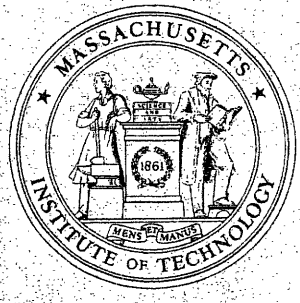
OPERATIONS RESEARCH CENTER

working paper



MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Jackson's Rule for One-Machine Scheduling: Making a Good Heuristic Better

by

Leslie A. Hall and David B. Shmoys

OR 199-89 August 1989

Jackson's Rule for One-Machine Scheduling: Making a Good Heuristic Better

Leslie A. Hall
Operations Research Center
M.I.T.

David B. Shmoys
Department of Mathematics
M.I.T.

August 1988

Abstract

We consider the scheduling problem in which jobs with release times and delivery times are to be scheduled on one machine. We present a 4/3-approximation algorithm for the problem with precedence constraints among the jobs, and two $(1+\epsilon)$-approximation algorithms for the problem without precedence constraints. Finally, we prove a strong negative result concerning a restricted version of the problem with precedence constraints that indicates that precedence constraints make the problem considerably more difficult to solve. At the core of each of the algorithms presented is Jackson's Rule–a simple but seemingly robust heuristic for the problem.

1 Introduction

Consider the following machine scheduling problem: n jobs are to be processed on one machine. Each job i has a *release time* r_j, before which it may not be processed, a *processing time* $p_j > 0$, and a subsequent *delivery time* q_j. Each job must begin processing on the machine sometime after its release time, and its delivery begins immediately after processing has been completed. At most one job can be processed at a time, but all jobs may be simultaneously delivered. Preemption is not allowed. The objective is to minimize the time by which all jobs are delivered. We study both the case in which there exist precedence constraints among the jobs and the case in which the jobs are independent.

The problem as stated is equivalent to that with release times and *due dates*, rather than delivery times, in which case the objective is to minimize the maximum lateness of any job. The equivalence is shown by replacing each delivery time q_j by

a due date $d_j = K - q_j$, where K is a constant. Accordingly, the problems may be denoted $1 \mid r_j \mid L_{max}$ and $1 \mid r_j, prec \mid L_{max}$, in the notation of Graham *et al.* [1979].

There are two reasons for studying the delivery-time model rather than the due-date model of the problem. First and foremost, the delivery-time model allows for a meaningful discussion of approximation algorithms. In the due-date model, the maximum lateness L_{max} could equal zero, regardless of the other data in the problem; thus any attempt at approximation algorithms that measure performance in terms of this objective function appears doomed to failure. The second reason for using this model is that it emphasizes the forward-backward symmetry of the problem: by interchanging r_j and q_j for every job j and inverting the precedence relation, we obtain an *inverse* problem with the property that $p_{j_1}, p_{j_2}, \ldots, p_{j_n}$ is an optimal ordering of the jobs for the original problem if and only if $p_{j_n}, p_{j_{n-1}}, \ldots, p_{j_1}$ is optimal for the inverse problem.

This scheduling problem is strongly NP-hard, even when no precedence constraints exist [Lenstra *et al.*, 1977]. A few special cases have been shown to be polynomially solvable, however. If all release times r_j are equal, the problem may be solved using *Jackson's rule* [Jackson, 1955]: schedule the jobs in order of non-increasing delivery time. An analogous result holds for equal delivery times, by applying Jackson's rule to the inverse problem. If all processing times $p_j = 1$ and all release times are integer, then the problem is solved by the *extended Jackson's rule*: whenever the machine is free and one or more jobs are available for processing, schedule an available job with largest delivery time. The problem becomes considerably more complex when all processing times are equal but arbitrary release times are allowed; the intuition behind this fact is that in this latter case, a new job may be released while another job is in the middle of being processed, and thus the new job may be wrongly postponed if one simply uses the extended Jackson's rule. This case is still polynomially solvable, however [Simons, 1978; Garey *et al.*, 1981], using an iterated version of the extended Jackson's rule, in which restrictions are introduced at each iteration. All of the special cases above are polynomially solvable when precedence constraints exist, as well.

Early work analyzing approximation algorithms for this problem includes Kise *et al.* [1979]. The best approximation algorithm until now was given by Potts [1980], and it achieves a schedule of length no more than 3/2 the length of the optimal schedule; in other words, it is a 3/2-approximation algorithm. Although Potts' algorithm is for the problem without precedence constraints, a simple modification allows it to achieve the same bound with precedence constraints, as will be discussed below.

The best approximation algorithm result that one can hope for is a *fully polynomial approximation scheme*, *i.e.*, a family of algorithms $\{A_\epsilon\}$ such that for any $\epsilon > 0$, A_ϵ produces a schedule of length at most $(1 + \epsilon)$ times the length of the optimal schedule and has running time polynomial in n and $1/\epsilon$. However, because this scheduling problem is strongly NP-hard, no such scheme exists unless P=NP [Garey and Johnson, 1978]. Consequently, the best result that we can hope to obtain is a

polynomial approximation scheme, namely a $(1+\epsilon)$-approximation algorithm whose running time depends exponentially on $1/\epsilon$ (as well as on n). We present two such polynomial approximation schemes below.

In the remainder of this paper, we present the following results. First, we present a modified version of Potts' algorithm [Potts 1980] that produces a schedule of length no more than 4/3 times the length of the optimal schedule; this algorithm can also handle precedence constraints. In Section 3, we present two polynomial approximation schemes for the problem without precedence constraints. The first of these is based on the result that, given a fixed number of different release times, and unary processing times, then the problem is polynomially solvable. Finally, we prove a strong negative result concerning the problem with precedence constraints: given a fixed number of release times, processing times, and delivery times, all of which are encoded in unary, the problem is NP-hard.

It will be seen that the algorithms presented here, like Potts' algorithm and those for the special cases discussed above, all depend in some manner on the extended Jackson's rule. In some fundamental sense, this rule is the "right" one for this problem and proves to be very powerful indeed, given the proper modifications.

2 A 4/3-approximation algorithm

To begin, we introduce some notation. There are n jobs $1, 2, \ldots, n$ to be scheduled on one machine, whose release, processing, and delivery times are given by r_j, p_j, and q_j, respectively, $j = 1, \ldots, n$. We let

$$P = \sum_{j=1}^{n} p_j.$$

We define T^* and T_H, respectively, to be the length of the optimal schedule and the length of the schedule given by the heuristic H under consideration. Clearly,

$$T^* \geq \max\{P, \max_j r_j, \max_j q_j\}.$$

We define here once again the

Extended Jackson's Rule Whenever the machine is free and one or more jobs is available for processing, schedule an available job with largest delivery time.

For any algorithm H, suppose that the algorithm generates a job ordering $j_1, j_2, \ldots, j_n$, and that job j_c finishes last in the schedule, *i.e.*, its delivery time finishes last. Let j_m be the earliest-scheduled job such that no idle time exists between the processing

of jobs j_m and j_c. Then

$$T_H = r_{j_m} + \sum_{h=m}^{c} p_{j_h} + q_{j_c}.$$

The sequence of jobs $j_m, \ldots, j_c$ is called a *critical sequence* for the schedule, and j_c is the *critical job* for that sequence.

When the extended Jackson's rule is used, resulting critical sequences have some nice properties. To begin with, we observe that every job j_h in a critical sequence $j_m, \ldots, j_c$ must have release time $r_{j_h} \geq r_{j_m}$; thus if every job j_h in a critical sequence satisfies $q_{j_h} \geq q_{j_c}$, then the heuristic schedule is optimal. If there exists a job j_b in the sequence such that $q_{j_b} < q_{j_c}$, then it must also be true that $r_{j_b} < r_{j_c}$. Such a job j_b is called an *interference job* for the critical sequence. We state the following well known result without proof [Potts 1980].

Lemma 1 Let $j_m, \ldots, j_c$ be a critical sequence for a schedule generated by the extended Jackson's rule heuristic J, and suppose that j_b is the interference job occurring latest in the sequence, (so that $q_{j_h} \geq q_{j_c}$ for all $h > b$). Then

$$\begin{aligned} T_J - T^* &< p_{j_b}; \\ T_J - T^* &\leq q_{j_c}. \end{aligned}$$

If there is an interference job for every critical sequence, the only hope of improving the schedule is to force some interference job to come after its corresponding critical job. This idea motivates Potts' algorithm.

Algorithm A

repeat

run the extended Jackson's rule on the current problem, and locate a critical sequence;

if an interference job j_b exists **then** set $r_{j_b} := r_{j_c}$

until there does not exist an interference job or n iterations have been performed;

return the shortest schedule generated.

Theorem 1 [Potts 1980] The above algorithm generates a schedule T_A with $T_A/T^* < 3/2$.

Proof There are two cases to be considered. Let us fix some optimal schedule.
Case 1 All precedence constraints introduced are consistent with the optimal schedule. If the algorithm stops at some point because no interference job exists, then the schedule is optimal. Otherwise, the algorithm continues for n iterations. There can be at most one job u with $p_u > T^*/2$. By Lemma 1, the only way that a schedule generated in the algorithm could have length $\geq (3/2)T^*$ would be if job u were the interference job. But u can be the interference job only $n-1$ times; thus at least one time, some other job $v \neq u$ with $p_v \leq T^*/2$ will be the interference job, and by Lemma 1 we have $T_A < (3/2)T^*$.

Case 2 At some point, an incorrect precedence constraint is introduced. Consider the first time that a precedence constraint is introduced, that violates the fixed optimal schedule. Immediately beforehand, we may assume that $p_{j_b} \geq T^*/2$ and $q_{j_c} \geq T^*/2$, because otherwise we have a schedule with $T_A < (3/2)T^*$. The fact that the precedence constraint is inconsistent implies that job j_b precedes job j_c in the optimal schedule. But then

$$T^* \geq r_{j_b} + p_{j_b} + p_{j_c} + q_{j_c} \geq r_{j_b} + T^*/2 + p_{j_c} + T^*/2 > T^*,$$

a contradiction. Therefore either $p_{j_b} < T^*/2$ or $q_{j_c} < T^*/2$, and that schedule satisfies $T_H < (3/2)T^*$ by Lemma 1. QED.

We note that with a slight modification the algorithm will work when there exist precedence constraints among the jobs. At the start of the algorithm, we preprocess the jobs as follows. If job i must precede job j, then

- if $r_j < r_i$ set $r_j := r_i$;
- if $q_i < q_j + p_j$ set $q_i := q_j + p_j$.

Notice that no feasible schedule is lengthened by these modifications. Furthermore, these modifications ensure that the extended Jackson's rule automatically enforces the precedence relations. To see this, notice that if i must precede j, the fact that $r_i \leq r_j$ ensures that job j does not become available for processing before job i; and the fact that $q_i \geq q_j + p_j > q_j$ ensures that if both i and j are available, Jackson's rule will schedule job i first. (If $p_j = 0$, technical modifications will ensure that ties are broken correctly.) We now modify Algorithm A as follows. Whenever an interference job b gets its release time reassigned as $r_b := r_c$, we then reassign $r_j := r_b (= r_c)$ for all jobs j that must be preceded by job b. It is straightforward to show that the proof for Case 1 of Theorem 2 above carries through. (For the purposes of the proof, if job c is forced to go after b and in actuality c must precede b, this constitutes a "bad" precedence constraint.) We shall refer to this modified algorithm as Algorithm A′.

Corollary 1 Algorithm A′ finds a feasible schedule of length $T_{A'} < (3/2)T^*$ for the delivery-time version of $1 \mid r_j,\ prec \mid L_{max}$.

Our modification of Potts' algorithm is based on running Algorithm A or A′ on the forward and inverse problem. In fact, experimental results of Grabowski, Nowicki and Zdrzalka (1986) show that this modification does improve the results obtained in practice. (Recall that the inverse problem exchanges the roles of r_j and q_j, for each job j, and reverses the direction of the precedence relation.) Generally, doing so is enough to obtain the 4/3-approximation; however, the situation in which two very long, incomparable jobs exist must be dealt with separately. Our initial solution to handling this case used the fact that the number of "interesting" schedules for this restricted case was quite small. The more elegant algorithm included here is due to Potts (1988).

Algorithm B

if there do not exist jobs u and v with $p_u,\ p_v > P/3$, **then** $(*)$

call Algorithm A′ (or A) on both the forward and inverse problem;

return the best schedule generated;

else $(\dagger)$

if the two long jobs u and v are comparable, **call** Algorithm A′ with the modification that $2n-1$ iterations of the extended Jackson's rule are performed, rather than n;

else

first add the constraint "u precedes v" and **call** Algorithm A′, with $2n-1$ iterations;

then instead add the constraint "v precedes u" and **call** Algorithm A′ with $2n-1$ iterations;

return the best schedule generated.

Theorem 2 Algorithm B generates a heuristic schedule of length T_B satisfying $T_B < (4/3)T^*$.

Proof

Case 1 At most one job, call it u, has length $p_u > P/3$, so that step $(*)$ of Algorithm B is carried out. Suppose that the solution delivered by the algorithm has length $\geq (4/3)T^*$. Note that $p_u > T^*/3$, since otherwise, by Lemma 1, on the first iteration in Algorithm A we have a schedule with the proper bound. As in the proof of Theorem 1, if in the forward or the inverse problem, all precedence constraints introduced are

consistent with some optimal schedule, then either an optimal schedule is found or in some iteration job u is not the interference job, in which case, by Lemma 1,

$$T_B < T^* + P/3 \leq (4/3)T^*.$$

Next, suppose that in both the forward and inverse runs, precedence constraints are incorrectly introduced, *i.e.*, constraints that together allow no optimal schedule. Since every optimal schedule is violated, we may fix an optimal schedule, called OPT, and consider the first iteration that a constraint violating OPT is introduced in the forward schedule. As in the proof to Theorem 1, we may assume that, just prior to introducing the "bad" precedence constraint, the interference job is the large job u and the critical job j_c satisfies $q_{j_c} \geq T^*/3$. The fact that the constraint introduced is "bad" implies that job u precedes job j_c in OPT. Next, define OPT_{inv} to be the inverse schedule to OPT, and consider the first iteration that a constraint violating OPT_{inv} is introduced in the inverse schedule. Again, u must be the interference job, and some job l_c is the critical job; and $\overline{q}_{l_c} \geq T^*/3$, by assumption, where $\overline{q}_{l_c} = r_{l_c}$ is the delivery time of job l_c in the inverse schedule. Again, because the constraint introduced is "bad" we see that job u precedes job l_c in OPT_{inv}, or equivalently, job l_c precedes u in OPT. Recalling that u precedes j_c in OPT, we have

$$\begin{aligned} T^* &\geq r_{l_c} + p_{l_c} + p_u + p_{j_c} + q_{j_c} \\ &> T^*/3 + p_{l_c} + T^*/3 + p_{j_c} + T^*/3 \\ &\geq T^* , \end{aligned}$$

a contradiction. Thus we have shown that, whenever there is a single job u with $p_u > P/3$, Algorithm B provides a schedule of length $< (4/3)T^*$.

Case 2 Two jobs u and v exist with $p_u > P/3$, $p_v > P/3$. In this case, step (†) of the algorithm is carried out. Without loss of generality, assume that there exists an optimal schedule in which job u precedes job v. Whether or not there actually exists a precedence relation between u and v, the algorithm runs Algorithm A′ with "u precedes v" enforced. We will focus on this part of the algorithm. As in Case 1, there are two possibilities. If no "bad" precedence constraint is ever introduced into the problem, then since each of the jobs u and v can each be the interference job at most $n-1$ times, either the algorithm stops with an optimal solution or in some iteration some other job j with $p_j < T^*/3$ is the interference job. In that case, by Lemma 1 $T_B < (4/3)T^*$. Next suppose that at some point a "bad" precedence constraint is introduced. We assume that, just prior to enforcing the bad constraint, either u or v is the interference job, and the critical job j_c satisfies $q_{j_c} \geq T^*/3$, since otherwise by Lemma 1 $T_B < (4/3)T^*$. If v is the interference job, then the fact that u precedes v and that the new constraint requiring v to follow j_c is bad implies

$$T^* \geq p_u + p_v + q_{j_c} > T^* .$$

Thus v cannot be the interference job. Now suppose that u is the interference job, and suppose further that either v is processed between jobs u and j_c or $v = j_c$. In either case, $q_v \geq q_{j_c}$. But then we have

$$T^* \geq p_u + p_v + q_{j_c} > T^*,$$

a contradiction. Finally, suppose that u is the interference job and that v is processed after the critical job j_c. Since u precedes j_c in the optimal schedule (because the constraint "u follows j_c" is bad),

$$T^* \geq r_u + p_u + q_{j_c}.$$

But because $\sum_{j \neq u,\ v} p_j < T^*/3$ and there is no idle time in the schedule between r_u and the processing of job j_c, we have

$$T_B < r_u + p_u + T^*/3 + q_{j_c} \leq (4/3)T^*,$$

by the previous inequality. Thus in every case with two long jobs, the schedule delivered satisfies

$$T_B < (4/3)T^*,$$

and the theorem is proved. QED.

Algorithm A requires time $O(n^2 \log n)$ to run [Potts 1980]; processing the additional precedence constraints does not dominate this time, so Algorithm A′ also run in time $O(n^2 \log n)$. It follows that Algorithm B also requires $O(n^2 \log n)$ time.

The example in Figure 1 shows that the bound of 4/3 is tight for Algorithm B. In the example given, no job j has $p_j > P/3$, and thus step $(*)$ is implemented; but when Algorithm A is run on the forward and inverse problem, a schedule of length $4Q + 3$ is obtained. The optimal schedule has length $3Q + 4$. Thus the ratio between the heuristic value and the optimal value can be made arbitrarily close to 4/3. Furthermore, the example can be modified so that step $(\dagger)$ of Algorithm B is carried out rather than step $(*)$. By setting $p_2 = p_4 = Q + 2$, we have $p_2,\ p_4 > P/3$. When job 2 is forced to precede job 4, then the same schedules as before are generated. When job 4 is forced to precede job 2, the schedules are even worse. Thus the best schedule generated will again be off by an amount arbitrarily close to $T^*/3$.

3 Two polynomial approximation schemes for $1 \mid r_j \mid L_{max}$

In this section we present two different polynomial approximation schemes for the one-machine scheduling problem without precedence constraints. The first algorithm

Release, processing, and delivery times for jobs 1 to 5

Job	rj	pj	qj
1	0	Q	0
2	1	Q	2Q+1
3	Q+1	1	2Q+2
4	2Q+1	Q	1
5	2Q+2	1	Q+1

Schedules:

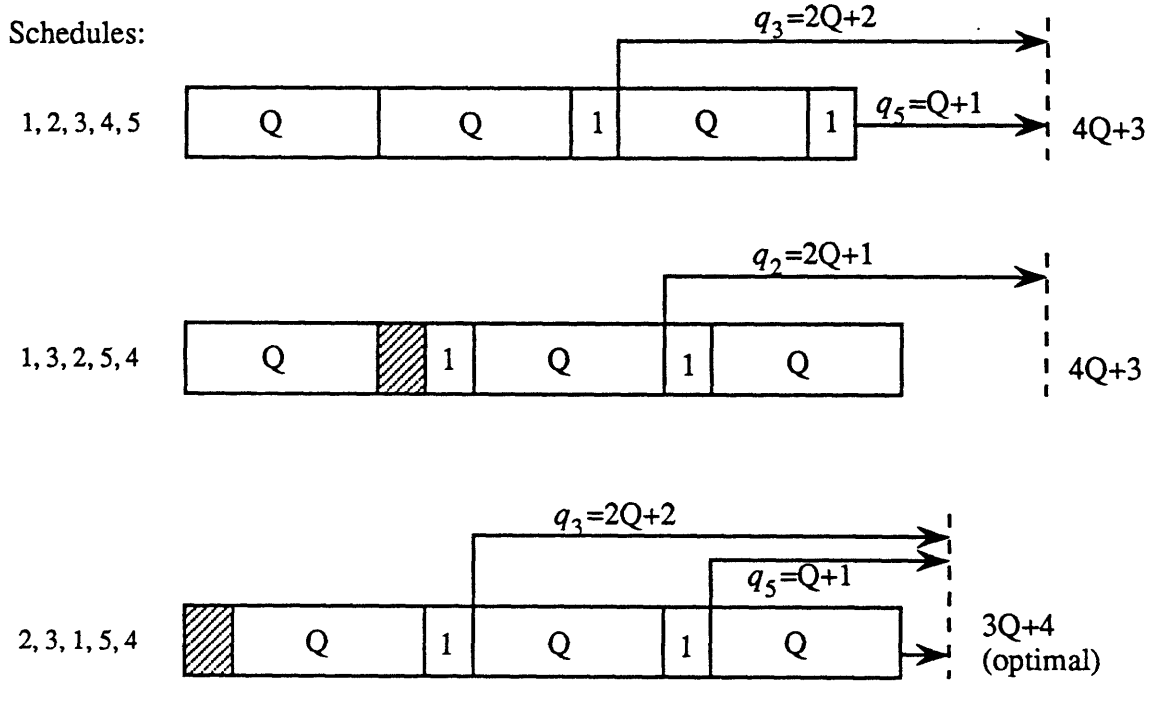


Figure 1: Example of a schedule that demonstrates that the 4/3-bound for Algorithm B is tight. The first ordering generated is $(1, 2, 3, 4, 5)$. Job 3 is the critical job, and job 2 is the interference job. After job 2 is forced to go after job 3, the new schedule becomes $(1, 3, 2, 5, 4)$. At this point there is no interference job, and the algorithm starts over with the inverse problem. Since the inverse problem is identical to the forward problem, the algorithm halts with a best schedule of length $4Q+3$. The optimal schedule, $(2, 3, 1, 5, 4)$, has length $3Q + 4$.

is based upon a polynomial-time dynamic programming formulation for the special case in which the number of release times is fixed and the processing lengths are encoded in unary. The second is based upon dividing the set of all jobs into two sets of jobs, large and small, respectively, such that the number of large jobs is fixed (relative to ϵ). The algorithm essentially tries every configuration for the large jobs and then tries various ways of fitting in the small jobs, using the extended Jackson's rule.

First, we make an observation that will simplify the analysis.

Lemma 2 Given a polynomial approximation scheme for the restricted version of $1 \mid r_j \mid L_{max}$ in which there is a fixed number of distinct release times, then there exists a polynomial approximation scheme for $1 \mid r_j \mid L_{max}$.

Proof Consider the following procedure.

1. Let $\eta = \frac{\epsilon}{2} \cdot \max_j r_j$.
2. Round down each r_j-value to the nearest multiple of η, to obtain a fixed number of release times;
3. Use the $(1 + \frac{\epsilon}{2})$-approximation algorithm on the rounded problem to obtain an ordering of the jobs;
4. Use the given ordering to schedule the jobs with their original release dates.

By rounding down in step 2 to multiples of η, we end up with $2/\epsilon + 1$ different release times, which is a fixed number. We observe that adding η to every σ_j obtained in step 3 results in a feasible schedule of length $(1 + \frac{\epsilon}{2})T^* + \eta$. Since $\max_j r_j \leq T^*$, it follows that $\eta \leq \frac{\epsilon}{2}T^*$, and thus step 4 delivers a schedule with length at most $(1 + \epsilon)T^*$. QED.

In the remainder of this section, we shall assume that we have only a fixed number of release times, denoted by κ, except in the final running time analyses of the approximation algorithms. We denote the κ possible release times by $\rho_1, \rho_2, \ldots, \rho_\kappa$; for each job j, $r_j = \rho_i$ for some $i \leq \kappa$. For notational convenience, let $\rho_{\kappa+1} = \infty$.

We present a polynomial approximation scheme that is based on a polynomial-time dynamic programming algorithm to solve $1 \mid r_j \mid L_{\max}$ when restricted to a fixed number of release times and given the processing times in unary. If there are κ release times and the sum of the processing times is $\tilde{P}$, then the algorithm runs in $O(n\kappa\tilde{P}^{2\kappa})$ time. We first show that such an algorithm yields a polynomial approximation scheme.

By Lemma 2, we need only consider instances that have only a fixed number of release times. Let $\mu = \epsilon P/n$, and round the processing times to the nearest multiple

of μ; that is, $\tilde{p}_j = \lfloor p_j/\mu \rfloor$. Note that the sum of the modified processing times,

$$\tilde{P} = \sum_{j=1}^{n} \tilde{p}_j \leq \sum_{j=1}^{n} (1 + p_j/\mu) \leq P/\mu + n = n(1 + 1/\epsilon).$$

Thus, the presumed algorithm can find an optimal schedule for the rounded problem in $O(n\kappa(n/\epsilon)^{2\kappa})$ time. Since we have rounded down the processing times, the length of this schedule is no more than the optimal schedule for the original problem, T^*. We now interpret the optimal schedule for the rounded problem as a schedule for the original problem; by rescaling and expanding the processing time of each job, we increase the length of the schedule by at most $\mu = \epsilon P/n$ with each job, and thus get a schedule of length at most $T^* + \epsilon P \leq (1 + \epsilon)T^*$.

In actuality, it is more convenient to view the algorithm that finds an optimal solution to the restricted problem as a collection of dynamic programming algorithms. Consider some schedule for a particular instance. We can subdivide the duration of the schedule into intervals in the following way: let Δ_i be defined so that $\rho_i + \Delta_i$ is the first point where a job with release time $\geq \rho_i$ is processed; this gives a partition $\{[\rho_i + \Delta_i, \rho_{i+1} + \Delta_{i+1}]\}$. Each job is processed entirely within one interval. (Note that we will allow jobs with $p_j = 0$, and such a job may fall in more than one interval, so we arbitrarily pick one of these to "contain" this job.) Finally, we shall call $\rho_i + \Delta_i$, $i = 1, \ldots, \kappa$, the *effective release times.*

Notice that there must exist an optimal schedule where the processor is active throughout the interval $[\rho_i, \rho_i + \Delta_i]$ (if $\Delta_i > 0$), so that each $\Delta_i \leq \tilde{P}$, the total processing time. Algorithm C, presented below, finds an optimal schedule, subject to the constraint that the effective release times are $\rho_i + \Delta_i$, $i = 1, \ldots \kappa$. By the previous observation, we need only run Algorithm C for each of the $\tilde{P}^\kappa$ plausible choices of effective release times to obtain an optimal schedule.

Suppose that $\rho_i + \Delta_i$, $i = 1, \ldots \kappa$, are the effective release times of an optimal schedule, and consider the set of jobs J_i scheduled in the interval $[\rho_i + \Delta_i, \rho_{i+1} + \Delta_{i+1}]$. By the definition Δ_{i+1}, all jobs in J_i have been released by ρ_i, so Jackson's rule must give an optimal ordering of J_i. This implies that if we are given the optimal Δ_i, then we need only determine the interval in which each job is processed.

In fact, this observation is the core of the dynamic programming recursion. Sort the jobs so that $q_1 \geq \ldots \geq q_n$. Let $f_\Delta(\vec{a}; j) = f_\Delta(a_1, \ldots, a_\kappa; j)$ denote the minimum completion time of a schedule for jobs $\{1, \ldots, j\}$ that uses exactly a_i total processing time in the interval $[\rho_i + \Delta_i, \rho_{i+1} + \Delta_{i+1}]$. Then it is easy to see that

$$f_\Delta(\vec{a}; j) = \min\{\max\{f_\Delta(a_1, \ldots, a_{i-1}, a_i - p_j, a_{i+1} \ldots, a_\kappa; j-1), \rho_i + \Delta_i + a_i + q_j\} \ : i \text{ such that } r_j \leq \rho_i\}$$

for all $\vec{a}$ such that $\rho_i + \Delta_i + a_i \leq \rho_{i+1} + \Delta_{i+1}$, $i = 1, \ldots, \kappa - 1$. Notice that Jackson's rule is crucial to the correctness of this recurrence, since it is based on the idea that job j must be contained in some interval, but no matter which, we may assume that it is

scheduled within that interval after all jobs previously scheduled (with longer delivery times). The basis for the recursion is easy, since we may set $f_\Delta(0,\ldots,0;0) = 0$, and all other $f_\Delta(\vec{a};0) = \infty$. This gives the following algorithm:

Algorithm C

order the jobs such that $q_1 \geq q_2 \geq \cdots \geq q_n$.

initialize the array $f_\Delta = \infty$ except $f_\Delta(0,\ldots,0;0) = 0$.

for $j := 1,\ldots,n$ **do**

for all κ-tuples $(a_1,\ldots,a_\kappa,)$ with

$$\sum_{i=1}^{\kappa} a_i = \sum_{l=1}^{j} p_l, \text{ and } \rho_i + \Delta_i + a_i \leq \rho_{i+1} + \Delta_{i+1},\ i = 1,\ldots,\kappa,$$

for all i such that $r_j \leq \rho_i$ compute the length of the optimal schedule with job j in the interval $[\rho_i+\Delta_i, \rho_{i+1}, \Delta_{i+1}]$ (using $f_\Delta(a_1,\ldots,a_{i-1}, a_i - p_j, a_{i+1},\ldots,a_\kappa; j-1)$).

choose the best interval and schedule job j last among the jobs already scheduled in that interval; let $f_\Delta(\vec{a};j)$ be the length of this extended schedule.

return $\min_{\vec{a}} f_\Delta(\vec{a};\ n)$ and the actual schedule corresponding to this minimum value.

Combining Lemma 2, the observation that a polynomial-time algorithm for the restricted case with a fixed number of release times and unary processing times yields a polynomial approximation scheme, and Algorithm C, we get the following result.

Theorem 3 There exists a polynomial approximation scheme for the delivery-time version of $1 \mid r_j \mid L_{max}$, based on Algorithm C above, which for any fixed $\epsilon > 0$ delivers a schedule of length less than $(1+\epsilon)T^*$ and requires running time

$$O(16^{1/\epsilon}(n/\epsilon)^{4/\epsilon\ +3}).$$

Proof Given $\epsilon > 0$, the approximation algorithm consists of running Algorithm C for each possible value of the vector Δ, $O(n\tilde{P}^\kappa\kappa)$ time. Since there are $O(\tilde{P}^\kappa)$ choices for Δ, the overall running time is bounded by

$$\begin{aligned} O(n\kappa\tilde{P}^{2\kappa}) &= O\left(n(1+2/\epsilon)(n+2n/\epsilon)^{2(1+2/\epsilon)}\right) \\ &= O\left(16^{1/\epsilon}(n/\epsilon)^{3+4/\epsilon}\right). \quad \text{QED.} \end{aligned}$$

We turn now to the second polynomial approximation scheme. Suppose that $\epsilon > 0$ is given. We introduce the following notation.

Definition For a given schedule, let σ_j denote the *starting time* of job j, *i.e.*, the time at which job j begins processing.

(Of course, in any feasible schedule, $\sigma_j \geq r_j$, for any job j.) Let

$$\delta = \frac{P\epsilon}{2\kappa}$$

where $P = \sum_j p_j$ and κ is the number of distinct release dates. Suppose that we divide the jobs into two sets, according to the length of their processing times,

$$A = \{j \ : \ p_j \geq \delta\} \ \text{ and } \ B = \{j \ : \ p_j < \delta\} \ .$$

(Notice that $\mid A \mid \leq P/\delta = 2\kappa/\epsilon$, a fixed number.) Now suppose that, for some fixed optimal schedule with starting times $\sigma_1^*, \ldots, \sigma_n^*$, we know which interval $[\rho_i, \ \rho_{i+1})$ contains σ_j^*, for all $j \in A$; let I_i denote the set of jobs in A that have start times in $[\rho_i, \ \rho_{i+1})$. Let

$$M_i := \sum (p_j \ : \ \sigma_j^* \in [\rho_i, \ \rho_{i+1}) \text{ and } j \in B),$$

and suppose that we know $N_i := \lceil M_i/\delta \rceil$, for $i = 1, \ldots, \kappa$; that is, we know approximately how much time in the interval $[\rho_i, \ \rho_{i+1})$ is spent processing the small jobs. Call $\{(I_i, N_i) : i = 1, \ldots, \kappa\}$ the *outline* of the optimal schedule. Consider the following procedure that uses the outline as part of its input.

Algorithm D

$J_i := \emptyset$, $i = 1, \ldots, \kappa$;

for $i := 1, \ldots, \kappa$ $(*)$

let $W_i := \{j \in B \ : \ j \notin J_k$, for any $k < i$, and $r_j \leq \rho_i \}$; let $\mid W_i \mid = m$; let $j_1, \ldots, j_m$ denote the jobs in W_i, such that $q_{j_1} \geq \cdots \geq q_{j_m}$;

if $\sum_{j \in W_i} p_j < N_i\delta$ **then** let $J_i = W_i$;

else let J_i be the minimal set $\{j_1, \ldots, j_l\}$ such that $\sum_{j \in J_i} p_j \geq N_i\delta$;

order the jobs in $I_i \cup J_i$ in order of non-increasing q_j;

schedule all jobs as follows: first schedule the jobs in $I_1 \cup J_1$ in the order determined above, then schedule the jobs in $I_2 \cup J_2$, and so on, through $I_\kappa \cup J_\kappa$.

We first prove an important dominance relation concerning the scheduling of the small jobs by Algorithm D. We wish to show that the set of small jobs that are available at iteration i of the heuristic is easier to schedule than the comparable set of "available" jobs in the optimal schedule. To capture the appropriate notion for the optimal schedule, let

$$V_i = \{j \in B : r_j \leq \rho_i \text{ and } \sigma_j^* \geq \rho_i\}.$$

For notational convenience, if $Z = \{j_1, \ldots, j_m\}$ is a set of jobs ordered so that $q_{j_1} \geq q_{j_2} \cdots \geq q_{j_m}$, and t is any value such that $t \leq \sum_{j \in Z} p_j$, then define $\alpha(Z,t) = j_k$ so that $\sum_{l=1}^{k-1} p_{j_l} < t \leq \sum_{l=1}^{k} p_{j_l}$ and let $S(Z,t) = \{j_1, \ldots j_k\}$. Note that these definitions depend on the particular ordering of the set Z, so that for definiteness, order jobs of equal delivery time by increasing release time, and break ties among jobs of equal delivery and release times by their index.

Lemma 3 For any iteration i, let W_i and V_i be the sets of available jobs in the heuristically and optimally constructed schedules. Then

(1) $\sum_{j \in W_i} p_j \leq \sum_{j \in V_i} p_j$; and

(2) for any $t \leq \sum_{j \in W_i} p_j$, $q_{\alpha(W_i,t)} \leq q_{\alpha(V_i,t)}$.

Proof The proof uses induction on i, the iteration of Algorithm D being performed. For $i = 1$, $W_i = V_i$ and thus (1) and (2) hold trivially.

Assume that (1) and (2) hold for all iterations up to $i-1$; in particular, they hold for W_{i-1} and V_{i-1}. It is useful to observe that property (1) implies that $\alpha(V_i,t)$ is well-defined in (2). There are two cases to consider. If $W_{i-1} = J_{i-1}$, Algorithm D schedules all short jobs with release time less than ρ_i in the first $i-1$ iterations, so $W_i \subseteq V_i$. Properties (1) and (2) follow immediately from this relation.

Now suppose that $W_{i-1} \neq J_{i-1}$, which implies that $\sum_{j \in J_{i-1}} p_j \geq N_{i-1}\delta$. Let us consider how W_i and V_i differ from W_{i-1} and V_{i-1}, respectively. Define $R_i = \{j \in B : r_j = \rho_i\}$. Also, let $U_{i-1} = \{j \in B : \rho_{i-1} \leq \sigma_j^* < \rho_i\}$. Then $W_{i-1} \cap R_i = \emptyset$ and $V_{i-1} \cap R_i = \emptyset$; and

$$W_i = W_{i-1} \setminus J_{i-1} \cup R_i, \text{ and } V_i = V_{i-1} \setminus U_{i-1} \cup R_i.$$

To prove the inductive claim, we will first show that $W_{i-1} \setminus J_{i-1}$ and $V_{i-1} \setminus U_{i-1}$ satisfy (1) and (2). To see that (1) holds for these sets, observe that

$$\sum_{j \in J_{i-1}} p_j \geq N_{i-1}\delta \geq M_{i-1} \geq \sum_{j \in U_{i-1}} p_j.$$

To prove (2), let t be any value such that $t \leq \sum_{j \in W_{i-1} \setminus J_{i-1}} p_j$. We need to show that $q_{\alpha(W_{i-1} \setminus J_{i-1}, t)} \leq q_{\alpha(V_{i-1} \setminus U_{i-1}, t)}$. Let

$$t' = t + \sum(p_j \ : \ j \in J_{i-1} \text{ and } q_j \geq q_{\alpha(W_{i-1} \setminus J_{i-1}, t)}), \text{ and}$$

$$t'' = t + \sum(p_j \ : \ j \in U_{i-1} \text{ and } q_j \geq q_{\alpha(V_{i-1} \setminus U_{i-1}, t)}).$$

Because J_{i-1} is taken from the beginning of the list W_{i-1},

$$\sum(p_j \ : \ j \in J_{i-1} \text{ and } q_j \geq q_{\alpha(W_{i-1} \setminus J_{i-1}, t)}) = \sum_{j \in J_{i-1}} p_j.$$

Thus it follows that $t' \geq t''$, since

$$\sum_{j \in J_{i-1}} p_j \geq N_{i-1}\delta \geq M_{i-1} \geq \sum_{j \in U_{i-1}} p_j \geq \sum(p_j \ : \ j \in U_{i-1} \text{ and } q_j \geq q_{\alpha(V_{i-1} \setminus U_{i-1}, t)}).$$

Thus if we examine t' and t'' in the inductive hypothesis with respect to W_{i-1} and V_{i-1}, then because $t' \geq t''$ we see that

$$q_{\alpha(W_{i-1} \setminus J_{i-1}, t)} = q_{\alpha(W_{i-1}, t')} \leq q_{\alpha(W_{i-1}, t'')} \leq q_{\alpha(V_{i-1}, t'')} = q_{\alpha(V_{i-1} \setminus U_{i-1}, t)},$$

where the latter inequality follows from the inductive hypothesis. Therefore, we have shown that (1) and (2) hold for $W_{i-1} \setminus J_{i-1}$ and $V_{i-1} \setminus U_{i-1}$.

We now show that we can add in the elements of R_i to both $W_{i-1} \setminus J_{i-1}$ and $V_{i-1} \setminus U_{i-1}$ while maintaining these two properties. The first property is trivial, since R_i and $V_{i-1} \setminus U_{i-1}$ are disjoint. To see that (2) holds, first consider the case where $R_i = \{j^*\}$. Recall that j^* is inserted into both $W_{i-1} \setminus J_{i-1}$ and $V_{i-1} \setminus U_{i-1}$ at a position such that any job following j^* has delivery time strictly less than j^*. Let G and H denote the sets of jobs that precede j^* in $V_{i-1} \setminus U_{i-1}$ and $W_{i-1} \setminus J_{i-1}$, respectively. Furthermore, assume that j^* is inserted between jobs g and g^*, and between jobs h and h^* in these two sets. Set

$$t = \sum_{j \in H} p_j, \tag{3}$$

and let $\beta = \alpha(V_{i-1} \setminus U_{i-1}, t)$. Because (1) and (2) hold for $W_{i-1} \setminus J_{i-1}$ and $V_{i-1} \setminus U_{i-1}$, we have $q_h \leq q_\beta$. Also, by definition,

$$t \leq \sum(p_j \ : \ j \in S(V_{i-1} \setminus U_{i-1}, t)). \tag{4}$$

On the other hand, the positioning of j^* in the two sets implies that $q_h \geq q_{j^*} > q_{g^*}$. Thus, $q_\beta > q_{g^*}$, and this implies that

$$\sum(p_j \ : \ j \in S(V_{i-1} \setminus U_{i-1}, t)) \leq \sum_{j \in G} p_j. \tag{5}$$

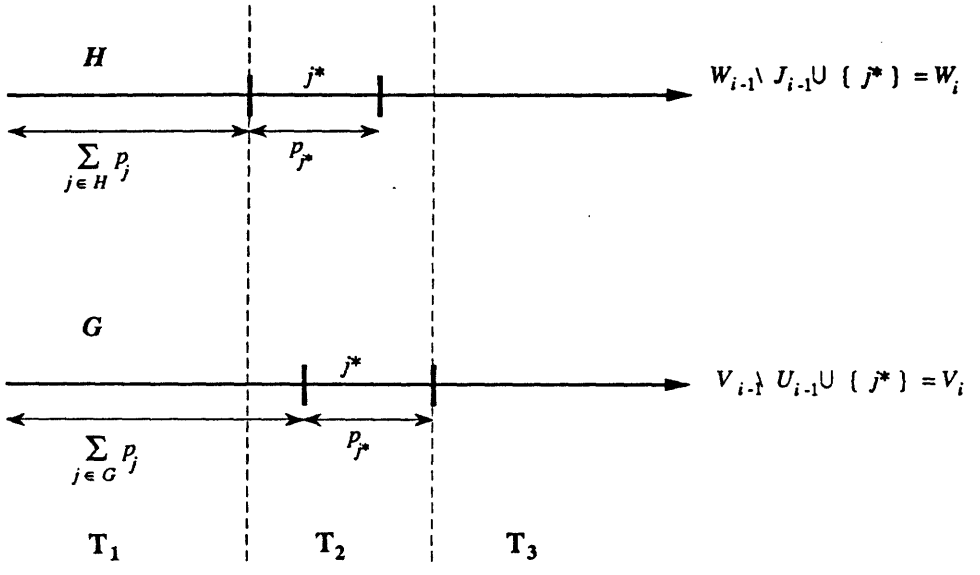


Figure 2: Domination of the heuristic scheduling of small jobs over the schedule OPT.

Combining (3), (4), and (5) gives

$$\sum_{j \in H} p_j \leq \sum_{j \in G} p_j.$$

This inequality implies that the processing requirements lists of the (ordered) sets $W_i = W_{i-1} \setminus J_{i-1} \cup \{j^*\}$ and $V_i = V_{i-1} \setminus U_{i-1} \cup \{j^*\}$ are related as shown in Figure 2. This figure depicts three distinct time periods and we prove that (2) holds by considering these three cases separately. For $t \in T_1$, the sets are unchanged by adding j^*, so (2) must still hold. Similarly, for $t \in T_3$, we see that $\alpha(W_{i-1} \setminus J_{i-1}, t - p_{j^*}) = \alpha(W_i, t)$ and $\alpha(V_{i-1} \setminus U_{i-1}, t - p_{j^*}) = \alpha(V_i, t)$, so we obtain the desired outcome. Finally, consider $t \in T_2$. In this case, either $j^* = \alpha(W_i, t)$ or j^* comes before $\alpha(W_i, t)$, in W_i; and either $j^* = \alpha(V_i, t)$ or j^* comes after $\alpha(V_i, t)$, in V_i. This implies that $q_{\alpha(V_i,t)} \geq q_{j^*} \geq q_{\alpha(W_i,t)}$.

To complete the proof, one need only notice that R_i of larger cardinality can be handled identically, merely adding one element at a time, and applying induction. QED.

Theorem 4 Given the outline of an optimal schedule, Algorithm D delivers a schedule of length $T_D < (1 + \epsilon)T^*$.

Proof We will show that every job j finishes in the heuristic schedule before time $T^* + 2\kappa\delta$. It then follows that

$$T_D \quad < \quad T^* + 2\kappa\delta$$

$$\begin{aligned} &= T^* + 2\kappa\frac{P\epsilon}{2\kappa} \\ &\leq T^* + T^*\epsilon. \end{aligned}$$

Consider any job j, and suppose that $j \in I_i \cup J_i$. Suppose that in the schedule generated by Algorithm D,

$$\tau_A = \sum(p_l \ : \ l \in I_i \text{ and } q_l \geq q_j),$$

and

$$\tau_B = \sum(p_l \ : \ l \in J_i \text{ and } q_l \geq q_j).$$

Let σ_l denote the starting time of job l in the schedule produced by Algorithm D. If we define $\xi_i = \min(\sigma_l \ : \ l \in I_i \cup J_i)$ and ζ_j to be the time at which j finishes being delivered, then

$$\zeta_j \leq \xi_i + \tau_A + \tau_B + q_j. \tag{6}$$

Next consider the jobs l in the optimal schedule with $\sigma_l^* \in [\rho_i, \ \rho_{i+1})$. By the definition of the outline, we know that all jobs in I_i are contained in this set. Furthermore, there is at least as much time from small jobs scheduled in the optimal schedule after time ρ_i as there is assigned to $\bigcup_{l=i}^{\kappa} J_l$. Thus, by Lemma 3, we know that there exists at least τ_B time from small jobs with starting times in the optimal schedule $\geq \rho_i$ and delivery times $\geq q_j$; thus, including the jobs from I_i, we see that there is at least $\tau_A + \tau_B$ total time in the optimal schedule from jobs with starting times $\geq \rho_i$ and delivery times all $\geq q_j$. Therefore we have

$$T^* \geq \rho_i + \tau_A + \tau_B + q_j. \tag{7}$$

If we can show that

$$\xi_i < \rho_i + 2\kappa\delta,$$

then by inequalities (6) and (7) we are done. But this is true by the definition of the outline. The amount of time allotted to each of the sets $I_l \cup J_l$, $l < i$, is approximately equal to that allotted to each interval $[\rho_l, \ \rho_{l+1})$; in fact, the error is bounded by 2δ per interval: δ due to the round-off error ($N_i\delta - M_i < \delta$), and δ due to overshooting $N_i\delta$ ($\sum_{l \in J_i} p_l \ - N_i\delta < \delta$). Thus

$$\xi_i < \rho_i + 2(i-1)\delta \leq \rho_i + 2\kappa\delta,$$

and we are done. QED.

Combining Lemma 2 and Theorem 4, we have the following result.

Theorem 5 There exists a polynomial approximation scheme for the scheduling problem $1 \mid r_j \mid L_{max}$ based on Algorithm D, that, given $\epsilon > 0$, delivers a schedule of length less than $(1+\epsilon)T^*$ and requires running time

$$O\left(n \log n + n(2/\epsilon)^{16/\epsilon^2 \ + \ 8/\epsilon}\right).$$

Proof Ordering all jobs with respect to length of delivery time must be done once and requires time $n \log n$. If the sets I_i and J_i are kept ordered, then the amortized work of step $(*)$ in Algorithm D is $O(n)$, and thus Algorithm D requires $O(n)$ time. Algorithm D must be called once for each potentially optimal outline of the schedule; and the number of such outlines is bounded by the number of assignments of large jobs to intervals,

$$= O(\kappa^{|A|}) = O(\kappa^{2K/(\epsilon/2)}) = O(\kappa^{4\kappa/\epsilon});$$

times the number of ways to choose $N_1, \ldots, N_\kappa$,

$$= O(\kappa^{P/\delta}), \text{ where } \delta = P(\epsilon/2)/2\kappa = P\epsilon/4\kappa,$$

$$= O(\kappa^{4\kappa/\epsilon});$$

and since $\kappa = 2/\epsilon + 1$, this product equals

$$O\left((2/\epsilon)^{8(2/\epsilon+1)/\epsilon}\right) = O\left((2/\epsilon)^{16/\epsilon^2 + 8/\epsilon}\right).$$

Thus the total time required is bounded by

$$O\left(n \log n + n(2/\epsilon)^{16/\epsilon^2 + 8/\epsilon}\right). \quad \text{QED.}$$

We make one final remark concerning Algorithm D. As in Algorithm B, the actual scheduling of all jobs could be done using the extended Jackson's rule, rather than the method given. The resulting schedule would be at least as good as that produced by the current version of Algorithm D.

4 An NP-Completeness Result

We conclude this paper with a negative result on $1 \mid r_j,\ prec \mid L_{max}$, that indicates that it may not be easy to find an ϵ-approximation for this problem. Consider the following special case of the problem.

RESTRICTED MAXIMUM LATENESS. Input: two release dates, R_1 and R_2, two processing times, P_1 and P_2, and two delivery times Q_1 and Q_2; a set of n jobs with $r_j = R_1$ or R_2, $p_j = P_1$ or P_2, and $q_j = Q_1$ or Q_2, for $j = 1, \ldots, n$; precedence relations among the jobs; and a deadline, D.
Question: does there exist a 1-machine schedule satisfying release times and precedence constraints, such that every job is delivered by time D?

Theorem 6 RESTRICTED MAXIMUM LATENESS is strongly NP-hard.

Proof The reduction is from CLIQUE.

CLIQUE. Input: $G = (V,\ E)$, $K <\mid V \mid$.
Question: Does G contain a complete subgraph $H = (V',\ E')$ with $\mid V' \mid \geq K$?

Given an instance of CLIQUE, we construct an instance of RESTRICTED MAXIMUM LATENESS as follows. There will be $\mid V \mid + \mid E \mid +1$ jobs altogether:

- $\mid V \mid$ vertex jobs with $p_v = \frac{K(K-1)}{2} + 1$; $r_v = 0,\ q_v = 0$;
- $\mid E \mid$ edge jobs with $p_e = 1$; $r_e = 0, q_e = 0$;
- One dummy job, job 0, with $p_0 = 1$; $r_0 = (K+1)\left(\frac{K(K-1)}{2} + 1\right) - 1$; $q_0 = (\mid V \mid -K-1)\left(\frac{K(K-1)}{2} + 1\right) + \mid E \mid +1$.

Thus we have $R_1 = 0$, $R_2 = r_0$; $P_1 = \frac{K(K-1)}{2} + 1$, $P_2 = 1$; and $Q_1 = 0$, $Q_2 = q_0$.
There exist precedence constraints among the jobs as follows:
For all $e = \{u,\ v\} \in E$, vertex jobs u and v must precede edge job e, and the deadline is set to

$$D = \mid V \mid \left(\frac{K(K-1)}{2} + 1\right) + \mid E \mid +1\ .$$

We need to show that the given instance of CLIQUE is a "yes"-instance if and only if the given instance of RESTRICTED MAXIMUM LATENESS is a "yes"-instance. First, we observe that job 0 must be processed at exactly time r_0, if the deadline D is to be met. Thus job 0 essentially partitions the processing time into two intervals. Furthermore, if D is to be met then all jobs must be processed without any idle time, starting at time 0. Provided that job 0 begins at time r_0, the only way to avoid idle time in the schedule is to process K vertex jobs followed by $K(K-1)/2$ edge jobs, before job 0. But this is possible if and only if a K-clique exists in G, because of the precedence constraints (see Figure 3). Furthermore, if a K-clique does exist in G, then a feasible schedule may be completed by processing the remaining vertex jobs followed by the remaining edge jobs, after job 0. Thus we have shown that the reduction is valid. Since this reduction is clearly polynomial-time, and since the magnitudes of the numbers involved are polynomial in K, we have shown that RESTRICTED MAXIMUM LATENESS is indeed strongly NP-complete. QED.

Notice that in addition to all of the other restrictions allowed, the reduction requires a precedence-tree of depth only two. Even so, the existence of a polynomial approximation scheme for $1 \mid r_j,\ prec \mid L_{max}$ is not ruled out.

Acknowledgements The authors are grateful to Chris Potts, for allowing them to include his more elegant version of the 4/3-approximation algorithm.

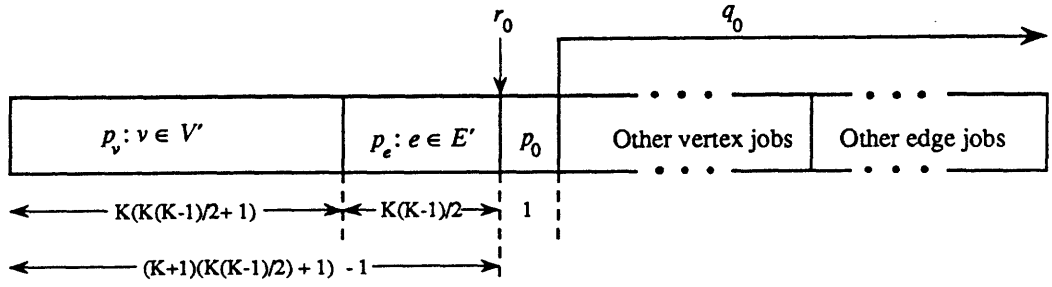


Figure 3: This diagram illustrates the only way in which a feasible schedule satisfying the bound D may be implemented. Such a schedule is possible if and only if a clique $H = (V', E')$ of size K exists.

5 References

Garey, M.R. and D.S. Johnson. (1978) Strong NP-completeness results: motivation, examples and implications, *J. Assoc. Comput. Mach. 25*, 499-508.

Garey, M. R., D. S. Johnson, B. B. Simons, and R. E. Tarjan. (1981) Scheduling unit-time tasks with arbitrary release times and deadlines, *SIAM J. Comput.*, Vol. 10, No. 2, 256-69.

Grabowski, J., E. Nowicki, and S. Zdrzalka. (1986) A block approach for single-machine scheduling with release dates and due dates, *European J. Oper. Res. 26*, 278-285.

Graham, R. L., E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey, *Ann. Discret Math. 5*, 287-326.

Jackson, J. R. (1955) Scheduling a production line to minimize maximum tardiness, Research Report 43, Management Science Research Project, UCLA.

Kise, H., T. Ibaraki, and H. Mine. (1979) Performance Analysis of Six Approximation Algorithms for the One-Machine Maximum Lateness Scheduling Problem with Ready Times, *J. Opns. Res. Soc. of Japan*, Vol. 22, No. 3, 205-24.

Lenstra, J. K., A. H. G. Rinnooy Kan, and P. Brucker. (1977) Complexity of Machine Scheduling Problems. *Ann. Discrete Math. 1*, 343-62.

Potts, C. N. (1980) Analysis of a Heuristic for One Machine Sequencing with Release Dates and Delivery Times, *Operations Research*, Vol. 28, No. 6, 1436-41.

Potts, C.N. (1988) Private communication.

Simons, B. (1978) A fast algorithm for single processor scheduling, *IEEE 19th Symp. on Foundations of Comp. Sci.*, 246-52.