Techniques for Scheduling with Rejection

Daniel W. Engels* David R. Karger† Stavros G. Kolliopoulos‡
Sudipta Sengupta† R. N. Uma§ Joel Wein¶

*dragon@glenfiddich.lcs.mit.edu. Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139. Research supported by NSF Contract MIP-9612632.

†{karger, sudipta}@theory.lcs.mit.edu. Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139. Research supported in part by ARPA contract N00014-95-1-1246 and NSF contract CCR-9624239, as well as grants from the Alfred P. Sloane and David and Lucille Packard foundations.

‡stavros@mcmaster.ca. Department of Computing and Software, Faculty of Engineering, McMaster University, 1280 Main Street West, Hamilton, Ontario L8S 4L7, Canada. This research was done while the author was a graduate student at the Department of Computer Science, Dartmouth College, Hanover, NH 03755-3510. Research partially supported by NSF Award CCR-9308701 and NSF Career Award CCR-9624828.

§rnuma@utdallas.edu. Department of Computer Science, University of Texas at Dallas, M/S EC 31, Richardson, TX 75083-0688. This research was done while the author was a graduate student at the Department of Computer Science, Polytechnic University, Brooklyn, NY, 11201. Research partially supported by NSF Grant CCR-9626831.

¶wein@mem.poly.edu. Department of Computer Science, Polytechnic University, Brooklyn, NY, 11201. Research partially supported by NSF Grant CCR-9626831 and a grant from the New York State Science and Technology Foundation, through its Center for Advanced Technology in Telecommunications.

Running head : Techniques for Scheduling with Rejection

Contact Author : R. N. Uma
Department of Computer Science
University of Texas at Dallas
M/S EC 31
Richardson, TX 75083-0688

(rnuma@utdallas.edu)

Abstract

We consider the general problem of scheduling a set of jobs where we may choose not to schedule certain jobs, and thereby incur a penalty for each rejected job. More specifically, we focus on choosing a set of jobs to reject and constructing a schedule for the remaining jobs so as to optimize the sum of the weighted completion times of the jobs scheduled plus the sum of the penalties of the jobs rejected.

We give several techniques for designing scheduling algorithms under this criterion. Many of these techniques show how to reduce a problem with rejection to a (potentially more complex) scheduling problem without rejection. Some of the reductions are based on general properties of certain kinds of linear-programming relaxations of optimization problems, and therefore are applicable to problems outside of scheduling; we demonstrate this by giving an approximation algorithm for a variant of the facility-location problem.

1 Introduction

Most of traditional scheduling theory begins with a set of n jobs to be scheduled in a particular machine environment so as to optimize a particular optimality criterion. At times, however, a higher-level decision has to be made: given a set of tasks, and limited available capacity, choose only a subset of these tasks to be scheduled, while perhaps incurring some penalty for the jobs that are not scheduled. Knapsack is a "pure" instance of the problem, where we care only about what subset to "accept." In scheduling, however, the set of accepted jobs must also be scheduled at some cost.

In this paper we contribute to the understanding of this problem by studying several scheduling models in which the scheduler can choose to *reject* (not schedule) certain jobs at a penalty. We give several techniques for designing exact and approximation algorithms in this paradigm.

1.1 Problem Definition

For much of this paper we study variants of the following basic scheduling problem. We are given n jobs $j = 1, \ldots, n$, each with a nonnegative processing time p_j, a weight w_j, and a *rejection penalty* e_j. For each job we must decide either to schedule that job (on a machine that can process at most one job at a time) or to reject it. If we schedule job j, we denote its completion time by C_j. If we reject job j, we pay its rejection penalty e_j. Our goal is to choose a subset S of the n jobs to schedule on the machine so as to minimize the sum of the weighted completion times of the scheduled jobs and the penalties of the rejected jobs. We denote the set of rejected jobs by $\overline{S}$, and thus our overall optimality criterion may be denoted as $\sum_{j \in S} w_j C_j + \sum_{j \in \overline{S}} e_j$.

At times we consider scheduling models in which the processing of the jobs is constrained by either *release date* constraints (job j cannot begin processing before a specified release date r_j) or *precedence* constraints ($j \prec k$ is a precedence constraint stating that job k can begin its processing no earlier than job j completes its processing). Precedence constraints impose a partial ordering on the jobs. We also consider models with more than one machine available for processing and other related optimality criteria; we leave their details to be introduced as appropriate in the body of the paper.

Our interest in these models arises primarily from two considerations. First, it is becoming widely understood that in actual manufacturing settings a scheduling algorithm is only one element of a larger decision analysis tool that takes into account a variety of factors such as inventory, potential machine disruptions, etc. [17]. Furthermore, some practitioners [15] have identified as a critical element of this larger decision picture the "pre-scheduling negotiation" in which one considers the capacity of your production environment and then agrees to schedule certain jobs at the requested quality of service (e.g. job turnaround time) and other jobs at a reduced quality of service, while rejecting other jobs altogether.

Typically, the way that current systems handle the integration of objective functions is to have a separate component for each problem element and to integrate these components in an ad-hoc heuristic fashion. Therefore, models that integrate more than one element of this decision process while remaining amenable to solution by algorithms with performance guarantees have the potential to be very useful elements in this decision process.

Secondly, our work is directly inspired by that of Bartal, Leonardi, Marchetti-Spaccamela, Sgall and Stougie [1] and Seiden [22] who studied a multiprocessor scheduling problem with the objective of trading off between schedule *makespan* (length) and job rejection cost. Makespan and average weighted completion time are perhaps the two most basic and well-studied of all scheduling optimality criteria; therefore, it is of interest to understand the impact of the "rejection option" on the $\sum w_j C_j$ objective as well.

Since we will be dealing with a number of scheduling models, we will use the scheduling notation introduced by Graham, Lawler, Lenstra and Rinnooy Kan [8] to denote the model being considered.

1.2 Relevant Previous Work

There has been much work on scheduling without rejection to minimize $\sum w_j C_j$. For the simplest variant of the problem, $1||\sum w_j C_j$ (scheduling jobs with no side constraints on one machine to minimize $\sum w_j C_j$) Smith [24] proved that an optimal schedule could be constructed by ordering the jobs by non-decreasing p_j/w_j ratios. More complex variants are typically $\mathcal{NP}$-hard, and recently there has been a great deal of work on the development of approximation algorithms for them (e.g. [18, 10, 7, 2, 21, 20]). A *ρ-approximation algorithm* is a polynomial-time algorithm that always finds a solution of objective function value within a factor of ρ of optimal (ρ is also referred to as the *performance guarantee*).

To the best of our knowledge the only previous work on scheduling models that include rejection in our manner is that of [1, 22]. Bartal et. al. seem to have formally introduced the notion. They considered the problem of scheduling with rejection in a multiprocessor setting with the aim of minimizing the makespan of the scheduled jobs and the sum of the penalties of the rejected jobs [1]. They give a $(1+\phi)(\approx 2.618)$-competitive algorithm for the on-line version, where ϕ is the golden ratio. Seiden [22] gives a 2.388-competitive algorithm for the above on-line problem when preemption is allowed.

1.3 Our Results

Our work differs from that of Bartal et. al. in that we focus on $\sum w_j C_j$ rather than the makespan. More importantly, we focus less on a particular problem and more on giving general techniques that are applicable to a number of problems. We address only the off-line setting (the instance is fully specified in advance).

We begin in Section 2 by studying the simplest version of the problem, $1||\sum_S w_j C_j + \sum_{\overline{S}} e_j$. We show that the problem $1||\sum_S w_j C_j + \sum_{\overline{S}} e_j$ is weakly $\mathcal{NP}$-Complete, and give a pseudopolynomial-time algorithm based on dynamic programming that solves it exactly. Our dynamic program can be interpreted as capturing the option of assigning a job to a "rejection machine". In addition, we modify the dynamic program to get an FPTAS. We conclude Section 2 by giving a direct reduction (using the idea of a "rejection machine") of certain scheduling problems with rejection to more complex problems without.

In Sections 3 and 4 we give another example of how techniques that are used for scheduling problems without rejection can be adapted to handle problem variants with rejection. In contrast to a direct problem-to-problem reduction, however, we examine the structure of certain linear-programming relaxations of such scheduling problems, show how they can be augmented to include a rejection variable to which a job can be assigned, and then give a thresholding technique that allows us to obtain an approximation algorithm for a problem with rejection. This technique is in fact quite general and is applicable to many sorts of optimization problems; we demonstrate its generality by giving an approximation algorithm for a variant of the facility location problem with rejection.

2 Scheduling with Rejection

Consider the problem of scheduling a set of independent jobs with the flexibility of rejecting some of the jobs, i.e., excluding some of the jobs from the scheduled set, so as to minimize the sums of the weighted completion times of the scheduled jobs and the rejection penalties for the excluded jobs. We denote this problem as $1||(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ where each job is characterized by a processing requirement, p_j, a weight w_j, and a rejection penalty e_j. The set of scheduled jobs is denoted by S, and the completion time for each job is denoted by C_j. The set of rejected jobs is denoted by $\bar{S} = N - S$, where N is the original set of jobs.

In this section, we focus on the problem of scheduling jobs on a single machine to minimize weighted completion time. If rejection is not considered, the problem is solvable in polynomial time using Smith's rule: schedule the jobs in increasing order of p_j/w_j. We show that adding the option of rejection makes the problem weakly $\mathcal{NP}$-complete. Such problems cannot admit polynomial-time solutions unless $\mathcal{P} = \mathcal{NP}$. But they can admit *pseudo-polynomial* time algorithms that run fast when the input numbers are small. We give such an algorithm, based on dynamic programming, for our scheduling problem.

Intuitively, the dynamic program can be seen as solving a scheduling problem on two machines, one a "rejection machine" that takes care of all of the rejected jobs and has an unusual cost function for doing so. It is therefore a variant of Rothkopf [19] and Lawler and Moore's [13] pseudo-polynomial algorithms for scheduling to minimize weighted completion time on a fixed number of parallel machines. Our algorithm also generalizes to schedule with rejection on any fixed number of parallel machines. We use the "rejection machine" intuition again to devise a 3/2-approximation algorithm for the problem of scheduling with rejection on an arbitrary number of machines.

Although we use the two-machine intuition, the rejection cost that the second machine pays is in some sense a simpler function than the weighted completion time. So for certain special cases, we are able to give simple and efficient polynomial time algorithms based on greedy methods.

2.1 Complexity of Scheduling with Rejection

For any fixed number of processors $m > 1$, $Pm| \, |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ is trivially seen to be $\mathcal{NP}$-complete by restricting the problem to $Pm| \, |\sum w_j C_j$, a known weakly $\mathcal{NP}$-complete problem [6]. However, we can also prove that adding rejection makes the *single* machine problem $\mathcal{NP}$-complete. The decision problem formulation of $1| \, |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ is defined as follows:

> Given a set of n independent jobs, $N = \{J_1, \dots, J_n\}$, with processing times p_j, $\forall\, 1 \le j \le n$, weights w_j, $\forall\, 1 \le j \le n$, and rejection penalties e_j, $\forall\, 1 \le j \le n$, a single processor, and a number K, is there a schedule of a subset of jobs $S \subseteq N$ on the processor such that $\sum_{j \in S} w_j C_j + \sum_{j \in \bar{S} = N - S} e_j \le K$?

We reduce the Partition Problem [6] to this problem proving that even on one processor, scheduling with rejection is $\mathcal{NP}$-complete.

Theorem 2.1 *The decision problem formulation of* $1| \, |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ *is weakly* $\mathcal{NP}$-*complete.*

Proof. $1| \, |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ is clearly in $\mathcal{NP}$. To prove that it is also $\mathcal{NP}$-hard, we reduce the Partition Problem [6] to $1| \, |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ in a straightforward manner. The Partition Problem is defined as follows:

> Given a set $A = \{a_1, a_2, \dots, a_n\}$ of n numbers such that $\sum_{i \in A} a_i = 2b$, is there a subset of A, A', such that $\sum_{i \in A'} a_i = b$?

Given an instance $A = \{a_1, \dots, a_n\}$ of the partition problem, we create an instance of $1| \, |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ with n jobs, $J_1, \dots, J_n$. Each of the n elements a_i in the Partition Problem corresponds to a job J_i in $1| \, |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ with weight and processing time equal to a_i and rejection cost equal to $ba_i + \frac{1}{2}a_i^2$, where $b = \frac{1}{2}\sum_{i=1}^n a_i$. Since $p_i = w_i, \forall\, 1 \le i \le n$, the ordering of the scheduled jobs does not affect the value of $\sum_{i \in S} w_i C_i$. Using this fact and substituting for the rejection penalty, we can rewrite the objective function as follows:

$$
\begin{aligned}
\sum_{i \in S} w_i C_i + \sum_{i \in \bar{S}} e_i &= \sum_{i \in S} a_i \sum_{(j \le i, j \in S)} a_j + \sum_{i \in \bar{S}} e_i \\
&= \sum_{i \in S} a_i^2 + \sum_{(i<j, i,j \in S)} a_i a_j + \sum_{i \in \bar{S}} (ba_i + \frac{1}{2}a_i^2) \\
&= \frac{1}{2}[(\sum_{i \in S} a_i)^2 + \sum_{i \in S} {a_i}^2] + b\sum_{i \in \bar{S}} a_i + \frac{1}{2}\sum_{i \in \bar{S}} {a_i}^2 \\
&= \frac{1}{2}(\sum_{i \in S} a_i)^2 + b\sum_{i \in \bar{S}} a_i + \frac{1}{2}\sum_{i=1}^{n} {a_i}^2
\end{aligned}
$$

Since $\sum_{i=1}^n a_i^2$ does not depend on the choice of S, this is equivalent to minimizing the following function with $x = \sum_{i \in S} a_i$

$$
\frac{1}{2}x^2 + b(2b - x) = \frac{1}{2}(\sum_{i \in S} a_i)^2 + b(2b - \sum_{i \in S} a_i)
$$

$$= \frac{1}{2}(\sum_{i \in S} a_i)^2 + b \sum_{i \in \bar{S}} a_i$$

This function has a unique minimum of $\frac{3}{2}b^2$ at $x = b$, i.e., when $\sum_{i \in S} a_i = b$. Hence if such an S exists the resulting value is optimal for the scheduling problem. Therefore, if the solution to $1||(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ has value less than or equal to $\frac{3}{2}b^2 + \frac{1}{2}\sum_{i=1}^n a_i^2$, then the answer to the Partition Problem is 'Yes'. Conversely, if the answer to the Partition Problem is 'Yes', there is an S with $\sum_{i \in S} a_i = b$; hence there is a schedule of value $\frac{3}{2}b^2 + \frac{1}{2}\sum_{i=1}^n a_i^2$. ■

2.2 Dynamic Programming

We give an $O(n \sum_{j=1}^n w_j)$ time algorithm using dynamic programming to solve $1||(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$. In Section 2.3, we will show how to modify it to obtain an FPTAS for $1||(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$.

To solve our problem, we set up a dynamic program for a harder problem: namely, to find the schedule that minimizes the objective function when the total weight of the scheduled jobs is given. We number the jobs in ascending order of p_j/w_j. Let $\phi_{w,j}$ denote the optimal value of the objective function when the jobs in consideration are $j, j+1, \ldots, n$, and the total weight of the scheduled jobs is W. Note that $\phi_{W,n} = e_n$ for any $W \neq w_n$ and $\phi_{w_n,n} = w_n p_n$, forming the boundary conditions for the dynamic program.

Now, consider any optimal schedule for the jobs $j, j+1, \ldots, n$ in which the total weight of the scheduled jobs is W. In any such schedule, there are two possible cases — either job j is rejected or job j is scheduled.

Case 1: Job j is rejected. Then, the optimal value of the objective function is clearly $\phi_{W,j+1} + e_j$, since the total weight of the scheduled jobs among $j+1, \ldots, n$ must be W.

Case 2: Job j is scheduled. In this case, the total weight of the scheduled jobs among $j+1, \ldots, n$ must be $W - w_j$. Also, when job j is scheduled before all jobs in the optimal schedule for jobs $j+1, j+2, \ldots, n$, the completion time of every scheduled job among $j+1, j+2, \ldots, n$ is increased by p_j. Then, the optimal value of the objective function is clearly $\phi_{W-w_j,j+1} + Wp_j$.

Combining the above two cases, we have:

$$\phi_{W,j} = \min(\phi_{W,j+1} + e_j, \phi_{W-w_j,j+1} + Wp_j)$$

Now, observe that the weight of the scheduled jobs can be at most $\sum_{j=1}^n w_j$, and the answer to our original problem is $\min\{\phi_{W,1} : 0 \leq W \leq \sum_{j=1}^n w_j\}$. Thus, we need to compute exactly $n \sum_{j=1}^n w_j$ values $\phi_{W,j}$. We remark that a similar dynamic program solves $1||(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ in polynomial time when the weights are arbitrary but the processing times are polynomially bounded.

Theorem 2.2 *Dynamic programming yields an $O(n \sum w_j)$-time (or an $O(n \sum p_j)$-time) algorithm for exactly solving $1| |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$.*

We remark that some special cases of the problem can be solved in polynomial time. For example, the case with identical weights (and the case with identical processing times) can be solved in $O(n^2)$ time.

2.3 Fully polynomial time approximation scheme

In this section, we describe a fully polynomial time approximation scheme (FPTAS) for $1| |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$. The algorithm runs in time polynomial in n, $\frac{1}{\epsilon}$, and the size (number of bits) of the integers involved.

We trim the state space of the dynamic program of section 2.2 by fusing states that are "close" to each other. This is a classical approach used to arrive at an FPTAS from a dynamic program [12]. A general technique that applies to a fairly broad class of dynamic programs was developed by Woeginger [26]. This fusion of "close" states is achieved by constraining the jobs in any schedule to finish at times of the form $\tau_i = (1 + \epsilon')^i$, for $i \geq 0$, and $\epsilon' > 0$. We call such schedules *ϵ'-aligned* schedules. Note that an ϵ'-aligned schedule may contain idle time.

We identify any given schedule Γ to its "closest" ϵ'-aligned schedule by sliding the scheduled time of each job (starting from the first scheduled job and proceeding in order) forward in time until its completion time coincides with the next time instant of the form τ_i. The job is then said to be *ϵ'-aligned.* Note that when we slide the scheduled time of job i, the scheduled times of later jobs also get shifted forward in time by the same amount as for job i. When the time comes to ϵ'-align job j, its completion time has already moved forward in time by an amount equal to the sum of the amounts moved by the completion times of the jobs scheduled earlier than itself. This identification of a schedule with its "closest" ϵ'-aligned schedule gives us the notion of "closeness" of two schedules i.e. two schedules are "close" if the same jobs are scheduled in both, and they have the same "closest" ϵ'-aligned schedule.

The following lemma establishes an upper bound on the increase in the optimal objective function value when we restrict our attention to ϵ'-aligned schedules only. Without any loss of generality, we can assume that the smallest processing time is at least 1. Otherwise, we can divide each processing time p_j and rejection penalty e_j by the smallest processing time $p < 1$. This increases the size of each processing time and rejection penalty by $\log \frac{1}{p} = -\log p$ bits, and hence the running time of the algorithm (see Theorem 2.5) by at most a polynomial additive factor of $\frac{n^2}{\epsilon} \log \frac{1}{p}$.

Lemma 2.3 *For $1| |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$, the optimal objective function value increases by a factor of at most $(1 + \epsilon')^n$ for any $\epsilon' > 0$, when we restrict our attention to ϵ'-aligned schedules only.*

Proof. We establish, by induction on i, that in any given schedule, the completion time

C_i of the i^{th} scheduled job increases by a factor of at most $(1+\epsilon')^i$ after the schedule is ϵ'-aligned. Note that if a job finishes at time $t \in (\tau_{i-1}, \tau_i]$ after all the jobs before it have been ϵ'-aligned, its completion time after being ϵ'-aligned is $\tau_i < (1+\epsilon')t$. Clearly, C_1 increases by a factor of at most $(1+\epsilon')$, since it is made to coincide with the right end-point of the interval $(\tau_{\ell-1}, \tau_\ell]$ it is initially contained in. Note that the smallest τ_i is $\tau_0 = 1$, so that we made use here of the assumption that the smallest processing time is at least 1.

Now, assume by induction, that C_i increases by a factor of at most $(1+\epsilon')^i$. When the turn comes to ϵ'-align the $(i+1)^{th}$ scheduled job, its completion time C_{i+1} has already increased by an amount equal to the amount moved by C_i, which is at most $[(1+\epsilon')^i - 1]C_i$. When it is ϵ'-aligned, C_{i+1} *further* increases by a factor of at most $(1+\epsilon')$. Thus, the final value of C_{i+1} is at most $(1+\epsilon')([(1+\epsilon')^i - 1]C_i + C_{i+1})$. Since $C_i < C_{i+1}$, this is at most $(1+\epsilon')^{i+1}C_{i+1}$.

Since the number of scheduled jobs is at most n, the result follows. ∎

Setting $\epsilon' = \frac{\epsilon}{2n}$ gives us a $(1+\epsilon)$-factor increase in the optimal objective function value, as stated in the next lemma.

Lemma 2.4 *For* $1||(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$*, the optimal objective function value increases by a factor of at most* $(1+\epsilon)$ *for any* $\epsilon > 0$*, when we restrict our attention to* $\frac{\epsilon}{2n}$*-aligned schedules only.*

Proof. We first establish the inequality $(1+\frac{x}{m})^m \leq 1 + 2x$, which holds for any $0 \leq x \leq 1$, and any real $m \geq 1$. The left-hand side of the inequality is a convex function in x, and the right-hand side is a linear function in x. Moreover, the inequality holds at $x = 0$ and $x = 1$. Hence, it holds for any $0 \leq x \leq 1$. Setting $x = \frac{\epsilon}{2}$ and $m = n$, we have

$$(1+\frac{\epsilon}{2n})^n \leq (1+\epsilon)$$

The lemma now follows from Lemma 2.3 and the above inequality. ∎

Let $\epsilon' = \frac{\epsilon}{2n}$. For our FPTAS, we set up a dynamic program for a harder problem: namely, to find the ϵ'-aligned schedule that minimizes the objective function when the completion time of the latest scheduled job is on or before τ_i, for a given i. We number the jobs in ascending order of p_j/w_j. Let $\phi_{i,j}$ denote the optimal value of the objective function when the jobs in consideration are $1, 2, \ldots, j$, and the latest scheduled job (if any) in an ϵ'-aligned schedule completes at time less than or equal to τ_i for $i \geq 0$. Note that

$$\phi_{i,1} = \begin{cases} \min(e_1, w_1\tau_j) & \text{if } p_1 \in (\tau_{j-1}, \tau_j] \text{ and } \tau_j \leq \tau_i \\ \infty & \text{otherwise} \end{cases} \tag{1}$$

for $i \geq 0$. This forms the boundary conditions for the dynamic program.

Now, consider any optimal schedule for the jobs $1, 2, \ldots, j, j+1$ in which the latest scheduled job completes by time τ_i. In any such schedule, there are two possible cases — either job $j+1$ is rejected or job $j+1$ is scheduled.

Case 1: Job $j+1$ is rejected. Then, the optimal value of the objective function is clearly $\phi_{i,j} + e_{j+1}$, since the last of the scheduled jobs among $1, 2, \ldots, j$ must finish at time less than or equal to τ_i.

Case 2: Job $j+1$ is scheduled. This is possible only if $p_{j+1} \leq \tau_i$. In this case, if there was a job scheduled before job $j+1$, it must have completed at time $\tau_{i'}$, where $(\tau_i - \tau_{i'}) \geq p_{j+1}$. Otherwise, all the jobs $1, 2, \ldots, j$ were rejected. Then, the optimal value of the objective function is clearly $\phi_{i',j} + w_{j+1}\tau_i$, where i' is the greatest value of i'' such that $\tau_i - \tau_{i''} \geq p_{j+1}$.

Combining the above two cases, we have:

$$\phi_{i,j+1} = \min(\phi_{i,j} + e_{j+1}, \phi_{i',j} + w_{j+1}\tau_i)$$

Now, observe that for finding an ϵ'-aligned schedule with the optimum objective function value, it is sufficient to assume that the completion time of the latest scheduled job is at most $(1+\epsilon')^n \sum_{j=1}^n p_j$. The answer to our original problem is $\phi_{L,n}$, where L is the smallest integer such that $\tau_L \geq (1+\epsilon')^n \sum_{j=1}^n p_j$. Thus, L is the smallest integer greater than or equal to $\frac{\log \sum_{j=1}^n p_j}{\log(1+\epsilon')} + n$, whence $L = O(\frac{n}{\epsilon} \log \sum_{j=1}^n p_j)$. Thus, we need to compute exactly $n(L+1)$ values $\phi_{i,j}$. Computation of each such value takes $O(1)$ time, so that the overall time for the dynamic program (FPTAS) is $O(nL) = O(\frac{n^2}{\epsilon} \log \sum_{j=1}^n p_j)$. This is polynomial in the input size, since we need $\sum_{j=1}^n \log p_j$ bits to represent the processing times. We also note that dividing each processing time and rejection penalty by the smallest processing time $p < 1$ increases the running time of the algorithm by at most a polynomial additive factor of $\frac{n^2}{\epsilon} \log \frac{1}{p}$.

Theorem 2.5 *Dynamic programming yields a $(1+\epsilon)$-factor approximation algorithm (FPTAS) for $1| |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$, which runs in time $O(\frac{n^2}{\epsilon} \log \sum_{j=1}^n p_j)$.*

2.4 Worst-Case Performance Guarantees

We now turn to several NP-hard variants of the problem and give small-constant-factor approximation algorithms; we do this by reducing the problem with rejection to a scheduling problem without rejection.

For simplicity we consider first the single machine problem $1| |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ which was studied extensively in the previous subsections. We will then show how to generalize our reduction to accommodate release dates and parallel machines. An instance I of $1| |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ can be reduced in an approximation-preserving manner to an instance I' of $R| | \sum w_j C_j$ with $n+1$ unrelated machines. Let the machines be indexed $1, 2, \ldots, (n+1)$. Machine $(n+1)$ is the *original* machine with $p_{n+1,j} = p_j$ for job j. Job j has $p_{jj} = \frac{e_j}{w_j}$ on machine j and $p_{ij} = \infty$ on machine i where $i = 1, \ldots, n$ and $i \neq j$. Let $S(I)$ be a schedule for I with objective value $\pi(I)$. We demonstrate a schedule $S'(I')$ for I' with objective value $\pi'(I') = \pi(I)$. A job j that is rejected in $S(I)$ is scheduled on machine j in $S(I')$ and has by our construction weighted completion time e_j. Non-rejected jobs in $S(I)$ are scheduled to run on machine $(n+1)$ in $S(I')$. Conversely if $S'(I')$ is a schedule for I' with objective value $\pi'(I')$ a schedule $S(I)$ for I can be constructed with objective $\pi(I) = \pi'(I')$.

A similar reduction can be obtained from $1|r_j|(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ to $R|r_j|\sum w_j C_j$. The main differences from the reduction above is that in instance I' we define p_{jj} to be $\frac{e_j}{w_j} - r_j$. to ensure that the weighted completion time of job j on machine j is again e_j. Observe that the instance I' is well defined since we can assume without loss of generality $e_j > w_j r_j$ for all j. Finally, it is easy to extend the reduction to the case of parallel unrelated machines. An instance I of the rejection problem with m machines and n jobs can be reduced to minimizing the average completion time of n jobs on $m+n$ unrelated machines. Using the approximation algorithms of Chudak [3] or Schulz and Skutella [20] for $R||\sum w_j C_j$ and of Schulz and Skutella for $R|r_{ij}|\sum w_j C_j$ [21] we obtain the following theorem.

Theorem 2.6 *There exists a $\frac{3}{2}$-approximation algorithm for $R||\sum_S w_j C_j + \sum_{\bar{S}} e_j$ and a 2-approximation algorithm for $R|r_j|\sum_S w_j C_j + \sum_{\bar{S}} e_j$.*

For problem variants with precedence constraints we know of no such simple reduction and thus "heavier artillery" is necessary. Approximation algorithms for such scheduling problems will be a consequence of the general technique to be presented in the next section.

3 A General Problem $\mathcal{P}$ with Rejection

In this section we give a rather general method of converting certain kinds of approximation algorithms for optimization problems without rejection to approximation algorithms for problem variants with rejection. In contrast to the last result in the previous section, however, which was a direct reduction to a scheduling problem in which the solution to that scheduling problem was used as a black box, in this case we exploit the common structure of different linear-programming relaxations used in approximation for scheduling and other optimization problems.

We begin by stating a general "assignment" optimization problem and gaining some understanding of the structure of solutions to two relaxations of this problem. Then we use this structure in a general algorithm for the optimization problem augmented to include the possibility of rejection.

Let us consider the following general problem $\mathcal{P}$. We have two sets of objects $\mathcal{O}_1$ and $\mathcal{O}_2$. We want to assign each object in $\mathcal{O}_1$ to an object in $\mathcal{O}_2$ such that a given set of constraints are satisfied by the assignment and some objective function of the assignment is minimized. We will formulate this problem as an integer program which uses the 0-1 decision variables x_{ij}, where x_{ij} is 1 if object $i \in \mathcal{O}_1$ is assigned to object $j \in \mathcal{O}_2$ and 0 otherwise. Fundamentally we want to optimize some cost function $\mathbf{c}^{\mathbf{T}}\mathbf{x}$ given that every object in $\mathcal{O}_1$ is assigned to one object in $\mathcal{O}_2$ and at most one object is assigned to any object in $\mathcal{O}_2$.

We will, however, present a more general integer program that captures more applications, including two specific ones that we address in later sections. To this end we augment our basic version of the IP to have variables y_ℓ, $0 \le \ell \le |\mathcal{O}_1| + |\mathcal{O}_2|$ where there is one variable for each object; this allows us to capture other costs and constraints that are not directly expressible in the assignment variables.

Furthermore, we will allow an object i in $\mathcal{O}_1$ to be assigned not just one time but at most s_i times, s_i being a positive integer. Finally, we will allow the assignment, to each object in $\mathcal{O}_2$, at most κ_j objects from $\mathcal{O}_1$, where κ_j is also a positive integer.

This more general problem $\mathcal{P}$ is formulated by the following integer program (IP). Let $\mathbf{c}, \mathbf{d} \ge \mathbf{0}$.

$$\text{minimize} \quad \mathbf{c^T x} + \mathbf{d^T y} \tag{2}$$

subject to

$$\begin{aligned}
\sum_j x_{ij} &= s_i && \forall i \in \mathcal{O}_1 && (3)\\
\sum_i x_{ij} &\le \kappa_j && \forall j \in \mathcal{O}_2 && (4)\\
\mathbf{A}(\mathbf{x} \circ \mathbf{y}) &\le \mathbf{b} && && (5)\\
x_{ij} &\in \{0,1\} && \forall i \in \mathcal{O}_1, \forall j \in \mathcal{O}_2 && (6)\\
x_{ij} &= 0 && (i,j) \in M \subseteq \mathcal{O}_1 \times \mathcal{O}_2 && (7)
\end{aligned}$$

Here, $\mathbf{x} \circ \mathbf{y}$ denotes the concatenation of vectors $\mathbf{x}, \mathbf{y}$. $\mathbf{A}(\mathbf{x} \circ \mathbf{y}) \le \mathbf{b}$ is a set of additional constraints that have to be satisfied by any feasible assignment. We call the constraints (4) *capacity constraints*. Finally, M captures the set of assignments that are illegal.

We can replace the integrality constraint on x_{ij} by the constraint $0 \le x_{ij} \le 1$. This essentially means that an object $i \in \mathcal{O}_1$ can be fractionally assigned to objects in $\mathcal{O}_2$. We will let $(LP1)$ denote the resulting relaxation to (IP).

$$\text{minimize} \quad \mathbf{c^T x} + \mathbf{d^T y}$$

subject to

$$\begin{aligned}
\sum_j x_{ij} &= s_i && \forall i \in \mathcal{O}_1\\
\sum_i x_{ij} &\le \kappa_j && \forall j \in \mathcal{O}_2\\
\mathbf{A}(\mathbf{x} \circ \mathbf{y}) &\le \mathbf{b} && \\
x_{ij} &\ge 0 && \forall i \in \mathcal{O}_1, \forall j \in \mathcal{O}_2\\
x_{ij} &= 0 && (i,j) \in M \subseteq \mathcal{O}_1 \times \mathcal{O}_2
\end{aligned}$$

We will now consider the following additional relaxation to $(LP1)$. Let β be a positive constant that is at most 1. We will further relax the constraint $\sum_j x_{ij} = s_i$ to the constraint

$\beta s_i \leq \sum_j x_{ij} \leq s_i$. This says that at least a β fraction of object $i \in \mathcal{O}_1$ must be assigned to some object(s) in $\mathcal{O}_2$. We will call this linear program $(LP2)$; it will be understood implicitly that this LP is parameterized by β.

$$\text{minimize} \quad \mathbf{c^T x} + \mathbf{d^T y}$$

subject to

$$\begin{array}{rcll}
\sum_j x_{ij} & \geq & \beta s_i & \forall i \in \mathcal{O}_1 \\
\sum_i x_{ij} & \leq & \kappa_j & \forall j \in \mathcal{O}_2 \\
\mathbf{A}(\mathbf{x} \circ \mathbf{y}) & \leq & \mathbf{b} & \\
x_{ij} & \geq & 0 & \forall i \in \mathcal{O}_1, \forall j \in \mathcal{O}_2 \\
x_{ij} & = & 0 & (i,j) \in M \subseteq \mathcal{O}_1 \times \mathcal{O}_2
\end{array}$$

Let us denote their corresponding optimal solutions by IP^*, $LP1^*$ and $LP2^*$ respectively. Since $(LP1)$ is a relaxation of (IP) and $(LP2)$ is a relaxation of $(LP1)$, we have the inequality $LP2^* \leq LP1^* \leq IP^*$.

Let $\mathcal{A}$ be an algorithm that accepts a feasible solution to $(LP1)$ as input and returns a solution to (IP) with value at most ρ times the input's objective function value. It thus is a ρ-approximation algorithm for (IP); we will call such an algorithm an approximation algorithm *with respect to* $(LP1)$. We will show that if problem $\mathcal{P}$ satisfies certain Conditions 3.1 and 3.2 (to be stated later), then we can use $\mathcal{A}$ to build an approximation algorithm (with somewhat worse performance) for a version of $\mathcal{P}$ that allows rejection. We now state Condition 3.1.

Condition 3.1 *There exists a polynomial-time algorithm to convert a feasible solution* $\mathbf{q}$ *to* $(LP2)$, *of value* $v(\mathbf{q})$, *to a feasible solution to* $(LP1)$ *of value within at most* $f(\beta) \cdot v(\mathbf{q})$, *for some function* f *of* β.

We now turn to the problem with rejection. Specifically, no longer does every object i in $\mathcal{O}_1$ need to be assigned s_i times to objects in $\mathcal{O}_2$; rather we can choose to not assign it at all and instead pay a penalty g_i in the objective function. We will call this modified problem $\mathcal{P}_R$.

The problem $\mathcal{P}_R$ can be formulated similar to (IP). But now we will have an additional 0-1 decision variable z_i which is 1 if object $i \in \mathcal{O}_1$ is rejected and 0 otherwise. The objective function to be minimized is now given by $\mathbf{c^T x} + \mathbf{d^T y} + \mathbf{g^T z}$ and the constraint $\sum_j x_{ij} = s_i$ in the (IP) is replaced by $\frac{1}{s_i} \sum_j x_{ij} + z_i = 1$. Let the corresponding linear relaxation be $(LP1_R)$.

$$\text{minimize} \quad \mathbf{c^T x} + \mathbf{d^T y} + \mathbf{g^T z}$$

subject to

$$\begin{aligned}
\frac{1}{s_i}\sum_j x_{ij} + z_i &= 1 && \forall i \in \mathcal{O}_1 \\
\sum_i x_{ij} &\le \kappa_j && \forall j \in \mathcal{O}_2 \\
\mathbf{A}(\mathbf{x}\circ\mathbf{y}) &\le \mathbf{b} && \\
x_{ij} &\ge 0 && \forall i \in \mathcal{O}_1, \forall j \in \mathcal{O}_2 \\
x_{ij} &= 0 && (i,j) \in M \subseteq \mathcal{O}_1 \times \mathcal{O}_2 \\
z_i &\ge 0 && \forall i \in \mathcal{O}_1
\end{aligned}$$

Our algorithmic technique can now be described as follows. We formulate and solve $(LP1_R)$. We then apply a simple thresholding rule by rejecting all objects in $\mathcal{O}_1$ for which $\sum_j x_{ij} < \beta s_i$; this increases the "rejection part" of the objective function by at most $\frac{1}{1-\beta}$ since at least $(1-\beta)$ of each of these objects was rejected fractionally. If we can guarantee that what remains constitutes a feasible solution to $(LP2)$ (and in many applications it does) and the problem satisfies Condition 3.1, we can then obtain an assignment of the remaining objects with cost at most $f(\beta)$ times the contribution of the fractional assignment to the objective function of $(LP1_R)$. The ability to apply this thresholding is captured in Condition 3.2.

We first define some terms that will be used in the statement of the condition. Let $S_\beta \subseteq \mathcal{O}_1$ be the set of objects in $\mathcal{O}_1$ that have been selected using the thresholding technique described above with β being the thresholding factor. Let $\mathbf{x}(S_\beta)$ be the vector $\mathbf{x}$ restricted to the set S_β. That is, there are entries in $\mathbf{x}(S_\beta)$ only for objects in S_β. Let $(LP2(S_\beta))$ be $(LP2)$ (parameterized by β) restricted to the assignment space $S_\beta \times \mathcal{O}_2 \subseteq \mathcal{O}_1 \times \mathcal{O}_2$. We deem the upcoming Condition 3.2 to be a natural one for the rejection problem $\mathcal{P}_R$. Intuitively, rejecting the objects in $\mathcal{O}_1 - S_\beta$ should not make it any "harder" to assign the objects in S_β. On the contrary, any previous assignment from S_β to $\mathcal{O}_2$ should remain valid, i.e. satisfy the linear constraints.

Condition 3.2 *Let* $(\mathbf{x},\mathbf{y},\mathbf{z})$ *be a feasible solution to* $(LP1_R)$. *Then* $(\mathbf{x}(S_\beta),\mathbf{y}(S_\beta))$ *is a feasible solution to* $(LP2(S_\beta))$, *for any* β.

Our algorithm $\mathcal{B}$ to get a solution for $\mathcal{P}_R$ is as follows. First solve $(LP1_R)$. For each i, if $\frac{1}{s_i}\sum_j x_{ij} < \beta$, then include $i \in \mathcal{O}_1$ in the set $\bar{S}$ of rejected objects. Else include $i \in S$. Trivially, $S \cup \bar{S} = \mathcal{O}_1$ and $S \cap \bar{S} = \emptyset$. $\beta \in (0,1]$ is a value that will be chosen later to optimize the performance guarantee, based on $f(\beta)$. If $\mathcal{P}$ satisfies Condition 3.2 then the resulting fractional solution is a solution to (LP2(S)). Convert it to a solution to (LP1) (as described in Condition 3.1) and follow algorithm $\mathcal{A}$ to assign objects in S to $\mathcal{O}_2$.

Theorem 3.1 *Let* $\mathcal{P}$ *satisfy Conditions 3.1 and 3.2. If there is a* ρ*-approximation algorithm for* $\mathcal{P}$ *w.r.t* $(LP1)$, *then* $\mathcal{B}$ *is a* $(\max\{f(\beta)\rho, \frac{1}{1-\beta}\})$*-approximation algorithm for* $\mathcal{P}_R$ *w.r.t* $(LP1_R)$.

Proof. Let $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ be an optimal solution to $(LP1_R)$ with objective value $LP1_R^* = \gamma + \psi + \delta$, where $\gamma = \mathbf{c^T x}^*$, $\psi = \mathbf{d^T y}^*$, and $\delta = \mathbf{g^T z}^*$. The cost of rejecting objects in $\bar{S}$ is at most $\frac{\delta}{1-\beta}$. The term γ can be written as $\gamma^S + \gamma^{\bar{S}}$, where γ^S is the optimal cost of fractionally assigning objects in S to objects in $\mathcal{O}_2$ and $\gamma^{\bar{S}}$ is the optimal cost of fractionally assigning objects in $\bar{S}$ to objects in $\mathcal{O}_2$. Likewise, $\psi = \psi^S + \psi^{\bar{S}}$. Denote by $\mathbf{x}^*(S)$ the vector $\mathbf{x}^*$ restricted to elements x_{ij}^* such that $i \in S$ and by $\mathbf{y}^*(S)$ the corresponding restriction of the vector $\mathbf{y}^*$. By Condition 3.2, $\mathbf{q}$=$(\mathbf{x}^*(S), \mathbf{y}^*(S))$ is a feasible solution to $(LP2(S))$ of $\mathcal{P}(S)$, with value $v = \gamma^S + \psi^S$. Since Condition 3.1 is satisfied by $\mathcal{P}$, $\mathbf{q}$ can be converted to a feasible solution to $(LP1(S))$ of value at most $f(\beta)\rho v$. Therefore $\mathcal{B}$ is a $(\max\{f(\beta)\rho, \frac{1}{1-\beta}\})$-approximation algorithm for $\mathcal{P}_R$. ∎

In our applications we will be using the functions $f(\beta) = \frac{1}{\beta}$ and $f(\beta) = \frac{1}{\beta^2}$; we simplify the theorem statement for these cases.

Corollary 3.2 *Let $\mathcal{P}$ satisfy Conditions 3.1 and 3.2 and let $f(\beta) = \frac{1}{\beta}$. If there is a ρ-approximation algorithm for $\mathcal{P}$ w.r.t $(LP1)$, then $\mathcal{B}$ is a $(1+\rho)$-approximation algorithm for $\mathcal{P}_R$. Furthermore, if $f(\beta) = \frac{1}{\beta^2}$ then $\mathcal{B}$ is a $(2/(\rho+2-\sqrt{\rho(\rho+4)}))$-approximation algorithm for $\mathcal{P}_R$.*

Proof. Note that $\max\{\frac{\rho}{\beta}, \frac{1}{1-\beta}\}$ is minimized for $\beta = \frac{\rho}{\rho+1}$ yielding a performance guarantee of $1+\rho$. Similar calculations yield the latter result. ∎

4 Applications – Scheduling Problem

We consider the problem of scheduling jobs on a single machine subject to release dates and precedence constraints, $1|r_j, prec|\sum w_j C_j$. Hall, Schulz, Shmoys and Wein [10] gave the first constant-factor approximation algorithm for this problem. This has been subsequently improved by Schulz and Skutella [20] to the currently best known value of e (≈ 2.719) for the performance guarantee.

Dyer and Wolsey [5] introduced several integer linear programming formulations for scheduling problems, of which two time-indexed formulations have been the basis for designing constant factor approximation algorithms. We will extend the preemptive time-indexed formulation (that was used in [20]) to include the rejection component. The modified formulation uses decision variables y_{jt} and z_j where y_{jt} is 1 if job j is processed in the time period $(t, t+1]$ and 0 otherwise and z_j is 1 if job j is rejected and 0 otherwise. The relaxation to the integer linear program (that corresponds to $(LP1_R)$) is given by

$$\text{minimize} \sum_{j=1}^{n} (w_j C_j + z_j e_j) \tag{8}$$

subject to

$$\frac{1}{p_j} \sum_{t=r_j}^{T} y_{jt} + z_j \quad = \quad 1 \qquad\qquad \forall j \tag{9}$$

$$
\begin{aligned}
\sum_{j=1}^{n} y_{jt} &\leq 1 && t = 0,\ldots,T && (10)\\
\frac{1}{p_j}\sum_{l=r_j}^{t} y_{jl} &\geq \frac{1}{p_k}\sum_{l=r_k}^{t} y_{kl} && \forall j \prec' k, t = 0,\ldots,T && (11)\\
\frac{p_j}{2} + \frac{1}{p_j}\sum_{t=0}^{T} y_{jt}(t+\frac{1}{2}) &= C_j && \forall j && (12)\\
y_{jt} &= 0 && \forall j, t = 0,\ldots,r_j - 1 && (13)\\
y_{jt} &\geq 0 && \forall j, t = r_j,\ldots,T && (14)\\
z_j &\geq 0 && \forall j && (15)
\end{aligned}
$$

where $T = \max_j r_j + \sum_j p_j$ is a trivial upper bound on the length of the schedule. The *soft* precedence constraint, $j \prec' k$, means that at each point in time the completed fraction of job k must not be larger than the completed fraction of its predecessor job j.

In the following proof (for notational convenience) let $(LP1)$, $(LP2)$ and $(LP1_R)$ refer to the corresponding linear programs for this problem as specified in Section 3.

Theorem 4.1 *There is a 4.5-approximation algorithm for* $1|r_j, prec|(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$.

Proof. We shall prove that Conditions 3.1 and 3.2 are satisfied. The result will then follow by substituting 2.719 for ρ in Corollary 3.2.

Let $(LP1)$, $(LP2)$ and $(LP1_R)$ be the corresponding formulations of the problem with decision variables y_{jt} and z_j as explained earlier. Let the solution to $(LP1_R)$ be $(\mathbf{y}^*, \mathbf{z}^*)$.

Let S be the set of jobs accepted for scheduling according to algorithm $\mathcal{B}$ of Section 3. Define vector $\mathbf{y}^*(S)$, which is a restriction of the vector $\mathbf{y}^*$, as follows: $y^*_{jt}(S) = y^*_{jt}$ if $j \in S$. To prove that Condition 3.2 is satisfied we need to show that $\mathbf{y}^*(S)$ is a feasible solution to $(LP2(S))$. That is, we need to show that the precedence constraints are not violated by the set of jobs accepted for scheduling. To see this, consider a precedence constraint $j \prec k$. This precedence constraint will be violated if job k is accepted and job j is rejected. Let job k be accepted. Therefore $\sum_t y^*_{kt} \geq \beta \cdot p_k$. Since $(\mathbf{y}^*, \mathbf{z}^*)$ is a feasible solution to $(LP1_R)$ and $j \prec' k$ we have from constraint (11), $\frac{1}{p_j}\sum_{l=r_j}^{T} y^*_{jl} \geq \frac{1}{p_k}\sum_{l=r_k}^{T} y^*_{kl} \geq \beta$. Therefore, $\frac{1}{p_j}\sum_{l=r_j}^{T} y^*_{jl} \geq \beta$ and hence job j is also accepted. Therefore, the set of accepted jobs respect the precedence constraints. Therefore Condition 3.2 is satisfied.

To prove that Condition 3.1 is satisfied consider the feasible solution $\mathbf{y}^*(S)$ to $(LP2(S))$. For ease of notation, we will just write $\mathbf{y}^*$ instead of $\mathbf{y}^*(S)$. Let us focus on a job j (that is accepted for scheduling). Let t be such that $\sum_{l=r_j}^{t-1} y^*_{jl} \leq \beta \cdot p_j$ and $\sum_{l=r_j}^{t} y^*_{jl} \geq \beta \cdot p_j$. We split $y^*_{jt} = y'_{jt} + y''_{jt}$ such that $\sum_{l=r_j}^{t-1} y^*_{jl} + y'_{jt} = \beta \cdot p_j$. Create vector $\hat{\mathbf{y}}$ by setting $\hat{y}_{jl} = \frac{1}{\beta} y^*_{jl}$ for $l = r_j, \ldots, t-1$, $\hat{y}'_{jt} = \frac{1}{\beta} y'_{jt}$ and $\hat{y}_{jl} = 0$ for all other l. Therefore, the value of the solution $\hat{\mathbf{y}}$ is $1/\beta$ times the value of the solution $\mathbf{y}^*$. But $\hat{\mathbf{y}}$ is not a feasible solution to $(LP2(S))$ as it violates the capacity constraints. We are exceeding the capacity in each time period by at most $1/\beta$ units. To accomodate this excess, scale the time horizon by $1/\beta$. So the jobs that were originally assigned to the time period $(t, t+1]$ will now be assigned to the time interval $(\frac{1}{\beta}t, \frac{1}{\beta}(t+1)]$. This will increase the fractional completion time of a job by at most a factor of $1/\beta$. Therefore we have constructed a feasible solution to $(LP1(S))$ from the feasible solution $\mathbf{y}^*$ to $(LP2(S))$ and the value of the solution to $(LP1(S))$ is at most $1/\beta^2$ of the value of the solution to $(LP2(S))$. Therefore, Condition 3.1 is satisfied and hence Corollary 3.2 can be applied yielding the result. ■

Notice that Theorem 4.1 only gives a pseudo-polynomial time approximation algorithm since there are exponentially many variables y_{jt}. This drawback can be overcome by using an idea of Hall, et al [11] where the time horizon is subdivided at geometrically increasing points. So instead of an exponential number of time periods we have a polynomial number of time intervals. But the price paid is through a slightly increased performance guarantee. For the problem without rejection, this technique increases the performance guarantee by ε (for any $\varepsilon > 0$). Applying this technique to the rejection variant of the problem it can be seen that the rest of the proof of Theorem 4.1 carries through. Therefore,

Corollary 4.2 *There is a polynomial time $(4.5+\epsilon)$-approximation algorithm for $1|r_j, prec|$-$(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$, for any $\epsilon > 0$.*

The rejection technique can be easily extended to the case of identical parallel machines. Consider the problem $P|r_j|\sum w_j C_j$. In this case, we can consider that each job j is assigned to a machine-time pair (i, t). Schulz and Skutella [20] give a 2-approximation algorithm for this problem. As before, it can be shown that Conditions 3.1 and 3.2 are satisfied yielding a 3.74-approximation algorithm for the rejection variant of the problem. But recall that in Section 2.4 we give a 2-approximation algorithm for a more complex problem $R|r_j|(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ using direct reduction.

Munier, Queyranne and Schulz give a 4-approximation algorithm for $P|r_j, prec|\sum w_j C_j$ [16]. As in the single machine case, we can extend this to get a 5.83-approximation for the rejection variant of the problem.

We finally remark that the techniques developed in Section 3 are quite general and can be applied to other classes of optimization problems also. As an example, we apply these techniques to the *uncapacitated facility location problem* [23, 9, 4]. Due to space constraints, we state here only the theorem and include the proof in [25].

Theorem 4.3 *There is a 2.736-approximation algorithm for the uncapacitated facility location problem with rejection.*

Acknowledgments. We thank David Shmoys for providing pointers to the literature and giving us access to [14], and David Williamson for helpful discussions.

References

[1] Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall, and L. Stougie. Multiprocessor scheduling with rejection. In *Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms*, pages 95–103, 1996.

[2] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. Approximation techniques for average completion time scheduling. In *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms*, pages 609–618, January 1997.

[3] F. Chudak. A min-sum 1.5-approximation algorithm for scheduling unrelated parallel machines. *Journal of Scheduling*, 2(2), 1999.

[4] F. A. Chudak. Improved approximation algorithms for uncapacitated facility location. In R. E. Bixby, E. A. Boyd, and R. Z. Ríos-Mercado, editors, *IPCO: 6th Integer Programming and Combinatorial Optimization Conference*, volume 1412 of *Lecture Notes in Computer Science*, pages 180 – 194. Springer, Berlin, 1998.

[5] M.E. Dyer and L.A. Wolsey. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*, (26):255–270, 1990.

[6] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

[7] M. Goemans. Improved approximation algorithms for scheduling with release dates. In *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms*, pages 591–598, 1997.

[8] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[9] S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[10] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, (3):513–544, August 1997.

[11] L. A. Hall, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. In *Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms*, pages 142–151, January 1996.

[12] O. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, pages 463–468, 1975.

[13] E. L. Lawler and J. M. Moore. A functional equation and its application to resource allocation and sequencing problems. In *Management Science*, volume 16, pages 77–84, 1969.

[14] E. L. Lawler and D. B. Shmoys. Chapter 5: Weighted number of late jobs (preliminary version). To appear in: J.K. Lenstra and D.B. Shmoys (eds.) Scheduling, Wiley.

[15] W.L. Maxwell. Personal communication. 1996.

[16] A. Munier, M. Queyranne, and A. S. Schulz. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. In R. E. Bixby, E. A. Boyd, and R. Z. Ríos-Mercado, editors, *IPCO: 6th Integer Programming and Combinatorial Optimization Conference*, volume 1412 of *Lecture Notes in Computer Science*, pages 367 – 382. Springer, Berlin, 1998.

[17] I. M. Ovacik and R. Uzsoy. *Decomposition Methods for Complex Factory Scheduling Problems.* Kluwer Academic Publishers, 1997.

[18] C. Phillips, C. Stein, and J. Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming B*, 82:199–224, June 1998.

[19] M. H. Rothkopf. Scheduling independent tasks on parallel processors. In *Management Science*, volume 12, pages 437–447, 1966.

[20] A. S. Schulz and M. Skutella. Random–based scheduling: New approximations and LP lower bounds. In J. Rolim, editor, *Randomization and Approximation Techniques in Computer Science*, volume 1269 of *LNCS*, pages 119 – 133. Springer, Berlin, 1997. Proceedings of the International Workshop RANDOM'97.

[21] A. S. Schulz and M. Skutella. Scheduling–LPs bear probabilities: Randomized approximations for min–sum criteria. In R. Burkard and G. Woeginger, editors, *Algorithms – ESA'97*, volume 1284 of *LNCS*, pages 416 – 429. Springer, Berlin, 1997. Proceedings of the 5th Annual European Symposium on Algorithms.

[22] S. Seiden. Preemptive multiprocessor scheduling with rejection. *Theoretical Computer Science*, 2000. To appear.

[23] D. B. Shmoys, É. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 265–274, 1997.

[24] W.E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[25] R. N. Uma. *Theoretical and Experimental Perspectives on Hard Scheduling Problems.* PhD thesis, Polytechnic University, Brooklyn, NY, 2000.

[26] G. J. Woeginger. When does a dynamic programming formulation guarantee the existence of an FPTAS ? In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 820–829, 1999.