Approximation Schemes for Constrained Scheduling Problems

Leslie A. Hall[*] O. R. Center, M.I.T.

David B. Shmoys[†] Dept. of Mathematics, M.I.T.

Abstract

In this paper we consider several constrained scheduling problems. We describe the first polynomial approximation schemes for the problem of minimizing maximum completion time in a two-machine flow shop with release dates, and for the problem of minimizing maximum lateness for the single- and parallel-machine problem with release dates. All of these algorithms are based upon the notion of an *outline*, a set of information with which it is possible to compute, with relatively simple procedures and in polynomial time, an optimal or near-optimal solution to the problem instance under consideration. Finally, we discuss two related precedence-constrained scheduling problems and present new approximation results.

1 Introduction

In this paper we consider several constrained machine scheduling problems, all of which are strongly $\mathcal{NP}$-hard, and present the first polynomial approximation schemes for them. All of these algorithms are based on the notion of an *outline*, a set of information from which it is possible to compute, with relatively simple procedures and in polynomial time, an optimal or near-optimal solution to the problem instance under consideration. In the final section we discuss two related precedence-constrained scheduling problems and present new approximation results.

We begin by defining the models for the problems under consideration. The first model consists of n jobs and m identical machines that operate in parallel. Each job j has a *processing time* p_j, and must be processed without interruption for time p_j on any one of the m machines. In addition, each job j has a *release date* r_j, when it becomes available for processing, and a *delivery time* q_j. Each job's delivery begins immediately after its processing has been completed, and all jobs may be delivered simultaneously.

A *schedule* Σ for an instance of this problem consists of a set of starting times $\sigma_1, \ldots, \sigma_n$ with $\sigma_j \geq r_j$, $j = 1, \ldots, n$, and an assignment of jobs to machines so that each machine processes at most one job at a time. For a given schedule, we define $C_j := \sigma_j + p_j + q_j$ to be the *completion time* of job j, and the object of the problem is to minimize, over all possible schedules, $C_{max} := \max_j C_j$.

The problem as stated is equivalent to that with release dates and *due dates*, d_j, rather than delivery times, in which case the objective is to minimize the maximum lateness, $L_j = \sigma_j + p_j - d_j$, of any job j. When considering the performance of approximation algorithms, the delivery-time model is preferable (see [7, 3]). Because of this equivalence, we shall denote the problem as $P|r_j|L_{max}$, using the notation of Graham *et al.* [2]. The special case of this problem in which there is a single machine shall be denoted $1|r_j|L_{max}$.

The second model that we consider is a two-machine flow shop. This problem consists of two machines, $M1$ and $M2$, and n jobs. Each job j must be processed first on $M1$ for time a_j and then on $M2$ for time b_j. Each job j has a release date r_j, when it becomes available for processing on $M1$. In this model, a schedule consists of a set of $M1$-starting times $\sigma_1, \ldots, \sigma_n$, and $M2$-starting times $\tau_1, \ldots, \tau_n$, so that each job completes its processing on $M1$ before it begins processing on $M2$, and each machine may process at most one job at a time. The *completion time* of job j (with respect to a given schedule) is $C_j := \tau_j + b_j$, and the object is to minimize, over all schedules, $C_{max} := \max_j C_j$. In the notation of Graham *et al.*, this problem is denoted $F2|r_j|C_{max}$.

In the algorithms discussed below, we shall frequently refer to "scheduling" a particular ordered list of jobs on a particular machine. We use this term in the natural sense, to mean processing the jobs on the machine with as little idle time as possible, while keeping them in the

*Current address: Dept. of Civil Eng. & Operations Research, Princeton University.

†Current address: School of Operations Research & Ind. Eng., Cornell University. Research partially supported by an NSF Presidential Young Investigator award with matching support from UPS, IBM, and Sun Microsystems, by Air Force contract AFOSR-86-0078.

CH2806-8/89/0000/0134/$01.00 © 1989 IEEE

specified order.

Next we turn to the concepts of approximation algorithms and outlines, which we define with respect to a minimization problem, Π. For any instance I of Π, we denote an optimal solution of I by OPT(I) (or simply OPT) and we denote the value of the optimal solution by z^*(I) (or simply z^*). A $(1+\epsilon)$-*approximation algorithm* for Π is an algorithm guaranteed to deliver, for any feasible instance I of Π, a feasible solution to I with value at most $(1+\epsilon)z^*$(I), in time polynomial in the size of I. (Note that ϵ is a fixed number, and thus the running time may depend upon $1/\epsilon$ in any manner.) A *polynomial approximation scheme* for Π is a family of algorithms $\{A_\epsilon : \epsilon > 0\}$ such that for each $\epsilon > 0$, A_ϵ is a $(1+\epsilon)$-approximation algorithm for Π.

The algorithms that we present here essentially use two building blocks. The first, a well-known technique, consists of preprocessing the data to form an approximate version of the original problem that has a simpler structure. Typically, such preprocessing steps consist of rounding data in such a way that the optimal value of the rounded problem is nearly equal to that of the original problem. We believe that the extent to which this technique is used in these results is unprecedented. The second, more novel technique that we use is a $(1+\epsilon)$-*optimal outline scheme.*

Definition. An *outline scheme* for a problem Π is a labeling of feasible solutions such that, for each feasible solution x, the associated label, called an *outline*, provides concise information about x.

For example, one outline scheme for a scheduling problem labels each job with its starting time in the schedule. Of course, this outline is particularly useful because with it we can reconstruct the original schedule, x. Most outlines reveal only partial information about their corresponding schedule or schedules; even so, this partial information might still be enough to construct a schedule that is "close to", or almost as good as, the outline's corresponding schedule or schedules.

Definition. For $\epsilon > 0$, a $(1+\epsilon)$-*optimal outline scheme* for Π is an outline scheme for Π and an algorithm A with the following property: given any outline ω for an instance I, Algorithm A delivers a feasible solution to I of value at most $(1+\epsilon)$ times the value of any feasible solution of I that is labeled with ω.

In particular, given an outline corresponding to an optimal solution to I, the algorithm delivers a solution of value at most $(1+\epsilon)z^*$(I). Moreover, if for every instance I of Π, the number of distinct outlines associated with a $(1+\epsilon)$-optimal outline scheme is polynomial in the size of I (or if we can determine a polynomial-sized subset of outlines that is guaranteed to contain one corresponding to an optimal solution), then by running the algorithm on every outline for I, we obtain a $(1+\epsilon)$-approximation algorithm for Π. We refer to a 1-optimal outline scheme as simply an optimal outline scheme.

2 Two easy applications

In this section, we present polynomial approximation schemes for the two problems $1|r_j|L_{max}$ and $F2|r_j|C_{max}$. These problems share the property that, if all jobs are released at the same time (or, equivalently, if all $r_j = 0$), then each problem may be solved to optimality in polynomial time. Furthermore, each has an optimal outline scheme based upon an algorithm that is a natural extension of the algorithm for the version with equal release dates.

First, we perform a preprocessing step that will simplify the algorithms that follow. Consider any of the problems $P|r_j|L_{max}$, $1|r_j|L_{max}$, or $F2|r_j|C_{max}$, and let $\Phi = \max_j r_j$. We will show that for each of these problems, we can restrict attention to the case where there are only a constant number of distinct release dates. Consider the following algorithm for the unrestricted version. Round each release date down to the nearest multiple of $\Phi\epsilon/2$. Apply a $(1+\epsilon/2)$-approximation algorithm that can handle $1+2/\epsilon$ distinct release dates, and then add $\Phi\epsilon/2$ to the each job's starting time in the output. This yields a feasible schedule for the original instance. It is easy to see that this is a $(1+\epsilon)$-approximation algorithm. Thus in the remainder of this paper, without loss of generality we shall restrict our attention to problems that have a fixed number of distinct release dates.

2.1 $1|r_j|L_{max}$

The special case of this problem in which all $r_j = 0$ is solved optimally by the following algorithm, known as Jackson's rule [5]: Schedule the jobs without idle time in order of non-increasing delivery times. Jackson's rule does not solve the more general version of the problem with release dates; however, we can use it for the general problem in the context of an optimal outline scheme. Let us fix a specific instance I of $1|r_j|L_{max}$, and suppose that the release dates $r_1, \ldots, r_n$ take on κ distinct values, which we denote by $\rho_1 < \cdots < \rho_\kappa$. We set $\rho_{\kappa+1} = \infty$. In addition, we fix a specific feasible solution Σ for I, consisting of job starting times $\sigma_1, \ldots, \sigma_n$. If $\rho_i \le \sigma_j < \rho_{i+1}$, then we call ρ_i the *effective release date* (with respect to Σ) of job j, and we denote it by $\overline{r}_j$. (Note that $\overline{r}_j \ge r_j$.) Of course, the effective release dates depend upon the choice of Σ; we observe, however, that if each job's release date is replaced by its effective release date $\overline{r}_j$ for some fixed choice of Σ, then Σ is feasible for I'.

Theorem 1
The set of effective release dates $\overline{r}_1, \ldots, \overline{r}_n$, with the algorithm sketched below, is an optimal outline scheme for $1|r_j|L_{max}$.

Sketch of Proof: For any instance I and feasible schedule Σ, given the set of effective release dates for all jobs it is possible to apply the following algorithm to obtain a schedule no worse than Σ.

1. Partition the jobs into sets $S_i := \{j : \overline{r}_j = \rho_i\}$, $i = 1, \ldots, \kappa$;
2. Order each set S_i by non-increasing delivery time;
3. Schedule the jobs in S_1 (in order), followed by those in $S_2, \ldots$, followed by those in S_κ.

Let us refer to the schedule generated by the algorithm above as Σ'. In both schedules Σ and Σ' all of the jobs in S_i are scheduled before all of those in S_{i+1}, for every i; thus, the first job in S_i to be processed is begun at time $\max_{1 \le h \le i} \{\rho_h + \sum_{l=h}^{i-1} (\sum_{j \in S_l} p_j)\}$, independent of the order in which the jobs in each of the sets S_l is processed. Thus the only difference between Σ and Σ' is the order in which each set S_i is processed. Each subproblem, on a set S_i, is essentially an instance of the problem for which all release dates are equal, (to ρ_i), and thus Jackson's rule, invoked by Σ', provides the optimal ordering for each subproblem. ∎

Next we describe a $(1+\epsilon)$-optimal outline scheme, based upon the ideas in the optimal outline scheme presented above. As before, we fix a feasible schedule Σ and define effective release dates $\overline{r}_j$ with respect to Σ. Let $\delta = \Phi\epsilon/2\kappa$, for $\Phi = \sum_j p_j$. We divide the jobs into two sets, according to the length of their processing times, $A = \{j : p_j \ge \delta\}$ and $B = \{j : p_j < \delta\}$. Define $I_i := \{j \in A : \overline{r}_j = \rho_i\}$. Let $H_i := \sum(p_j : \overline{r}_j = \rho_i \text{ and } j \in B)$, and let $N_i := \lceil H_i/\delta \rceil \delta$, for $i = 1, \ldots, \kappa$; that is, N_i is the approximate amount of time in the interval $[\rho_i, \rho_{i+1})$ that is spent processing the small jobs.

Theorem 2 The set $\{(I_i, N_i) : i = 1, \ldots, \kappa\}$, with the algorithm outlined below, is a $(1+\epsilon)$-optimal outline scheme for $1|r_j|L_{max}$.

Sketch of Proof: First we describe the algorithm that uses an outline to obtain a solution Σ' guaranteed to be almost as good as Σ. For all i, starting with $i = 1$, we construct a set of jobs $J_i \subseteq B$ as follows: among all "available" jobs $j \in B$ (*i.e.*, $r_j \le \rho_i$ and j not yet assigned) we continue to assign a job with largest delivery time to J_i, until the first time that $\sum_{j \in J_i} p_j \ge N_i$ (or until there are no more available small jobs); then we stop assigning jobs to J_i and move on to J_{i+1}. (The definition of N_i ensures that every job in B gets assigned to some set J_i.) Now let $S'_i := I_i \cup J_i$, and construct a schedule as in the optimal-outline case: that is, order each set S'_i by non-increasing delivery time, and concatenate the ordered sets to obtain the ordering for the schedule Σ'.

As before, define S_i to be the set of jobs j with effective release dates equal to ρ_j. Also, let $z(\Sigma)$ and $z(\Sigma')$ denote the values of the schedules generated by Σ and Σ', respectively. To see why this process yields a $(1+\epsilon)$-approximation to Σ, first we observe that, for any i, $\sum_{j \in J_i} p_j - H_i < 2\delta$. Thus each set S'_i as a unit begins its scheduling in Σ' at most $2\delta(i-1)$ time units later than S_i begins in Σ. Furthermore, because of the greedy manner in which the sets J_i are constructed, it is possible to show that the delay just described is the only error introduced, so that $z(\Sigma') \le z(\Sigma) + 2\delta\kappa = z(\Sigma) + \Phi\epsilon \le (1+\epsilon)z(\Sigma)$. Since the processing times of the jobs in B are small, these jobs cannot individually contribute large errors with respect to the position of other jobs in the schedule, by being placed badly; thus it is possible to obscure the individual identities of these small jobs and group them greedily into packets of approximately the correct size, thereby incurring an error only due to the size of the packets. ∎

For any instance, the number of distinct outlines is equal to the number of ways in which jobs in A can be assigned effective release dates, times the number of ways to select the N_i. By applying the $(1+\epsilon)$-optimal outline scheme to each of these constant number of outlines, we obtain a $(1+\epsilon)$-approximation algorithm.

Theorem 3 The $(1+\epsilon)$-optimal outline scheme described above provides a polynomial approximation scheme for $1|r_j|L_{max}$.

2.2 $F2|r_j|C_{max}$

The development of this section roughly parallels that of the previous section. First, it is not hard to show that, for any instance of $F2|r_j|C_{max}$, there exists an optimal solution for which the job ordering on machine $M1$ is identical to the ordering on $M2$ [6]. Thus we will restrict our attention to schedules of this form.

In the case in which all $r_j = 0$, an algorithm due to Johnson solves the problem optimally [6]: first partition the jobs into two sets, those with $a_j \le b_j$, and those with $a_j > b_j$. Schedule the former set of jobs first, in order of non-decreasing a_j; then schedule the remaining jobs, in order of non-increasing b_j. As in the previous section, we use the notion of effective release dates to obtain an optimal outline scheme.

Theorem 4 A set of effective release dates $\overline{r}_1, \ldots, \overline{r}_n$, with an algorithm based on Johnson's algorithm, is an optimal outline scheme for $F2|r_j|C_{max}$.

The algorithm for this optimal outline scheme is virtually identical to the algorithm for $1|r_j|L_{max}$, in Theorem

1, if one substitutes "Johnson's algorithm" for "Jackson's rule"; we omit the proof of Theorem 4.

Next we derive a $(1+\epsilon)$-optimal outline scheme from the optimal outline scheme just presented. We fix a feasible schedule Σ and define effective release dates $\overline{r}_j$ with respect to Σ. Let $\delta = \Phi\epsilon/3\kappa$, with $\Phi = \sum_{j=1}^{n} a_j$. Again we partition the jobs into two sets A and B, with $A = \{j : a_j \geq \delta\}$ and $B = \{j : a_j < \delta\}$; and we define $I_i = \{j \in A : r_j = \rho_i\}$, $H_i = \sum(p_j : \overline{r}_j = \rho_i$ and $j \in B)$, and $N_i = \lceil H_i/\delta \rceil \delta$, for $i = 1, \ldots, \kappa$.

The same intuition underlies both problems $1|r_j|L_{max}$ and $F2|r_j|C_{max}$. "Small" jobs (those in B), if defined and scheduled properly, have little combinatorial impact on the schedule; "large" jobs (those in A) can be tried in every configuration, because there are not too many of them. The only difference in the algorithm needed for $F2|r_j|C_{max}$ is the greedy manner in which jobs $j \in B$ are selected: here, the rule is that jobs with large processing times on $M2$ relative to their processing times on $M1$ should be scheduled early, in order to minimize the amount of idle time that $M2$ experiences.

Theorem 5 The set $\{(I_i, N_i) : i = 1, \ldots, \kappa\}$, is the basis of a $(1+\epsilon)$-optimal outline scheme for $F2|r_j|C_{max}$, which in turn is the basis of a polynomial approximation scheme for $F2|r_j|C_{max}$.

3 $P|r_j|L_{max}$

In this section, we present a polynomial approximation scheme for $P|r_j|L_{max}$; this result is the common generalization of Theorem 3 and a result of Hochbaum and Shmoys [4], for $P||C_{max}$. The development of this scheme requires several layers of preprocessing by repeated rounding of the input data. Throughout this discussion we assume that $\epsilon > 0$ is an arbitrary constant. Furthermore, we assume that an estimate $\tilde{z}$ for the optimal value, such that $z^* \leq \tilde{z} \leq 2z^*$, is known (*e.g.*, see Section 4). At the core of the algorithm is a greedy scheduling rule similar to that used for $1|r_j|L_{max}$. We view the preprocessing steps as successive transformations to problems with simpler structure. We have already rounded release dates to obtain a fixed number of distinct ones; rounding delivery times is completely analogous to the release-date rounding, and thus we shall assume that there is a fixed number κ of distinct delivery times, as well as κ release dates, as we work toward the $(1+\epsilon)$-approximation algorithm. Next, let us again partition the jobs into two sets according to processing size, with A denoting the set of jobs requiring large processing time ($\geq \delta$) and B denoting the set of small jobs. Given Π and $\delta > 0$, consider the problem Π' derived from Π as follows: for each job j with $p_j \geq \delta$, p_j is an integer multiple of $\epsilon\delta$. Note that for any instance I of Π, an instance I' of Π' can be formed by setting $p'_j = \lfloor p_j/(\epsilon\delta) \rfloor \epsilon\delta$ for $j \in A$, and $p'_j = p_j$ otherwise.

Lemma 1 A $(1+\epsilon)$-approximation algorithm for Π yields a $(1+3\epsilon)$-approximation algorithm for Π'.

Since we know the estimate $\tilde{z}$ for I, and since $\max_j p_j \leq z^* \leq \tilde{z}$, by choosing $\delta = \epsilon\tilde{z}/4\kappa^2$ we can limit the number of distinct sizes of rounded jobs to $\tilde{z}/\epsilon\delta = 4\kappa^2/\epsilon^2$, a fixed number. (This choice for δ is obtained by balancing the need for few types of jobs without too great a loss of precision.) Thus we will further assume that all jobs j with $p_j \geq \delta$ have processing times that are multiples of $\epsilon\delta$. Then each job in A has one of κ distinct release dates, κ delivery times, and $4\kappa^2/\epsilon^2$ processing times. Hence there is a fixed number $\Lambda = 4\kappa^4/\epsilon^2$ of distinct *job types* for jobs in A; we label the job types $1, \ldots, \Lambda$.

Let us fix a particular machine, and a particular schedule, Σ. For this machine, we define sets S_i, with respect to Σ, in a manner similar to the single-machine case (recall that $\overline{r}_j$ is the effective release date of job j, with respect to Σ): $S_i = \{j : \overline{r}_j = \rho_i$ and j is assigned to the fixed machine$\}$. Let λ_{ih} be the number of jobs $j \in A \cap S_i$ of type h. Also, we let $M_{ik} = \sum_{j \in S_i \cap B}(p_j : q_j = \xi_k)$, that is, M_{ik} is the amount of small-job-time with effective release ρ_i and delivery time ξ_k, in Σ on machine h. We let $\nu_{ik} = \lceil M_{ik}/\delta \rceil$, $N_{ik} = \nu_{ik}\delta$, and we note that $N_{ik} - M_{ik} < \delta$. Finally, we need the following concept.

Definition A *machine configuration* is a pair (λ, ν), with $\lambda = (\lambda_{11}, \lambda_{21}, \ldots, \lambda_{\kappa 1}, \ldots, \lambda_{\kappa\Lambda})$, $\nu = (\nu_{11}, \nu_{12}, \ldots, \nu_{1\kappa}, \nu_{21}, \ldots, \nu_{\kappa\kappa})$, such that λ_{ih} is the number of jobs $j \in A$ of type h with $\overline{r}_j = \rho_i$, and $\nu_{ik}\delta$ is an overestimate of the small-job-time with delivery time ξ_k and effective release date ρ_i, so that $\nu_{ik} = \lceil M_{ik}/\delta \rceil$.

This allows us to say that, for a given schedule, a particular machine has a certain configuration. We denote the configurations as $l = 1, \ldots, \Gamma$. Now we may construct a $(1+\epsilon)$-optimal outline scheme for the problem.

Theorem 6 For any schedule, let x_l be the number of machines with configuration l for that schedule, so that $x = (x_1, \ldots, x_\Gamma)$ defines an outline for the schedule. Then the set of these outlines over all schedules, with the algorithm described below, is a $(1+\epsilon)$-optimal outline scheme for our restricted version of $P|r_j|L_{max}$.

Sketch of Proof: Consider the following algorithm, which uses the outline of Σ as part of its input to produce a schedule Σ'. For each machine configuration a, schedule x_a machines as follows:

1. For each i, we select a set of λ_{ih} jobs from A of job-type h, $h = 1, \ldots, \Lambda$, that have not yet been selected; call this set I_i. We further partition I_i according to the delivery time of each job: $I_i = \bigcup_{k=1}^{\kappa} I_{ik}$, and

$I_{ik} = \{j \in I_i : q_j = \xi_k\}$.
2. For $i = 1, \ldots, \kappa$ and $k = 1, \ldots, \kappa$, we construct a set of jobs J_{ik} as follows: among all "available" jobs of B with delivery time ξ_k, we continue to assign a job with largest release time to J_{ik}, until the first time that $\sum_{j \in J_{ik}} p_j \geq N_{ik}$ (or until there are no more available small jobs with delivery time ξ_k); then we stop assigning jobs to J_{ik} and move on to the next set.
3. Now we schedule the sets in the following order: $I_{11}, J_{11}, I_{12}, J_{12}, \ldots, I_{1\kappa}, J_{1\kappa}, I_{21}, J_{21}, \ldots, I_{\kappa\kappa}, J_{\kappa\kappa}$.

To see that the algorithm works correctly, first we must verify that every job gets assigned to some machine's schedule. Clearly, every job in A gets assigned to some set I_{ik}, by the definition of the outline. The definition of N_{ik}, plus the fact that we select jobs for J_{ik} in order of non-increasing release time, guarantees that every job in B gets assigned to some set J_{ik}. Finally, we need to show that for each machine, every job scheduled in Σ' on that machine completes delivery no later than $(1+\epsilon)z(\Sigma)$. By similar reasoning to the single-machine case, we can show that each block of jobs $I_{ik} \cup J_{ik}$ is delayed at most $(i-1)\kappa(2\delta) + (k-1)(2\delta) \leq 2\kappa^2\delta$ with respect to its counterpart in Σ; because of the structure of the outline, these are the only delays incurred. Thus $z(\Sigma') \leq z(\Sigma) + 2\kappa^2\delta = z(\Sigma) + \epsilon\tilde{z}/2 \leq (1+\epsilon)z(\Sigma)$. ∎

An optimal solution has the property that at most $\tilde{z}/\delta = 4\kappa^2/\epsilon$ jobs of A are scheduled on any one machine; thus in testing outlines, we may restrict our attention to those with $\lambda_{ih} \leq 4\kappa^2/\epsilon$, for all i and h. Similarly, the number of distinct choices that we need to consider for any value of N_{ik} is $\tilde{z}/\delta = 4\kappa^2/\epsilon$. Since κ and Λ are fixed numbers, the number of relevant machine configurations is fixed; call it γ. Then the number of choices of relevant outlines $x = (x_1, \ldots, x_\gamma)$, is m^γ, a polynomial in m. Therefore by trying every relevant outline, and choosing the best schedule generated overall, we have a $(1+\epsilon)$-approximation algorithm.

Theorem 7 The $(1+\epsilon)$-optimal outline described above provides a polynomial approximation scheme for $P|r_j|L_{max}$.

4 Precedence constraints

In precedence-constrained problems, there is a given partial order $\prec$, where $k \prec j$ has the interpretation that job j can not be processed (on a machine) until job k has completed processing (but the delivery of job k need not have been completed). For the models that we have considered so far, the situation with the additional imposition of precedence constraints is quite different. Graham [1] gives a simple $(2-1/m)$-approximation algorithm for the special case in which $q_j = 0$ and $r_j = 0$ for all j. However, in the intervening twenty-odd years, this factor of 2 has remained one of the many unexplained "barriers" in the worst-case analysis of approximation algorithms. In fact, even in the case where the number of machines is fixed (*e.g.*, $m = 100$), no polynomial-time algorithm is known that has a performance guarantee appreciably better.

We shall discuss two results relating to precedence constrained versions of scheduling problems. For $P|r_j, prec|L_{max}$, we show that a natural extension of Graham's algorithm is a 2-approximation algorithm. For $1|r_j, prec|L_{max}$, we show that there is a 4/3-approximation algorithm. The second result is based in part on an application of a technique of Lenstra [8], which implies the following rule of thumb: for any approximation algorithm that simply runs Jackson's rule on a collection of problems derived from the original instance and chooses the best solution found, any performance guarantee attainable without precedence constraints can also be attained with precedence constraints.

4.1 $P|r_j, prec|L_{max}$

Consider the following on-line *list scheduling* algorithm: whenever a machine is idle, choose any available job to start processing on that machine, where a job is *available*, if it has been released and all of its predecessors (in the precedence relation) have finished processing; if none are available, consider the next point in time when a job becomes available, and continue scheduling then.

Theorem 8 The list scheduling procedure is a 2-approximation algorithm for $P|r_j, prec|L_{max}$.

Sketch of Proof: Call a job *critical* for a schedule if it is one whose delivery is completed last. Consider a schedule produced by the list scheduling algorithm, and let j_1 denote a critical job. We will partition the time used by this schedule into two sets of time intervals, and then argue for each set that the total time in it is at most $z^*(\mathrm{I})$. This yields the claimed bound.

Consider the latest idle time t_1 that precedes the start of processing for j_1. Since job j_1 is not scheduled in this idle time, it is not available before t_1, and so either one of its predecessors has not been completed, or it has not been released. In the former case, let j_2 be a predecessor of j_1 that is processed (for some of its time) after time t_1. Now repeat this argument with job j_2. This process will stop when a job j_k is found that is not available before time t_k because it has not been released before t_k, and so we have constructed a chain C of jobs $j_k \prec \cdots \prec j_2 \prec j_1$.

Partition the schedule's time into two parts: (a) $[0, r_k]$ plus the time that some $j_i \in C$ is processed, as well as the delivery time of j_1; and (b) the remaining time. Since all jobs in C must be processed after time r_k, only

one job in C can be processed at a time, and the delivery of job j_1 must follow the processing of all jobs in C, the time in (a) is at most $z^*(\mathrm{I})$. However, it is not hard to see that no machine is idle during the time in (b), and so this time is also at most $z^*(\mathrm{I})$. ∎

4.2 $1|r_j, prec|L_{max}$

A natural extension of Jackson's rule for handling release dates, due to Schrage, is to schedule the next *available* job with the longest delivery time, whenever the machine becomes idle. Lenstra [8] adapts this procedure to handle precedence constraints, by updating the data as follows: until no further updates apply, if $i \prec j$, then (a) if $r_j < r_i$, set $r_j := r_i$; and (b) if $q_i < q_j + p_j$, set $q_i := q_j + p_j$. It is easy to verify that no feasible schedule is lengthened by these modifications, and that Schrage's extension of Jackson's rule delivers a schedule that satisfies the precedence constraints. Theorem 8 applies to the case $m = 1$, and since the extended Jackson's rule is a further specification of the list scheduling algorithm, it must be a 2-approximation algorithm. In fact, even in this special case, the analysis is tight.

Potts [12] gives a (3/2)-approximation algorithm for the case without precedence constraints, based on the following idea. Let j_c be a critical job for the schedule produced by the extended Jackson's rule. Focus on the latest idle time prior to j_c, and let the jobs scheduled after then, and up through j_c, be called the *critical sequence*. If q_{j_c} is the minimum delivery time for jobs in the critical sequence, then the schedule is optimal. On the other hand, suppose that this is not the case, and let job j_b be the job with delivery time greater than j_c that is scheduled latest in the critical sequence. Since job j_b can be seen as interfering with this proof of optimality of the heuristic schedule, we shall call it the interference job.

It is natural to try to force j_b to be scheduled after j_c, and this can be enforced by updating $r_{j_b} := r_{j_c}$. Potts' algorithm simply runs Schrage's algorithm for n iterations with this update, or until there is no interference job in the schedule generated; the best schedule generated is output. It is natural to view Potts' algorithm as augmenting the precedence relation among the jobs; if the preprocessing technique of Lenstra is incorporated for each iteration then we get a natural extension that handles $1|r_j, prec|L_{max}$.

Theorem 9 The extended Potts' algorithm is a (3/2)-approximation algorithm for $1|r_j, prec|L_{max}$.

We have obtained a (4/3)-approximation algorithm for $1|r_j, prec|L_{max}$ by extending the approach of Potts. The main difference is that we apply another technique of Lenstra, Rinnooy Kan and Brucker [9] to Potts' algorithm, which is based on the following idea: if the release dates are replaced by the delivery times, and vice versa, we obtain an equivalent problem, and yet Schrage's heuristic might well produce schedules of different lengths. Thus, the entire algorithm is run for both the original and the "inverted" instances. By running the essential algorithm this many times, it is possible to piece together sufficient information about the optimal schedule to obtain the tighter bound for an iteration that introduces an inconsistent constraint. Potts' analysis focuses on the existence of a job j with $p_j > z^*(\mathrm{I})/2$, whereas both our algorithm and analysis highlight jobs potentially with $p_j > z^*(\mathrm{I})/3$. There are at most two of these long jobs, i and j, and so the algorithm is run twice, imposing each of $i \prec j$ and $j \prec i$.

Theorem 10 There exists a (4/3)-approximation algorithm for $1|r_j, prec|L_{max}$ based on repeated executions of the extended Jackson rule.

The most natural extension of this result would be to obtain a polynomial approximation scheme for $1|r_j, prec|L_{max}$ by focusing on the set of jobs with processing time more than $\epsilon z^*(\mathrm{I})$, and trying to force them not to be the interference job. However, there does not seem to be "sufficient information" to obtain the tighter bound when an inconsistent precedence constraint is introduced.

References

[1] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–81, 1966.

[2] 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.

[3] L. A. Hall and D. B. Shmoys. Jackson's rule for one-machine scheduling: making a good heuristic better. *Mathematics of Operations Research*, to appear.

[4] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *J. Assoc. Comput. Mach.*. 34:144–162, 1987.

[5] J. R. Jackson. *Scheduling a Production Line to Minimize Maximum Tardiness*. Research Report 43, Management Science Research Project, UCLA, 1955.

[6] S. M. Johnson. Optimal two- and three-stage production schedules with setup times. *Naval Research Logistics Quarterly*, 1:61–68, 1954.

[7] H. Kise, T. Ibaraki, and H. Mine. Performance analysis of six approximation algorithms for the one-machine maximum lateness schedulng problem with ready times. *Journal of the Operations Research Society of Japan*, 22(3):205–224, 1979.

[8] J. K. Lenstra. *Sequencing by Enumerative Methods*. Mathematical Centre Tracts 69, Centre for Mathematics and Computer Science, Amsterdam, 1977.

[9] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[10] J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26:22–25, 1978.

[11] G. B. McMahon and M. Florian. On scheduling with ready times and due dates to minimize maximum latness. *Operations Research*, 23:475–482, 1975.

[12] C. N. Potts. Analysis of a heuristic for one machine sequencing with release dates and delivery times. *Operations Research*, 28:1436–1441, 1980.