approximation factor of $(\rho+\varepsilon)$ for* Π.

2.2 Implementation

A simple way to find the values of λ_1, λ_2 in Step 3 of Algorithm $\mathcal{A}_L$, is to use a binary search over the range $(0, \omega_R)$. This requires $O(\log(\max_j \frac{w_j}{c_j} \cdot \frac{c(\rho+\varepsilon)}{\rho}))$ steps, which is polynomial.

Indeed, as the weights and costs in the instance may be arbitrarily large, the search time may become large as function of the input size, n. In the following we show that, by allowing a small increase (of ε) in the approximation ratio, we can implement this binary search in time that is poly-logarithmic in n and $1/\varepsilon$, regardless of the weights and the costs of the elements.

Given an instance I and an accuracy parameter $\tilde{\varepsilon}$, (i) we initially guess the weight of an optimal integral solution, to within factor $(1-\tilde{\varepsilon})$; that is, we find a value $\tilde{W}$ satisfying $OPT(1-\tilde{\varepsilon}) \leq \tilde{W} \leq OPT$. This can be done in $O(\lg(n/\tilde{\varepsilon}))$ steps, since $\max_j w_j \leq OPT \leq n \cdot \max_j w_j$. We then omit from the input any element a_j whose weight is smaller than $\tilde{\varepsilon}\tilde{W}/n$. We scale the weights of the remaining elements, so that all the weights are in the range $[1, n/\tilde{\varepsilon}]$. (ii) For any element a_j with $c_j < \tilde{\varepsilon}B/n$, we round up c_j to $\tilde{\varepsilon}B/n$. We scale the other costs, such that all costs are in $[1, n/\tilde{\varepsilon}]$. (iii) We scale accordingly the size of the interval $(0, \omega_R)$. Denote the resulting instance by I'; the cost (weight) of a_j in I' is c'_j (w'_j).

Suppose that an optimal solution for I consists of the subset of elements A_o. We now show that the overall weight of A_o in I', denoted by OPT', is at least $OPT(1-2\tilde{\varepsilon})$. For any j, set $c_j = c'_j \cdot \tilde{\varepsilon}B/n$ and $w_j = w'_j \cdot \tilde{\varepsilon}\tilde{W}/n$. Then,

$$\begin{aligned}
OPT' &\geq \lambda_2 B + \sum_{a_j \in A_o} (w_j - \frac{\tilde{\varepsilon}\tilde{W}}{n}) - \sum_{a_j \in A_o} (c_j + \frac{\tilde{\varepsilon}B}{n})\lambda_2 \\
&\geq \lambda_2 B(1-\tilde{\varepsilon}) + \sum_{a_j \in A_o} (w_j - c_j\lambda_2) - \tilde{\varepsilon}\tilde{W} \\
&\geq OPT(1-2\tilde{\varepsilon}).
\end{aligned}$$

Hence, by setting the approximation parameter in the scaling and rounding procedure to be $\tilde{\varepsilon} = \frac{\varepsilon}{4(\rho+\varepsilon)}$, and in the algorithm A_L (applied to the instance I') to $\varepsilon' = \frac{\varepsilon}{2c(\rho+\varepsilon)}$, we get a $(\rho+\varepsilon)$-approximation.

The overall running time of A_L, when applying the rounding and scaling procedure, is $O(\log(\frac{n}{\tilde{\varepsilon}}) \cdot \log(\frac{n}{\tilde{\varepsilon}\varepsilon'})) = O((\log(\frac{n}{\varepsilon}))^2)$ times the running time of algorithm A_π.

2.3 Example: The Budgeted Maximum Weight Matching Problem

Let $G = (V, E)$ be an undirected graph and suppose that with each edge $e \in E$ we associate a *cost*, c_e, and a *weight*, w_e, which are non-negative integers. We consider the problem of finding a maximum weight matching in the graph G, such that the total cost of the edges in the matching does not exceed a given budget B. The problem is NP-hard by a reduction from knapsack [GJ-79].

Consider the following integer linear program for the problem. Let x_e be an indicator variable of edge $e \in E$, i.e., $x_e = 1$ if e belongs to the matching and $x_e = 0$ otherwise.

$$\text{maximize} \qquad \sum_{e \in E} w_e x_e$$

subject to:

$$\forall v \in V, \quad \sum_{e \in N(v)} x_e \leq 1$$

$$\sum_{e \in E} c_e x_e \leq B.$$

We now observe that the Lagrangian relaxation of this integer program is an instance of the maximum weight matching problem, meaning that $\rho = 1$. We can now apply our algorithm and obtain a fully polynomial time approximation scheme for the budgeted maximum weight matching problem.

3 Approximation Algorithms for BRS

3.1 Unit-cost Jobs

Consider the case where the jobs have unit costs, and $w_j = p_j$. Indeed, in this case, the budget B bounds the *number* of jobs that can be scheduled, and the goal is to maximize the total processing time, or *length* of the scheduled jobs. Alternatively, our goal is to maximize the total amount of time the machine is utilized. We show that, for such instances, we can obtain a 4-approximation, using a simple greedy algorithm.

Assume first that each job, J_j, can be scheduled in one specific interval, $[r_j, d_j]$, such that $d_j - r_j = p_j$ ($JISP_1$). Consider algorithm $\mathcal{A}_1$, which sorts the jobs in non-increasing

order by their processing times; then, $\mathcal{A}_1$ scans the sorted list and schedules the next job J_j if the machine is available in the time interval $[r_j, d_j]$. Formally,

1. Sort (and renumber) the jobs by their lengths such that $p_1 \geq p_2 \geq \ldots$
2. $j = 1$, $c = 0$
3. While $j \leq n$ and $c < B$
   If the interval $[r_j, d_j]$ is free {schedule J_j, $c = c + 1$}
   $j = j + 1$

Theorem 3.1 *Algorithm $\mathcal{A}_1$ yields a 3-approximation to the optimal utilization in $O(n \log n)$ steps.*

Proof: Let O and A denote the subsets of jobs (intervals) scheduled by an optimal algorithm and by $\mathcal{A}_1$, respectively. We use the next claims.

Claim 3.2 *If a job $J_j \in O$ does not overlap with any job in A, then $p_j \leq p_{min}(A)$, where $p_{min}(A)$ is the length of the shortest job in A.*

Proof: If J_j does not overlap with any job in A then it was not selected by $\mathcal{A}_1$ due to the budget constraint. Thus, $\mathcal{A}_1$ terminates before considering J_j, after having selected greedily B jobs, each of length at least p_j. ∎

Claim 3.3 *There is a mapping $f : O \rightarrow A$, such that any subset of jobs that is mapped to $J_j \in A$, satisfies $\sum_{J_k | f(k)=j} p_k \leq 3p_j$.*

Proof: Consider first the subset of jobs in O that overlap with jobs in A. Denote this subset by O_A. Define the mapping of jobs in O_A to jobs in A as follows. Scan the jobs in A by the order they were selected, and determine $f(k) = j$ for any job k in O_A that is not mapped yet and overlaps with J_j. Then omit these jobs from O_A.

At the time $J_j \in A$ is examined, J_j can overlap with at most one job, J_k that begins before r_j; at most one job, J_m, that ends after d_j; and a set of non-overlapping jobs L, such that for all $\ell \in L$, $r_\ell \geq r_j$ and $d_\ell \leq d_j$ (see Figure 1 in which $|L| = 1$). We bound the total length of these overlapping jobs.

(i) Suppose that $J_k \in O_A$ overlaps with J_j, and $r_k < r_j$. Since J_k does not overlap with any other job $J_r \in A$ with $p_r > p_j$, J_k was considered by $\mathcal{A}_1$ after J_j; thus, $p_k \leq p_j$.

(ii) Clearly, the total length of jobs in L - that are disjoint and in the interval $[r_j, d_j]$ - is at most p_j.

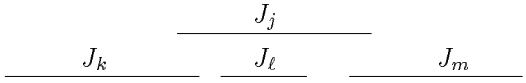

Figure 1: Possible overlaps of O_A jobs with a job $J_j \in A$

(iii) Finally, if J_j overlaps with some job $J_m \in O_A$, such that $r_m \geq r_j$ and $d_m > d_j$, then, as in (i), since J_m does not overlap with any J_r with $p_r > p_j$, $\mathcal{A}_1$ considered J_m after J_j; therefore, $p_k \leq p_j$.

So far the jobs of O_A were mapped to at most $|O_A|$ jobs in A. We now map the subset of jobs $\bar{O}_A$ (that do not overlap with any job in A) to the remaining jobs of A. Note that if $\bar{O}_A$ is non-empty, then $\mathcal{A}_1$ schedules exactly B jobs. Let $H \subseteq A$ denote the subset of jobs from A such that no job in O_A is mapped to a job in H. By the above, $|H| \geq B - |O_A| \geq |O| - |O_A| = \bar{O}_A$. Moreover, by Claim 3.2, any job $J_k \in \bar{O}_A$ satisfies $p_k \leq p_{min}(A)$; in particular any job in $\bar{O}_A$ is no longer than any job in H. Thus, each job $J_k \in \bar{O}_A$ can be mapped to some job J_j in H, and $p_k \leq p_j$. ■

The theorem now follows from Claim 3.3. The running time of algorithm $\mathcal{A}_1$ is dominated by the initial sorting step. ■

Towards obtaining an approximation algorithm for BRS, we consider now instances of $JISP_k$ where each job, J_j, is associated with a *set of intervals*, $\mathcal{I}_j$, each of length p_j; J_j can be scheduled in any of the intervals in $\mathcal{I}_j$. Algorithm $\mathcal{A}_2$ proceeds similar to $\mathcal{A}_1$, only that when scanning the sorted list of jobs, $\mathcal{A}_2$ schedules the next job, J_j, if the machine is available in *any* of the intervals in $\mathcal{I}_j$.

1. Sort (and renumber) the jobs by their lengths such that $p_1 \geq p_2 \geq \ldots$
2. $j = 1$, $c = 0$.
3. While $j \leq n$ and $c < B$
   If some interval in $\mathcal{I}_j$ is free
   {schedule J_j in the earliest such interval, $c = c + 1$}
   $j = j + 1$

We note that the choice to schedule a job in the earliest possible interval is arbitrary. Any schedule that do not overlap with already-selected intervals can do.

Theorem 3.4 *Algorithm $\mathcal{A}_2$ yields a 4-approximation to the optimal utilization, in $O(n \log n + \sum_j n_j)$ steps, where $n_j = |\mathcal{I}_j|$.*

Proof: The proof is similar to the proof of Theorem 3.1. Let O and A denote the subsets of jobs scheduled by an optimal algorithm and by $\mathcal{A}_2$, respectively. Denote by I_O and I_A

the sets of intervals selected in an optimal solution and by $\mathcal{A}_2$, respectively. We consider both the jobs and the intervals: each interval in I_O and I_A is associated with a job in O and a job in A, respectively. Claim 3.2 is generalized as follows.

Claim 3.5 *If $J_j \in O \setminus A$ is scheduled in O in an interval that does not overlap with any interval in I_A then $p_j \leq p_{min}(A)$, where $p_{min}(A)$ is the length of the shortest job in A.*

Proof: If J_j is not selected by $\mathcal{A}_2$, and the interval selected for it in the optimal solution does not overlap with any interval in I_A, then it was not selected by $\mathcal{A}_2$ due to the budget constraint. Thus, $\mathcal{A}_2$ terminates before considering J_j – after selecting greedily B jobs, each having length at least p_j. ■

The generalization of Claim 3.3 is valid with ratio 4, that is:

Claim 3.6 *There is a mapping $f : O \to A$, such that any subset of jobs that is mapped to $J_j \in A$ satisfies $\sum_{J_k | f(k)=j} p_k \leq 4p_j$.*

Proof: Consider first the subset of jobs in O whose selected intervals in I_O overlap with intervals in I_A, and the jobs in $O \cap A$. Note that a job which belongs to $O \cap A$ may be scheduled in different (in particular, non-overlapping) intervals in these solutions. Denote this subset by O_A. Define the mapping of jobs in O_A to jobs in A as follows. Scan the intervals in A by the order they were selected. Let $I_j \in \mathcal{I}_j$ be the next interval to be considered. Determine $f(k) = j$ for any job J_k in O_A that is not mapped yet, and whose selected interval in I_O overlaps with the interval selected for J_j in I_A. In addition, $f(j) = j$ if $j \in O \cap A$ and was not mapped yet. This may occur if the intervals selected for j in I_O and I_A are different and do not overlap with each other.

As in the proof of Claim 3.3, it holds that J_j can overlap with jobs of total length at most $3p_j$. In addition, if $f(j) = j$ but the corresponding intervals in I_O and I_A do not overlap, the total length of jobs from O mapped to j is at most $4p_j$. Having proved Claim 3.5, the remainder of the proof (mapping jobs in $O \setminus O_A$) is identical to this part in the proof of Theorem 3.1. ■

The statement of the theorem follows from Claim 3.6. The running time of algorithm $\mathcal{A}_2$ consists of the initial sorting step and the linear interval-consideration done for each job in the sorted list. ■

Note that, for continuous inputs, we can use the above algorithm $\mathcal{A}_2$ with a slight modification: instead of considering the sets of intervals associated with each job, we examine the corresponding time window, and while scanning the sorted list of jobs, we schedule J_j in the earliest possible time in (r_j, d_j). The ratio of 4 can be shown using arguments as in the proof of Theorem 3.4 since, once the jobs are selected for O and A, we may consider as before the *interval* associated with each job. As for the running time, we note that for each job considered by $\mathcal{A}_2$, we may need to examine at most n intervals within its time window, due to the selection of other jobs. Hence, we have

Corollary 3.7 *Algorithm $\mathcal{A}_2$ yields a 4-approximation for continuous BRS with unit costs, where $w_j = p_j$ for all j, in $O(n^2)$ steps.*

3.2 A $(2+\varepsilon)$-approximation Algorithm

In the following we derive a $(2+\varepsilon)$-approximation for discrete instances of BRS. A similar result can be obtained for the continuous case, by discretizing the instance (see in [BB+01]). Recall that in the discrete case, any job J_j can be scheduled in the intervals $I_{j,1}, \ldots, I_{j,n_j}$. We define a variable $x(j,\ell)$ for each interval $I_{j,\ell}$, $1 \leq j \leq n, 1 \leq \ell \leq n_j$. Then the integer program for the problem is:

$$
\begin{aligned}
(BRS) \qquad & \text{maximize} && \sum_{j=1}^{n} \sum_{\ell=1}^{n_j} w_j x(j,\ell) \\
\textit{subject to:} \qquad & && \forall j : \sum_{\ell=1}^{n_j} x(j,\ell) \leq 1 \\
& && \forall t : \sum_{t \in I_{j,\ell}} x(j,\ell) \leq 1 \\
& && \sum_{j=1}^{n} \sum_{\ell=1}^{n_j} c_j x(j,\ell) \leq B.
\end{aligned}
$$

In the linear program $x_j \in [0,1]$. Taking the Lagrangian relaxation, we get an instance of the *throughput maximization* problem. As shown in [BB+01], an algorithm based on the local ratio technique yields a 2-approximation for this problem, in $O(n^2)$ steps. This algorithm has a primal-dual interpretation; thus, we can apply the technique in Section 2 to obtain an algorithm, $\mathcal{A}$, which uses the algorithm for throughput maximization as a procedure.

Theorem 3.8 *Algorithm $\mathcal{A}$ is a $(2+\varepsilon)$-approximation algorithm for both discrete and continuous BRS. Its running time is $O((n\log(n/\varepsilon))^2)$, in the discrete case, and $O((n\log(n/\varepsilon))^2/\varepsilon)$, in the continuous case.*

4 Approximation Algorithms for RSO and $BRSO$

In this section, we present approximation algorithms for the RSO and $BRSO$ problems. In Section 4.1 we consider RSO. We give a randomized $e/(e-1)$-approximation algorithm for discrete inputs; then we describe a greedy algorithm that achieves a ratio of $(2-\epsilon)$ for continuous inputs, and $(3-\epsilon)$ for discrete inputs. In Section 4.2, we show that the greedy algorithm can be interpreted equivalently as a primal-dual algorithm. This allows us to apply the Lagrangian relaxation framework (Section 2) and to achieve a $(3+\varepsilon)$-approximation for $BRSO$ in the discrete case, where all the intervals corresponding to a job have the same length. For continuous inputs we obtain a $(2+\varepsilon)$-approximation algorithm.

4.1 The *RSO* Problem

In the RSO problem, we may select *all* the jobs, and the problem reduces to scheduling the jobs optimally so as to maximize the total amount of time in which the machine is utilized. Clearly, when $\forall j, p_j = d_j - r_j$, i.e., each job has only one possible interval, the schedule in which all the jobs are selected is optimal. When $\forall j, p_j \leq d_j - r_j$, the problem becomes hard to solve.

Theorem 4.1 *The RSO problem is strongly NP-hard.*

Proof: We show a reduction from 3-*partition*, which is strongly NP-hard [GJ-79]. An instance of 3-partition is defined as follows.

Input: a finite set A of $3q$ elements, a bound $B \in Z^+$, and a size $s(x)$ for each $x \in A$, such that each $s(x)$ satisfies $B/4 < s(x) < B/2$, and $\sum_{x \in A} s(x) = qB$.

Output: Is there a partition of A into q disjoint sets, $S_1, S_2, \ldots, S_q$, such that, for $1 \leq i \leq q, \sum_{x \in S_i} s(x) = B$? (Note that the above constraints on the element sizes imply that every such S_i must contain exactly three elements from A).

Given an instance of 3-partition, we construct an instance, I, for *RSO* such that for some u the maximum utilization is u iff A has a 3-partition.

The instance I consists of $4q - 1$ jobs. Each element $x \in A$ induces one job with $p_j = s(x), r_j = 0$, and $d_j = qB + q - 1$. For additional $q - 1$ jobs, $p_j = 1, r_j = j(B+1) - 1$, and $d_j = j(B+1)$. That is, each *additional* job must be schedule in a specific interval of length 1. Note that $\sum_{j \in I} p_j = qB + q - 1$, thus, in any schedule having utilization $qB + q - 1$ all the jobs must be scheduled with no overlaps at all. In such a schedule, the jth additional job is scheduled in the interval $[j(B+1) - 1, j(B+1)]$, and the jobs scheduled in the interval, of length B, between any two additional jobs, induce a triplet in the partition. Therefore, an optimal schedule induces a 3- partition. For a given partition, the additional jobs can be scheduled in their unique slots, and the jobs of each triplet in some idle B-interval. Since the additional slots generate exactly q idle B-intervals, all the jobs are scheduled and the utilization is $qB + q - 1$. ∎

4.1.1 A Randomized $e/(e-1)$-approximation algorithm

We start with a linear programming formulation of *RSO*. Assume that the input is given in a discrete fashion, and let $b_0, \ldots, b_m$ denote the set of start and end points (in sorted order), called *breakpoints*, of the time intervals $I_{j,\ell}$, $j = 1, \ldots, n$, $\ell = 1, \ldots, n_j$. We have a variable $x(j, \ell)$ for each interval $I_{j,\ell}$. For any pair of consecutive breakpoints b_{i-1} and b_i, the objective function gains $(b_i - b_{i-1})$ times the "coverage" of the interval $[b_{i-1}, b_i]$. Note that we take the minimum between 1 and the total cover, since we gain nothing if some interval is covered by more than one job.

$$(L-RSO) \quad \text{maximize} \qquad \sum_{i=1}^{m} \min\left(\sum_{j=1}^{n} \sum_{I_{j,\ell} \ni [b_{i-1},b_i]} x(j,\ell), 1\right) \cdot (b_i - b_{i-1})$$

subject to:

$$\text{For all jobs } J_j\text{:} \quad \sum_{\ell=1}^{n_j} x(j,\ell) \leq 1 \tag{2}$$

$$\text{For all } (j,\ell), \ell = 1,\ldots,n_j\text{:} \quad x(j,\ell) \geq 0. \tag{3}$$

We compute an optimal (fractional) solution to $(L-RSO)$. Clearly, the value of this solution is an upper bound on the value of an optimal integral solution. To obtain an integral solution, we apply randomized rounding to the optimal fractional solution. That is, for every job J_j, the probability that J_j is assigned to interval $I_{j,\ell}$ is equal to $x(j,\ell)$. If $\sum_{\ell=1}^{n_j} x(j,\ell) < 1$, then with probability $1 - \sum_{\ell=1}^{n_j} x(j,\ell)$ job J_j is not assigned to any interval.

We now analyze the randomized rounding procedure. Consider two consecutive breakpoints b and b'. Define for each job J_j, $y_j = \sum_{I_{j,\ell} \ni [b,b']} x(j,\ell)$. Clearly,

$$\sum_{j=1}^{n} y_j = \sum_{j=1}^{n} \sum_{I_{j,\ell} \ni [b,b']} x(j,\ell).$$

Without loss of generality, since each job J_j is assigned to a single interval, we can think of all the intervals of J_j that cover $[b,b']$ as a single (virtual) interval that is chosen with probability y_j. The probability that none of the virtual intervals is chosen is $P_0 = \prod_{j=1}^{n}(1-y_j)$. Let $r = \min(\sum_{j=1}^{n} y_j, 1)$. Then,

$$P_0 \leq \prod_{j=1}^{n}\left(1 - \frac{\sum_{i=1}^{n} y_i}{n}\right) = \left(1 - \frac{\sum_{i=1}^{n} y_i}{n}\right)^n < e^{-\sum_{i=1}^{n} y_i} \leq e^{-r}$$

Hence, the probability that $[b,b']$ is covered is

$$1 - P_0 \geq 1 - e^{-r} \geq \left(1 - \frac{1}{e}\right) \cdot r \geq \left(1 - \frac{1}{e}\right) \cdot \min\left(\sum_{j=1}^{n} y_j, 1\right)$$

Therefore, the expected contribution to the objective function of any interval $[b_{i-1}, b_i]$ is $(1-1/e) \cdot \min(\sum_{j=1}^{n} y_j, 1) \cdot (b_i - b_{i-1})$. By linearity of expectation, the expected value of the objective function after applying randomized rounding is

$$\left(1 - \frac{1}{e}\right) \cdot \sum_{i=1}^{m} \min\left(\sum_{j=1}^{n} \sum_{I_{j,\ell} \ni [b_{i-1},b_i]} x(j,\ell), 1\right) \cdot (b_i - b_{i-1}),$$

yielding an approximation factor of $1 - 1/e \approx 0.63212$.

4.1.2 A Greedy Approximation Algorithm

We now describe a greedy algorithm, which yields a $(2-\varepsilon)$-approximation for continuous instances of RSO, and $(3-\varepsilon)$-approximation for discrete instances. Assume that $\min_j r_j = 0$, and let $T = \max_j d_j$. Let I be the set of all the jobs in the instance; U is the set of unscheduled jobs. Denote by s_j, e_j the start-time and completion time of the job J_j in the greedy schedule, respectively. Given a partial schedule, we say that J_ℓ is *redundant* if we can remove J_ℓ from the schedule without decreasing the machine utilization. Algorithm Greedy proceeds in the interval $[0,T]$. The variable t denotes the time such that we can still schedule jobs in the interval $[t,T]$. When considering time t, the algorithm selects an arbitrary job, among the jobs J_i with $r_i < t$ and $d_i > t$. It schedules this job, J_k, such that its contribution to the utilization beyond time t is maximized. More specifically, the completion time of J_k is $\min(t+p_j, d_j)$. The following is a pseudocode of the algorithm.

Greedy
1. $U = I$, $t = 0$;
2. Let $J_j \in U$ be a job having $d_j > t$ and $r_j \leq t$.
   Schedule J_j such that its completion time, e_j, is $\min(t+p_j,\ d_j)$.
   Remove J_j from U.
   For any redundant job J_ℓ, omit J_ℓ from the schedule and return it to U.
3. Let $F \subseteq U$ be the set of unscheduled jobs, J_i,
having $d_i > e_j$.
Let $t_F = \min_{J_i \in F} r_i$, and let $t = \max(e_j,\ t_F)$.
If $F \neq \emptyset$ and $t < T$ go to step 2.

We use in the analysis the following properties of the greedy schedule.

Property 4.2 *Once an interval $[x_1, x_2] \in [0,T]$ is covered by* Greedy*, it remains covered until the end of the schedule.*

Proof: Follows from the fact that we remove from the schedule only redundant jobs. ■

Property 4.3 *When the algorithm considers time t, some job will be selected and scheduled such that for some $\varepsilon > 0$, the machine is utilized in the interval $[t, t+\varepsilon]$.*

Proof: The proof is by induction on the iteration number. In the first iteration $t = 0 = \min_j r_j$, and the selected job will be assigned to the interval $[0, p_j]$. We then determine (in step 3.) the value of t, to be the next time in $[0,T]$ in which the machine is idle, and some job J_j is available (that is, $r_j \leq t$ and $d_j > t$). Thus, if $F \neq \emptyset$ and $t < T$ we move to the next iteration in which at least one job can be scheduled. Finally, by step 2. of Greedy, the selected job, J_k, completes no later than $t+p_k$, i.e., J_k cannot start after t. In addition, the minimum completion time of J_k is $d_k = t+\varepsilon$ for some $\varepsilon > 0$. It follows that J_k is processed in $[t, t+\varepsilon]$. ■

Property 4.4 *Consider the set U of non-scheduled jobs at the end of the execution of* Greedy. *For any $J_j \in U$, the machine is utilized in the time interval $[r_j, d_j)$.*

Proof: Assume towards contradiction that, for some $J_j \in U$, the machine becomes idle within the interval $[r_j, d_j)$. Let $t_1 \in [r_j, d_j)$ be the earliest time that this occurs. Thus, $t_1 = 0$ or some job, J_k, is scheduled with $e_k = t_1$. We show that after scheduling J_k, we have in step 3. $F \neq \emptyset$, and we set $t = t_1$ (if $t_1 = 0$ then $t = t_1 = 0$ when the algorithm starts). We handle separately two cases.

(a) $J_j \in U$ when J_k is scheduled; then, in step 3., $J_j \in F$ (since $d_j > e_k$) and $t_F \leq r_j \leq t_1 = e_k$.

(b) $J_j \notin U$ when J_k is scheduled, but $J_j \in U$ at the end of the schedule. This means that J_j becomes redundant later, after we assign some job, J_i, *which is currently in* U. Consider the three events of (i) scheduling J_j, (ii) scheduling J_k, and (iii) scheduling J_i and returning J_j to U. Note that the events must occur in this order, since $J_j \notin U$ when J_k is scheduled, and $J_j \in U$ after J_i is scheduled. Since the algorithm proceeds from left to right in the interval $[0, T]$, and the order of the events is $(i), (ii), (iii)$, we have $e_j < e_k < e_i$. Recall that J_j is scheduled in $[s_j, e_j]$ and that once J_i is scheduled J_j becomes redundant. Therefore, $s_i \leq e_j$ which implies the $r_i \leq e_j < e_k$. In addition, since $e_k < e_i$, it follows that $e_k < d_i$. Thus, in the iteration in which J_k is scheduled, $J_i \in F$ and $t_F \leq r_i < e_k$.

We conclude that, in both cases, in step 3. of the iteration in which J_k is scheduled, $\max(e_k, t_F) = e_k = t_1$, therefore we set $t = e_k$. Also, $F \neq \emptyset$, since it contains at least J_j or J_i. Thus, the algorithm proceeds to step 2. with $t = t_1$. By Property 4.3, an interval $[t_1, t_1 + \varepsilon]$ will be covered in the next iteration. By Property 4.2, this interval remains covered until the end of the schedule, in contradiction to the assumption that t_1 is the leftmost non-covered point in $[r_j, d_j]$. ■

Theorem 4.5 Greedy *yields a $(2 - \varepsilon)$-approximation for continuous RSO.*

Proof: Let $S = I \setminus U$ denote the set of jobs scheduled by Greedy, and let $O \subseteq S$ denote the set of scheduled job such that $J_j \in O$ iff J_j overlaps with another scheduled job, J_k, and $e_j > e_k$.

(i) By Property 4.4, for any $J_j \in U$, the machine is utilized in the time interval $[r_j, d_j]$.

(ii) For any $J_j \in S$, if $s_j > r_j$ then the machine is utilized in the time interval $[r_j, s_j]$; otherwise, Greedy would have scheduled it earlier.

(iii) For any $J_j \in O$, the machine is utilized in the time interval $[r_j, d_j]$. This follows from (ii) and from the fact that $e_j = d_j$ (otherwise, J_j would not overlap with a job that completes earlier).

Given the schedule of Greedy, we allow OPT to add jobs in U and to shift the jobs in S in any way that increases the utilization. Consider the three *disjoint* sets of jobs, $U, O, S \setminus O$. By the above discussion, the utilization can be increased only by shifting to the left (i.e., scheduling earlier) the jobs in $S \setminus O$. Note that, at any time $t \in [0, T]$, at most one job $J_k \in S \setminus O$ is processed (if two or more jobs overlap then only the one with the earliest completion time is in $S \setminus O$). Assume that J_k overlaps in the Greedy schedule with another job for a time interval of length $(1 - \varepsilon_k)p_k$, for some $0 < \varepsilon_k \leq 1$. Then OPT may start processing J_k earlier, such that it does not overlap with any other job. Hence, OPT can increase the amount of time the machine is utilized in the greedy schedule at most by factor of $(2 - \varepsilon)$, for some $\varepsilon > 0$. ■

Analysis for discrete inputs For instances of $JISP_k$, we can apply Greedy with the following modification. Instead of considering the single release time and due date of a job, J_j, we consider the next interval, $I_{j,\ell}$, that includes or starts after time t. It is easy to verify that Properties 4.2-4.4 still hold (In Property 4.4, the machine is utilized in $\cup_\ell I_{j,\ell}$, for any $J_j \in U$). However, the proof of Theorem 4.5 is no longer valid. It may be the case that for a jobs J_j that is scheduled with overlaps, there is an interval, $I_{j,\ell}$ (starting later than the interval selected for processing J_j), in which the machine is idle. This means that OPT can gain by using this 'idle' interval for processing J_j. Since redundant jobs are removed by Greedy, we can only have two overlapping jobs at any point of time. By selecting alternative processing intervals for each such pair of jobs, and adding to the schedule another, non-scheduled, job, OPT can now triple the total machine utilization. Thus, we have shown

Theorem 4.6 Greedy *yields a* $(3 - \varepsilon)$*-approximation for discrete RSO.*

4.2 The *BRSO* Problem

As *BRSO* generalizes the *RSO* problem, Theorem 4.1 implies that it is strongly NP-hard. For budgeted $JISP_k$ with overlaps allowed (i.e., discrete inputs) we give an APX-hardness proof. A similar proof can be used for the case where overlaps are not allowed (*BRS*).

Theorem 4.7 *The discrete BRSO is APX-hard, already for instances where* $\forall j, n_j \leq k$ *(*$JISP_k$*), for any* $k \geq 3$.

Proof: We use an L-reduction from the maximum 3-bounded 3-dimensional matching problem (3DM-3), defined as follows.

Input: A set of triplets $T \subseteq X \times Y \times Z$, where $|X| = |Y| = |Z| = n$; the number of occurrences of any item of $X \cup Y \cup Z$ in T is at most 3. The number of triplets is $|T| \geq n$.

Output: A 3-dimensional matching in T of maximal cardinality, i.e., a subset $T' \subseteq T$, such that any item in X, Y, Z appears at most once in T', and $|T'|$ is maximal.

Kann showed in [K-91] that 3DM-3 is APX-hard, that is, there exists $\varepsilon_0 > 0$ such that it is NP-hard to decide whether an instance has a matching of size n, or if every matching has size at most $(1-\varepsilon_0)n$.

Given an instance of 3DM-3, construct an instance I for $BRSO$ in which each job can be scheduled in at most 3 possible intervals: Let t be the cardinality of T. Denote by $e_1, \ldots, e_t$ the triplets in T. All the intervals will be contained in $[0, 3t]$.

There are $3n + t$ jobs. The first $3n$ jobs represent the items in X, Y, Z. The last t jobs (denoted T-jobs) represent the triplets in T.

- The parameters of the X jobs are: $p_i = 1, c_i = 0$ and the intervals $[3h, 3h+1]$ for any h such that $x_i \in e_h$ (at most three), $\forall i = 1, .., n$.
- The parameters of the Y jobs are: $p_j = 1, c_j = 0$ and the intervals $[3h+1, 3h+2]$ for any h such that $y_j \in e_h$ (at most three), $\forall j = 1, .., n$.
- The parameters of the Z jobs are: $p_k = 1, c_k = 0$ and the intervals $[3h+2, 3h+3]$ for any h such that $z_k \in e_h$ (at most three), $\forall k = 1, .., n$.
- The parameters of the T-jobs are: $p_\ell = 3, c_\ell = 1$ and the single interval $[3\ell, 3(\ell+1)]$, $\forall \ell = 0, .., t-1$.

The budget is $B = t - n$; thus, all the first $3n$ jobs and additional $t - n$ (out of the t) T-jobs can be scheduled. The idea is to divide the time interval $[0, 3t]$ into t intervals of length 3 each. Each such 3-interval represents one triplet in T. In the 3-interval $[3\ell, 3(\ell+1)]$ jobs from X, Y, Z that belong to $e_{\ell+1}$, or one T-job, are scheduled.

If there is a matching of size n, then the utilization can be $3t$: n 3-intervals are covered by the jobs that form the matching, and $(t-n)$ 3-intervals are covered by T-jobs. If there is a matching of size $(n - n_1)$ then the maximal utilization is at most $(3t - n_1)$: Note that at most $(n - n_1)$ 3-intervals are covered by jobs that belong to the matching; the budget allows covering of additional $(t - n)$ 3-intervals with T-jobs. Each of the n_1 remaining 3-intervals can be utilized in at most two slots (by X-,Y-, or Z-jobs). Other X-,Y-, or Z-job must overlap with T-jobs and do not contribute to the utilization. Thus, the maximal machine utilization is $3(n - n_1) + 3(t - n) + 2n_1 = 3t - n_1$.

The APX-hardness follows from the fact that $t = O(n)$ for 3-bounded instances, so an ε_0-fraction of n is an ε_1-fraction of $3t$. Taking $n_1 = 3\varepsilon_1 t$, we get that it is NP-hard to decide whether an instance has utilization $3t$, or if every schedule has utilization at most $3t(1-\varepsilon_1)$. ∎

4.2.1 A Primal-Dual Algorithm

We first present a primal-dual algorithm for RSO, (with unlimited budget), and show that an execution of the Greedy algorithm given in Section 4.1.2 can be equivalently interpreted

as an execution of the primal-dual algorithm. Thus, the primal-dual algorithm finds a 3-approximate solution to RSO. This allows us to apply the Lagrangian relaxation technique (presented in Section 2) to achieve a $(3+\varepsilon)$-approximation for the $BRSO$ problem. The primal LP is equivalent to $L-RSO$, given in Section 4.1.1. There is a variable $x(j,\ell)$ for each interval $I_{j,\ell}$, and a variable z_i, $i=1,\ldots,m$ for each interval $[b_{i-1},b_i]$ defined by consecutive breakpoints. In the dual LP there is a variable y_j for each job J_j, and two variables, p_i and q_i for each interval $[b_{i-1},b_i]$ defined by consecutive breakpoints.

$$(L-RSO-Primal) \quad \text{maximize} \quad \sum_{i=1}^{m} z_i \cdot (b_i - b_{i-1})$$

subject to:

$$\begin{aligned}
\text{For all jobs } J_j: &\quad \sum_{\ell=1}^{n_j} x(j,\ell) \le 1 \\
\text{For all } i=1,\ldots,m: &\quad z_i \le 1 \\
\text{For all } i=1,\ldots,m: &\quad z_i - \sum_{I_{j,\ell} \ni [b_{i-1},b_i]} x(j,\ell) \le 0 \\
\text{For all } j,\ell,i: &\quad x(j,\ell), z_i \ge 0.
\end{aligned}$$

$$(L-RSO-Dual) \quad \text{minimize} \quad \sum_{j=1}^{n} y_j + \sum_{i=1}^{m} p_i$$

subject to:

$$\begin{aligned}
\text{For all } (j,\ell), \ell=1,\ldots,n_j: &\quad y_j - \sum_{I_{j,\ell} \ni [b_{i-1},b_i]} q_i \ge 0 \\
\text{For all } i=1,\ldots,m: &\quad p_i + q_i \ge (b_i - b_{i-1}) \\
\text{For all } j,i: &\quad y_j, p_i, q_i \ge 0.
\end{aligned}$$

Given an integral solution for $L-RSO-Primal$, we say that an interval I belongs to it if there is a job that is assigned to I. An integral solution for $L-RSO-Primal$ is *maximal*, if it cannot be extended and if no interval belonging to it is contained in the union of other intervals belonging to it.

Lemma 4.8 *Any maximal integral solution (x,z) to $L-RSO-Primal$ is a 3-approximate solution.*

Proof: If $[b_{i-1},b_i]$ is covered by (x,z), then set $p_i = b_i - b_{i-1}$, otherwise set $q_i = b_i - b_{i-1}$. Clearly, this defines a feasible dual solution in which $\sum_{i=1}^{m} p_i = \sum_{i=1}^{m} z_i \cdot (b_i - b_{i-1})$. Thus, it remains to bound $\sum_{j=1}^{n} y_j$ in this solution.

For each job J_j that is not assigned to any interval in (x,z), i.e., its intervals are contained in intervals of other jobs, we can set $y_j = 0$. Suppose that for job J_j, $x(j,\ell) = 1$.

Consider, for example, an interval $I_{j,\ell'}$, $\ell \neq \ell'$, that contains two consecutive breakpoints b_{i-1} and b_i such that $[b_{i-1}, b_i]$ is not covered by any job. In this case $q_i = b_i - b_{i-1}$ and $y_j \geq q_i$. Thus, in order to bound $\sum_{j=1}^{n} y_j$, we say that the values of q_i-s that determine the y_j-s "charge" the p_i values corresponding to the breakpoints covered by $I_{j,\ell}$. This can be done since all the intervals in which J_j can be scheduled have the same length. Since our primal solution is maximal, any point is covered by at most two intervals to which jobs are assigned, and therefore any variable p_i can be "charged" by intervals belonging to at most two different jobs. Thus, $\sum_j y_j \leq 2 \sum_i p_i$, proving that

$$\sum_{i=1}^{m} p_i \;=\; \sum_{i=1}^{m} z_i \cdot (b_i - b_{i-1}) \;\leq\; \sum_{j=1}^{n} y_j + \sum_{i=1}^{m} p_i \;\leq\; 3 \cdot \sum_{i=1}^{m} p_i,$$

meaning that (x, z) is a 3-approximate solution. ∎

Continuous Input For continuous instances, time can be discretized such that each time slot is of size ϵ. This will incur a $(1 + \epsilon')$ degradation in the objective function where $\epsilon' = \text{poly}(\epsilon, n)$. We can show that for a discrete input obtained from a discretization of a continuous input instance, the primal-dual algorithm yields a 2-approximate solution. Applying the Lagrange relaxation technique (presented in Section 2), for the budgeted problem we get the following.

Theorem 4.9 *BRSO can be approximated within factor* $(2 + \varepsilon)$ *in the continuous case, and* $(3 + \varepsilon)$ *in the discrete case, in* $O((n \log(n/\varepsilon))^2)$ *steps.*

4.3 An FPTAS for $JISP_1$

For instances of *BRSO* where $p_j = d_j - r_j$ ($JISP_1$), we use a reduction to the *budgeted longest path problem* in acyclic graph, to obtain an optimal polynomial time algorithm for unit costs, and an FPTAS for general instances. In the budgeted longest path problem, we are given an acyclic graph, $G(V, E)$; each edge $e \in E$ has a length $\ell(e)$, and a cost $c(e)$. Our goal is to find the longest path in G connecting two given vertices s, t, whose price is bounded by a given budget B. The problem is polynomially solvable for unit edge costs, and has an FPTAS for arbitrary costs [Ha-92].

Given an instance of *BRSO* where $\forall j,\ p_j = d_j - r_j$, construct the following graph, G. Each job j is represented by a vertex; there is an edge $e = (i, j)$ iff $d_i < d_j$ and $r_i \leq r_j$. The length of the edge is $\ell(e) = \min(d_j - d_i, p_j)$, and its cost is c_j. Note that $\ell(e)$ reflects the machine utilization gained if the deadlines of J_i, J_j are adjacent to each other in the schedule. In addition, each vertex j is connected to the source, s, where $\ell(s, j) = p_j$, $c(s, j) = c_j$, and to a sink t, where $\ell(j, t) = 0$ and $c(s, j) = 0$.

Theorem 4.10 *There is a schedule achieving utilization of* u *time units and having cost* $b \leq B$ *if and only if* G *contains a path of length* u *and price* b.

Proof: For a given schedule, sort the jobs in the schedule such that $d_{j_1} \leq d_{j_2} \leq \ldots \leq d_{j_w}$. W.l.o.g, the schedule does not include two jobs J_i, J_j such that $r_j < r_i$ and $d_i < d_j$, since in such a schedule we gain nothing from processing J_i. Thus, we can assume that $r_{j_i} \leq r_{j_{i+1}}$, $\forall 1 \leq i \leq w$. This implies that the graph G contains the path $s, j_1, j_2, \ldots, j_w, t$ (the first and last edges in this path exist by the structure of G). Suppose that the utilization of the schedule is u and its cost is b. We show that the length of the corresponding path in G is u and its cost is b. Recall that the edge (j_w, t) has length 0 and costs nothing, thus, we consider only the first w edges in this path. The utilization we gain from scheduling j_i is p_{j_i} if $i = 1$ and $\min(p_{j_i}, d_{j_i} - d_{j_{i-1}})$ if $1 < i \leq w$. This is exactly $\ell(j_{i-1}, j_i)$ (or $\ell(s, j_1)$ for the first vertex in the path). Also, the cost of the schedule is the total processing cost of the scheduled jobs, which is identical to the total cost of edges in the path.

For a given directed path in G, we schedule all the jobs whose corresponding vertices appear on the path. Note that the price of the path consists of the price of the participating vertices, thus, b is also the price of the schedule. Also, as discussed above $\ell(i, j)$ reflects the contribution of the corresponding job to the utilization, thus the path induces a schedule with the correct utilization and cost. ∎

5 Multiple Machines

Suppose that there are m identical machines, and a budget B, which can be distributed in any way among the machines. It can be shown that this model is equivalent to the single machine case, by concatenating the schedules on the m machines to a single schedule in the interval $[0, mT]$, on a single machine. Thus, all of our results carry over to this model.

When a budget is specified for each machine, we show that any approximation algorithm $\mathcal{A}$ for a single machine can be run iteratively on the machines and the remaining jobs, to obtain a similar approximation ratio. Denote this algorithm by $\mathcal{A}^*$.

Theorem 5.1 *If $\mathcal{A}$ is an r-approximation then the iterative algorithm $\mathcal{A}^*$ is an $(r+1)$-approximation.*

Proof: For a given optimal schedule, OPT, let S_i denotes the jobs in the i-th machine that *were not* scheduled by $\mathcal{A}^*$. Let A_i denote the jobs assigned to M_i by $\mathcal{A}^*$. Let $w(j), (w(S))$ denote the weight of a job j (A set S of jobs). First note that for all i, $w(A_i) \geq \frac{1}{r} w(S_i)$. This holds since Algorithm $\mathcal{A}$ is r-approximation and S_i was available to $\mathcal{A}$ when it considered M_i.

Summing over all the machines we get that $w(A^*) = \sum_i w(A_i) \geq \frac{1}{r} \sum_i w(S_i)$. There are two cases:

1. $\sum_i w(S_i) \geq \frac{r}{r+1} w(OPT)$, and then $w(A^*) \geq \frac{1}{r+1} w(OPT)$;

2. $\sum_i w(S_i) < \frac{r}{r+1} w(OPT)$, and then, by the definition of S_i, all the other jobs (whose weight is at least $\frac{1}{r+1}$ of OPT) are scheduled by $\mathcal{A}^*$.

■

Note that in most cases $\mathcal{A}^*$ performs better. For example, when we iterate Greedy (Section 4.1.2) for the *RSO* problem, it can be seen that the proof for a single machine is valid also for multiple machines, thus Greedy is a $(2-\varepsilon)$-approximation algorithm.

Our results can also be extended to apply for the case where the processing costs of the jobs are machine dependent, that is, the cost of processing J_j on the k-th machine is c_{jk}, $1 \leq j \leq n$, $1 \leq k \leq m$.

Acknowledgments

We thank Shmuel Zaks for encouraging us to work on RSO and its variants. We thank Magnús Halldórsson and Baruch Schieber for valuable discussions. We thank David Amzallag for providing us with the budgeted maximum weight matching example.

References

[AS-87] E.M. Arkin and E.B. Sliverberg. "Scheduling jobs with fixed start and end times". *Discrete Applied Math.*, 18:1–8, 1987.

[B-99] P. Baptiste. Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times. *J. of Scheduling*, 2:245–252, 1999.

[BB+01] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48:1069-1090, 2001.

[BG+99] A. Bar-Noy, S. Guha, J. Naor and B. Schieber Approximating the throughput of real-time multiple machine scheduling. *SIAM Journal on Computing*, 31:331-352, 2001.

[BD-00] P. Berman and B. DasGupta. "Multi-phase algorithms for throughput maximization for real-time scheduling." *J. of Combinatorial Optimization*, 4:307–323, 2000.

[COR-01] J. Chuzhoy, R. Ostrovsky, and Y. Rabani. "Approximation algorithms for the job interval selection problem and related scheduling problems." In proc. of *FOCS*, 2001.

[GJ-79] M.R. Garey and D.S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness.* W.H. Freeman, 1979.

[Ga-96] N. Garg, A 3-approximation for the minimum tree spanning k vertices, Proceedings of FOCS, 1996.

[Ha-92] R. Hassin, Approximation schemes for the restricted shortest path problem. In *Mathematics of Operations Research*, 17(1): 36–42, 1992.

[K-91] V. Kann. Maximum bounded 3-dimensional matching is max SNP-complete. *Information Processing Letters*, 37:27–35, 1991.

[KN-01] Y. Karuno and H. Nagamochi, 2-approximation algorithms for the multi-vehicle scheduling on a path with release and handling times. *Discrete Applied Mathematics*, 129: 433-447, 2003.

[KMN-99] S. Khuller, A. Moss, J. Naor, The budgeted maximum coverage problem. *Information Processing Letters* 70(1): 39-45, 1999.

[L-05] A. Levin, Real time scheduling with a budget: Parametric-search is better than binary search. Manuscript, 2005.

[LO-02] D. H. Lorenz and A. Orda, Optimal partition of QoS requirements on unicast paths and multicast trees, IEEE/ACM Transactions on Networking, Vol. 10, No. 1, pp. 102-114, 2002.

[LORS-00] D. H. Lorenz, A. Orda, D. Raz, Y. Shavitt, Efficient QoS partition and routing of unicast and multicast, *Eighth International Workshop on Quality of Service (IWQoS 2000)*, Pittsburgh, June, 2000.

[S-99] F. C. R. Spieksma. "On the approximability of an interval scheduling problem." *Journal of Scheduling,* **2**:215–227, 1999.

[T-00] M. Tennenholtz. Some tractable combinatorial auctions. *AAAI/IAAI*, 98-103, 2000.

[Va-01] V. Vazirani, Approximation algorithms, Springer Verlag, 2001.