A Linear Time Approximation Scheme for the Single Machine Scheduling Problem with Controllable Processing Times

Monaldo Mastrolilli

IDSIA, Galleria 2, 6928 Manno, Switzerland

Abstract

In a scheduling problem with controllable processing times the job processing time can be compressed through incurring an additional cost. We consider the problem of scheduling $n$ jobs on a single machine with controllable processing times. Each job has a release date when it becomes available for processing, and, after completing its processing, requires an additional delivery time. Feasible schedules are further restricted by job precedence constraints. We develop a polynomial time approximation scheme whose running time depends only linearly on the input size. This improves and generalizes the previous $(3/2 + \varepsilon)$-approximation algorithm by Zdrzalka. Moreover, this linear complexity bound gives a substantial improvement of the best previously known polynomial bound obtained by Hall and Shmoys for the special case without controllable processing times.

Key words: scheduling, approximation algorithms.

1 Introduction

In this paper we consider the following single machine scheduling problem. A set, $J = \{J_1, \ldots, J_n\}$, of $n$ jobs is to be processed without interruption on a single machine. For each job $J_j$ there is an interval $[\ell_j, u_j]$, $0 \leq \ell_j \leq u_j$, specifying its possible processing times. The cost for processing job $J_j$ in time $\ell_j$ is $c_j^\ell \geq 0$ and for processing it in time $u_j$ the cost is $c_j^u \geq 0$. For any value $\delta_j \in [0, 1]$ the cost for processing job $J_j$ in time $p_j(\delta_j) = \delta_j \ell_j + (1 - \delta_j)u_j$ is

* A preliminary version of this paper appeared in the Proceedings of FST&TCS’01 and SWAT’02.

Email address: monaldo@idsia.ch (Monaldo Mastrolilli).
\( c_j(\delta_j) = \delta_j c_j^f + (1 - \delta_j) c_j^u \), where \( \delta_j \) is the compression parameter. Additionally, each job \( J_j \) has a release date \( r_j \geq 0 \) when it first becomes available for processing and, after completing its processing on the machine, requires an additional delivery time \( q_j \geq 0 \); if \( s_j \geq r_j \) denotes the time \( J_j \) starts processing, then it has been delivered at time \( s_j + p_j(\delta_j) + q_j \), for compression parameter \( \delta_j \). Delivery is a non-bottleneck activity, in that all jobs may be simultaneously delivered. Feasible schedules are further restricted by job precedence constraints given by the partial order \( \prec \), where \( J_j \prec J_k \) means that job \( J_k \) must be processed after job \( J_j \). Let \( \eta \) be a permutation of the set \( J \) that is consistent with the precedence constraints; \( \eta \) denotes a processing order of jobs. Denote by \( Q(\delta, \eta) \) the (earliest) maximum delivery time of all the jobs for compression parameters \( \delta = (\delta_1, ..., \delta_n) \) and processing order \( \eta \). The total cost of compression parameters \( \delta \) is equal to \( \sum_{j \in J} c_j(\delta_j) \), and the total scheduling cost for compression parameters \( \delta \) and processing order \( \eta \) is defined as

\[
K(\delta, \eta) = Q(\delta, \eta) + \sum_{j \in J} c_j(\delta_j).
\]

The problem is to find \( \delta \) and \( \eta \) minimizing \( K(\delta, \eta) \). Since the special case with fixed processing times and without precedence constraints is strongly NP-hard [8], the stated problem is also strongly NP-hard.

When all processing times are fixed (\( \ell_j = u_j \)), 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, \( L_j = s_j + p_j - d_j \), of any job \( J_j \). When considering the performance of approximation algorithms, the delivery time model, which assumes \( d_j \leq 0 \), is preferable (see [3,5]). Without this restriction, results are likely to be elusive, since the problem of determining whether \( L_{\text{max}} \leq 0 \) is NP-complete. Because of this equivalence, we shall denote the problem with fixed processing times as \( 1|r_j, \text{prec}|L_{\text{max}} \) (and \( 1|r_j|L_{\text{max}} \) when there are no precedence constraints), using the notation of Graham et al. [2].

As these scheduling problems are known to be hard to solve optimally, most research focuses on giving polynomial-time approximation algorithms that produce a solution close to the optimal one. Ideally, one hopes to obtain a family of polynomial algorithms such that for any given \( \varepsilon > 0 \) the corresponding algorithm is guaranteed to produce a solution with a value within a factor of \( (1 + \varepsilon) \) of the optimum value; such a family is called a polynomial time approximation scheme (PTAS).

Hall and Shmoys [3,5] propose two polynomial time approximation schemes for problem \( 1|r_j|L_{\text{max}} \), the running time of which are \( O\left( n/\varepsilon \right)^{O(1/\varepsilon^2)} \) and \( O\left( n \log n + n (1/\varepsilon)^{O(1/\varepsilon^2)} \right) \). For the corresponding problem with controllable processing times, Zdrzalka [9] gives a polynomial time approximation algorithm with a worst-case ratio of \( 3/2 + \varepsilon \), where \( \varepsilon > 0 \) can be made arbitrarily
small. When the precedence constraints are imposed and the job processing
times are fixed $1|r_j, prec|L_{\text{max}}$, Hall and Shmoys [4] give a PTAS. This
consists of executing, for $\log_2 \Delta$ times, an extended version of their previous PTAS
for $1|r_j, L_{\text{max}}$, where $\Delta$ denotes an upper bound on the optimal value of any
given instance whose data are assumed to be integral. This polynomial running
time should be contrasted with the time complexity of their result for problem
$1|r_j/L_{\text{max}}$, where they were able to achieve a considerably better time. To some
extent, this is not surprising, since precedence constraints add a substantial
degree of difficulty, and one important area of research in scheduling theory
has been to study under what conditions a precedence-constrained problem is
computationally harder than its counterpart with independent jobs.

In this paper we provide the first known PTAS for problem $1|r_j, prec|L_{\text{max}}$ with
controllable processing times that runs in linear time (the hidden constant
depends exponentially on $1/\varepsilon$). This improves and generalizes all the previous
results [3–5,9]. The linear complexity bound is a substantial improvement
compared to the above mentioned result. Note that the time complexity of
this PTAS is the best possible with respect to the number of jobs. Moreover,
the existence of a PTAS whose running time is also polynomial in $1/\varepsilon$ for a
strongly NP-hard problem would imply P=NP [1].

In contrast to the approximation schemes introduced in [3–5], our algorithm
(that works also for the general variant with controllable processing times) is
more general, simpler and even faster. We start partitioning jobs into a con-
stant\(^1\) number of subsets (Section 2.1). We show that the precedence graph
can be simplified into a more primitive graph (Section 2.2). This simplifica-
tion depends on the desired precision $\varepsilon$ of approximation; the closer $\varepsilon$ is to
zero, the closer the modified graph will resemble the original one. Then, jobs
belonging to the same subset are grouped together into a single compact job
to obtain a smaller instance of constant size. The processing times and cost
of these compact jobs are constrained to belong to a constant sized set of
values; this set is computed by solving a constant number of linear programs
(Section 2.3). After this, a non-feasible solution is constructed by allowing pre-
emption. A feasible solution is obtained by processing preempted jobs without
interruptions (Section 3).

2 Simplifying the Input

We start by transforming any given instance into a standard form. Let $d_j = \min\{\ell_j + c_j^r, u_j + c_j^u\}$, $D = \sum_{j=1}^n d_j$, $r_{\text{max}} = \max_j r_j$ and $q_{\text{max}} = \max_j q_j$.

\(^1\) Throughout the paper we assume that $\varepsilon$, i.e. the desired approximation precision,
is a constant and when we say “constant”, that constant may depend on $\varepsilon$. 

3
Moreover, let \( OPT \) denote the optimal solution value of the given instance.

**Lemma 1** Without loss of generality, we can assume that the following holds:

- \( 1 \leq OPT \leq 3; \)
- \( \max\{D, r_{\max}, q_{\max}\} \leq 1; \)
- \( 0 \leq \ell_j \leq u_j \leq 3 \) and \( 0 \leq c^u_j \leq c^\ell_j \leq 3. \)

**Proof.** We begin by bounding the largest number occurring in any given instance. Let \( LB = \max\{D, r_{\max}, q_{\max}\} \), we claim that \( LB \leq OPT \leq 3LB \). Indeed, since \( D \), \( r_{\max} \) and \( q_{\max} \) are lower bounds for \( OPT \), \( LB \) is also a lower bound for \( OPT \). We show that \( 3LB \) is an upper bound for \( OPT \) by exhibiting a schedule with value at most \( 3LB \). Starting from time \( r_{\max} \) all jobs have been released and they can be scheduled one after the other in any fixed ordering of the jobs that is consistent with the precedence relation; this can be obtained by topologically sorting the precedence graph. Then every job can be completed by time \( r_{\max} + D \) and the total scheduling cost is bounded by \( r_{\max} + D + q_{\max} \leq 3LB \). By dividing every \( \ell_j, u_j, c^\ell_j, r_j \) and \( q_j \) by \( LB \), we may (and will) assume, without loss of generality, that \( r_{\max}, q_{\max} \leq 1, LB = 1 \) and \( 1 \leq OPT \leq 3. \)

Furthermore, we can assume, without loss of generality, that \( 0 \leq \ell_j \leq u_j \leq 3 \) and \( 0 \leq c^u_j \leq c^\ell_j \leq 3 \), for all jobs \( J_j \): if \( c^\ell_j < c^u_j \), then there exists an optimal solution with \( \delta_j = 1 \) (i.e., the processing time of job \( J_j \) is equal to \( \ell_j \)). Then, we can reset \( c^u_j := c^\ell_j \) without affecting the value of the objective function of any feasible schedule. Moreover, in any optimal solution the processing time of any job cannot be larger than 3; therefore, if \( u_j > 3 \) we can reduce, without loss of generality, the interval of possible processing times and get an equivalent instance by setting \( c^u_j := \frac{u_j - 3}{3}(c^\ell_j - c^u_j) + c^u_j \) and \( u_j = 3. \) Similar arguments hold if \( c^\ell_j > 3. \) ■

Following Lageweg, Lenstra and Rinnoy Kan [6], if \( J_j \prec J_k \) and \( r_j > r_k \), then we can reset \( r_k := r_j \) and each feasible schedule will remain feasible. Similarly, if \( q_j < q_k \) then we can reset \( q_j := q_k \) without changing the objective function value of any feasible schedule. Thus, by repeatedly applying these updates we can always obtain an equivalent instance that satisfies

\[
J_j \prec J_k \implies (r_j \leq r_k \text{ and } q_j \geq q_k) \tag{1}
\]

Such a resetting requires \( O(\ell) \) time, where \( \ell \) denotes the number of precedence constraints. Thus in the following we assume that (1) holds.

A technique used by Hall and Shmoys [3] allows us to deal with only a constant number of release dates and delivery times. The idea is to round each release and delivery time down to the nearest multiple of \( i\varepsilon \), for \( i \in \mathbb{N} \). Since \( r_{\max} \leq 1 \), the number of different release dates and delivery times is now bounded by
1/ε + 1. Clearly, the optimal value of this transformed instance cannot be greater than OPT. Every feasible solution for the modified instance can be transformed into a feasible solution for the original instance just by adding ε to each job’s starting time, and reintroducing the original delivery times. It is easy to see that the solution value may increase by at most 2ε.

Therefore, we will assume henceforth that the input instance has a constant number of release dates and delivery times, and that condition (1) holds. We shall refer to this instance as I. By the previous arguments, OPT ≥ OPT(I), where OPT(I) denotes the optimal value for instance I.

2.1 Partitioning the Set of Jobs

Partition the set of jobs in two subsets:

\[ L = \{ J_j : d_j > \varepsilon^2 \}, \]
\[ S = \{ J_j : d_j \leq \varepsilon^2 \}. \]

Let us say that L is the set of large jobs, while S the set of small jobs. Observe that the number of large jobs is bounded by 1/ε² by Lemma 1. We further partition the set S of small jobs as follows. For each small job J_j ∈ S consider the following three subsets of L:

\[ Pre(j) = \{ J_i \in L : J_i \prec J_j \}, \]
\[ Suc(j) = \{ J_i \in L : J_j \prec J_i \}, \]
\[ Free(j) = L - (Pre(j) \cup Suc(j)). \]

Let us say that T(j) = \{Pre(j), Suc(j), Free(j)\} represents a 3-partition of set L with respect to job J_j. We form the set T = \bigcup_j T(j) which is the set of all distinct 3-partitions. We will index the items in set T by T_1, ..., T_τ, where τ = |T|. The number τ of distinct 3-partitions is clearly bounded by the number of small jobs and by 3^{|L|} ≤ 3^{1/ε²}, therefore τ ≤ \min \{n, 3^{1/ε²}\}. Now, we define the execution profile of a small job J_j to be a 3-tuple \( < i_1, i_2, i_3 > \) such that \( r_j = \varepsilon \cdot i_1, q_j = \varepsilon \cdot i_2 \) and \( T(j) = T_{i_3}, \) where \( i_1, i_2 = 0, 1, ..., 1/\varepsilon \) and \( i_3 = 1, ..., \tau. \) For any given instance, the number of distinct execution profiles is clearly bounded by the number of jobs and, by the previous arguments, cannot be larger than \( (1 + 1/\varepsilon)^2 \tau. \)

**Corollary 2** The number π of distinct execution profiles is bounded by π ≤ \min \{n, 3^{1/ε²}(1 + 1/ε)^2\}. 

5
Partition the set $S$ of small jobs into $\pi$ subsets, $S_1, S_2, \ldots, S_\pi$, such that jobs belonging to the same subset have the same execution profile. Clearly, $S = S_1 \cup S_2 \ldots \cup S_\pi$ and $S_h \cap S_i = \emptyset$, for $i \neq h$. We illustrate the above by the following example.

**Example 1** Consider the precedence structure given by the graph in Figure 1. Shaded nodes represent large jobs, while the others denote small jobs. Assume that $r_3 = r_4 = r_9$ and $q_3 = q_4 = q_9$. Since $Pre(3) = Pre(4) = Pre(9) = \{J_1, J_2\}$ and $Suc(3) = Suc(4) = Suc(9) = \{J_7\}$, jobs $J_3, J_4$ and $J_9$ establish the same 3-partition of set $L$ and therefore $T(3) = T(4) = T(9)$. Moreover, jobs $J_3, J_4$ and $J_9$ have the same execution profile since they have equal release dates and delivery times. Therefore, the set $S = \{J_3, J_4, J_6, J_8, J_9\}$ of small jobs is partitioned into 3 subsets $S_1 = \{J_3, J_4, J_9\}$, $S_2 = \{J_6\}$ and $S_3 = \{J_8\}$.

### 2.2 Adding New Precedences

Let us say that job $J_h$ is a *neighbor* of set $S_i$ ($i = 1, \ldots, \pi$) if:

- $J_h$ is a small job;
- $J_h \notin S_i$;
- there exists a precedence relation between job $J_h$ and some job in $S_i$.

Moreover, we say that $J_h$ is a *front-neighbor* (*back-neighbor*) of $S_i$ if $J_h$ is a neighbor of $S_i$ and there is a job $J_j \in S_i$ such that $J_j \prec J_h$ ($J_h \prec J_j$).

Let $n_i = |S_i|$ ($i = 1, \ldots, \pi$), and let $(J_{1,i}, \ldots, J_{n_i,i})$ denote any fixed and complete ordering of the jobs from $S_i$ that is consistent with the precedence relation. In the rest of this section we restrict the problem such that the jobs from $S_i$ are processed according to this fixed ordering. Furthermore, every back-neighbor (front-neighbor) $J_h$ of $S_i$ ($i = 1, \ldots, \pi$) must be processed before (after) every job from $S_i$. This can be accomplished by adding a directed arc from $J_{j,i}$ to $J_{j+1,i}$, for $j = 1, \ldots, n_i - 1$, and by adding a directed arc from $J_h$ to $J_{1,i}$, if $J_h$ is a back-neighbor of $S_i$, or an arc from $J_{n_i,i}$ to $J_h$, if $J_h$ is a front-neighbor.

![Fig. 1. Graph of Example 1](image-url)
We will see later that this transformation does not considerably affect the optimal solution value. The idea is that the jobs from $S_i$ are “similar” and small. So fixing an arbitrary order among them and updating precedence constraints appropriately does not deteriorate too much the solution quality.

Finally, note that the resulting precedence graph is without cycles. Indeed, it can be easily checked that a neighbor $J_h$ of $S_i$ (for $i = 1, \ldots, \pi$) cannot be simultaneously a front neighbor and a back-neighbor of $S_i$. The number of added arcs can be bounded by $n + \ell$ (recall that $\ell$ denotes the number of precedence constraints of the input instance). The above is illustrated by the following example.

**Example 2** Consider Example 1. Observe that $(J_3, J_4, J_9)$ is an ordering of the jobs from $S_1$ that is consistent with the precedence relation. Job $J_8$ is a back-neighbor of $S_1$, while job $J_6$ is a front-neighbor of $S_1$. The new precedence structure is given by the graph in Figure 2, where the new added arcs are emphasized.

We observe that condition (1) is valid also after these changes. Indeed, if $J_h$ is a back-neighbor of $S_i$ then there is a job $J_j \in S_i$ such that $J_h \prec J_j$, and therefore by condition (1) we have $r_h \leq r_j$ and $q_h \geq q_j$. But, the jobs from $S_i$ have the same release dates and delivery times, therefore $r_h \leq r_j$ and $q_h \geq q_j$ for each $J_j \in S_i$. It follows that if we restrict $J_h$ to be processed before the jobs from $S_i$, condition (1) is still valid. Similar arguments hold if $J_h$ is a front-neighbor. Moreover, all the jobs from $S_i$ have the same release dates and delivery times, therefore condition (1) is still satisfied, if we restrict these jobs to be processed in any fixed ordering that is consistent with the precedence relation.

2.3 Compact Representation of Job Subsets

Consider set $S_i$ (for $i = 1, \ldots, \pi$). Note that the number of jobs in $S_i$ may be $\Theta(n)$. We replace the jobs from $S_i$ with one compact job $J_i^\#$. Job $J_i^\#$ has
the following linear program

\[
\text{min} \quad \sum_{j \in S}(\delta_j c^d_j + (1 - \delta_j)c^u_j)
\]
\[
\text{s.t.} \quad \sum_{j \in S}(\delta_j \ell_j + (1 - \delta_j)u_j) \leq p
\]
\[
0 \leq \delta_j \leq 1 \quad \text{for all } j \in S, J_j \in S_i
\]

By setting \(\delta_j = 1 - x_j\), it is easy to see that an optimal solution for \(LP(S_i, p)\) can be obtained by solving the following linear program:

\[
\text{max} \quad \sum_{j \in S}(c^d_j - c^u_j)x_j
\]
\[
\text{s.t.} \quad \sum_{j \in S}(u_j - \ell_j)x_j \leq p - \sum_{j \in S} \ell_j
\]
\[
0 \leq x_j \leq 1 \quad \text{for all } j \in S, J_j \in S_i
\]

Note that \(p - \sum_{j=1}^n \ell_j\) is non-negative, since \(p \in P_i\) and the smallest value of \(P_i\) cannot be smaller than \(\sum_{j=1}^n \ell_j\). The previous linear program corresponds to the classical knapsack problem with relaxed integrality constraints. By partially sorting jobs in nonincreasing \((c^d_j - c^u_j)/(u_j - \ell_j)\) ratio order, the set \(\{LP(S_i, p) : p \in P_i\}\) of \(O(1/\varepsilon^3)\) many problems can be solved in \(O(|S| \log \frac{1}{\varepsilon} + (1/\varepsilon^3) \log \frac{1}{\varepsilon})\) time by employing a median-finding routine (we refer to Lawler [7] for details). For each value \(p \in P_i\), the corresponding cost value \(C_i(p)\) is equal to the optimal solution value of \(LP(S_i, p)\) rounded up to the nearest value in set \(V_S\). It follows that the number of alternative pairs of processing times and costs for each compact job \(J_i\) is bounded by the cardinality of set \(P_i\). Furthermore, since \(\sum_{i=1}^\pi |S_i| \leq n\), it is easy to check that the amortized total time to compute the processing requirements of all compact jobs is \(O(n(1/\varepsilon^3) \log \frac{1}{\varepsilon})\). Therefore, each set \(S_i\) is transformed into one compact job
Fig. 3. Graph of Example 3

$J_i$ with $O(1/\epsilon^3)$ alternative pairs of costs and processing times. We use $S^#$ to denote the set of compact jobs.

**Example 3** Consider Example 2. We group jobs $J_3$, $J_4$ and $J_9$ together and get a new instance whose precedence structure is given by the graph in Figure 3.

Now, let us consider the modified instance as described so far and turn our attention to the set $L$ of large jobs. We map each large job $J_j \in L$ to a new job $J_j^#$ which has the same release date, delivery time and set of predecessors and successors as job $J_j$, but a more restricted set of possible processing times and costs. This restricted set is chosen such that we can still obtain a near-optimal solution, so the restriction does not significantly deteriorate the objective function value. More precisely, let $A_j$ (and $B_j$) be the value obtained by rounding $\ell_j$ (and $u_j$) up to the nearest value from set $V_L = \{\epsilon^3, \epsilon^3(1 + \epsilon), \epsilon^3(1 + \epsilon)^2, \ldots, 3\}$. The possible processing times for $J_j^#$ are specified by set $P_j := V_L \cap [\ell_j, u_j]$. For each value $p \in P_j$, the corresponding cost value $C_j(p)$ is obtained by rounding up to the nearest value of set $V_L$ the cost of job $J_j$ when its processing time is $p$. We use $L^#$ to denote the set of jobs obtained by transforming jobs from $L$ as described so far.

Let $I^#$ denote this modified instance. We observe that $I^#$ can be computed in $O(n(1/\epsilon^3) \log \frac{1}{\epsilon} + 2^{O(1/\epsilon^2)})$ time: the time required to partition the set of jobs into $\pi$ subsets can be bounded by $O(n + \ell + 2^{O(1/\epsilon^2)})$; $O(n + \ell)$ is the time to add new precedences; $O(n(1/\epsilon^3) \log \frac{1}{\epsilon})$ is the time to compute the alternative pairs of costs and processing times. Moreover, this new instance has at most $\nu = 3^{1/\epsilon^2} \cdot (1 + 1/\epsilon)^2 + 1/\epsilon^2$ jobs; each job has a constant number of alternative pairs of costs and processing times. Now let us focus on $I^#$ and consider the problem of finding the schedule for $I^#$ with the minimum scheduling cost such that compact jobs can be preempted, while interruption is not allowed for jobs from $L^#$.

Let $OPT_{relax}(I^#)$ denote the optimal solution value of $I^#$ when compact job can be preempted. The following lemma shows that $OPT_{relax}(I^#)$ is close to $OPT(I)$. Moreover, it gives a bound on the number of preempted jobs. (Since the proof of the following lemma is quite technical it is deferred to Section 4.)
Lemma 3 For any fixed \( \varepsilon > 0 \), it is possible to compute in constant time an optimal solution for \( I^\# \) with at most \( 1/\varepsilon \) preempted compact jobs. Moreover, 
\[
OPT_{relax}(I^\#) \leq (1 + 4\varepsilon)OPT(I).
\]

3 Generating a Feasible Solution

In this subsection we show how to transform the optimal solution \( SOL^\# \) for instance \( I^\# \) into a \((1 + O(\varepsilon))\)-approximate solution for instance \( I \).

First, replace the jobs from \( L^\# \) with the corresponding large jobs. Let \( p_j^\# \) and \( c_j^\# \) denote the processing time and cost, respectively, of job \( J_j^\# \in L^\# \) according to solution \( SOL^\# \). Then it is easy to check that the corresponding job \( J_j \in L \) can be processed in time and cost at most \( p_j^\# \) and \( c_j^\# \), respectively.

Second, we replace each compact job \( J_i^\# \) with the corresponding small jobs from set \( S_i \) as follows. Remove job \( J_i^\# \), this clearly creates gaps into the schedule. Then, fill in the gaps by inserting the small jobs from set \( S_i \) according to any fixed ordering that is consistent with the precedence relation, and by allowing preemption; the processing time and cost of these small jobs are chosen according to the optimal solution of \( LP(S_i, p_i^\#) \) (see Subsection 2.3), where \( p_i^\# \) denotes the processing time of job \( J_i^\# \) according to solution \( SOL^\# \). (Recall that the optimal solution of \( LP(S_i, p_i^\#) \) chooses the processing requirements of jobs from \( S_i \) such that the sum of processing times is at most \( p_i^\# \) and the sum of costs is minimum.) However, these replacements do not yield a feasible solution for \( I \), since there may be a set \( M \) of preempted small jobs. By Lemma 3, we have that the number of preempted small jobs is at most \( 1/\varepsilon \).

For each \( J_j \in M \) let \( s_j \) be the time at which job \( J_j \) starts in the preemptive schedule. Remove each \( J_j \in M \) and schedule \( J_j \) without interruption at time \( s_j \) with processing time \( p_j \) and cost \( c_j \), where \( p_j + c_j = d_j \). It is easy to see that the maximum delivery time may increase by at most \( \sum_{J_j \in M} p_j \) and the cost by at most \( \sum_{J_j \in M} c_j \). Therefore, the solution value is increased by at most \( \sum_{J_j \in M} d_j \leq |M|\varepsilon \leq \varepsilon \cdot OPT(I) \), since \( |M| \leq 1/\varepsilon \), \( M \subseteq S \) and 
\[
S = \{ J_j : d_j \leq \varepsilon^2 \}.
\]

Finally, we have already observed that every feasible solution for the modified instance with only a constant number of release dates and delivery times can be transformed into a feasible solution for the original instance by simply delaying each job starting time by at most \( \varepsilon \), and reintroducing the original delivery times. This may increase the value of the solution by at most \( 2\varepsilon \). Therefore, by Lemma 3, the value of the returned solution is at most \( (1 + 7\varepsilon) \cdot OPT(I) \), that confirms that this construction does in fact yield an \((1 + O(\varepsilon))\)-approximate solution of \( I \). To conclude, we have shown that problem \( 1|r, prec|L_{\max} \) with controllable processing times admits a PTAS.
(S-1) Initialize the solution $BestFound$ to be the empty solution and set the corresponding value $V$ to infinity.

(S-2) For each $(\tau_1, \ldots, \tau_\nu) \in P_1 \times \ldots \times P_\nu$ and for each $(\rho_1, \ldots, \rho_\lambda) \in R_1 \times \ldots \times R_\lambda$ such that $\rho_j \geq \max_{J^\#h \in \text{Pre}(j)} \rho_h$ (for $j = 1, \ldots, \lambda$):

(S-2.1) Modify instance $I^#$ to get instance $I^\diamond$ with release dates $r^\diamond_j$, delivery times $q^\diamond_j$, processing times $p^\diamond_j$ and costs $c^\diamond_j$ fixed to the following values:

- $p^\diamond_j := \tau_j,$
- $c^\diamond_j := C_j(\tau_j),$  
- $r^\diamond_j := \rho_j,$  
- $r^\diamond_j := \max\{r^\#: J^\#_h \in \text{Pre}(j) r^\diamond_h\},$ for $J^\#_j \in L^\#,$
- $q^\diamond_j := \max\{q^\#: J^\#_h \in \text{Pre}(j) q^\diamond_h\},$ for $J^\#_j \in S^\#,$
- $q^\diamond_j := \max\{q^\#: J^\#_h \in \text{Pre}(j) q^\diamond_h\},$ for $J^\#_j \in L^\#.$

(S-2.2) Apply the preemptive extended Jackson’s rule to instance $I^\diamond$. Let $\Sigma$ and $m(I^\diamond, \Sigma)$ denote the solution and the solution value returned. If $m(I^\diamond, \Sigma) < V,$ then let $BestFound := \Sigma$ and $V := m(I^\diamond, \Sigma).$

(S-3) Return solution $BestFound$ of value $V.$

Fig. 4. Computing the optimal solution for $I^#$.

**Theorem 4** For problem $1|r_j, prec|L_{\text{max}}$ with controllable processing times there exists a linear-time approximation scheme.

### 4 Proof of Lemma 3

In this section we give a proof of Lemma 3. Our goal is to provide an optimal solution for $I^#$ with at most $1/\varepsilon$ preemption compact jobs and such that $O\text{PT}_{\text{relax}}(I^#) \leq (1 + 4\varepsilon)O\text{PT}(I).$ The procedure to compute such a solution is displayed in Figure 4 and explained in the following.

The bound on the number of preempted compact jobs is obtained by showing that the optimal solution for $I^#$ can be computed by applying a known algorithm (i.e. the preemptive version of the extended Jackson’s rule [6], see step (S-2.2) in Figure 4) for which preemption may occur at a release date; the claim follows by the number of different release dates.

The bound on the solution value is obtained by first examining two artificial instances, $I^{\text{fix}}$ and $I^{\text{target}},$ that we shall use as tool in the remaining part of the proof. Consider instance $I$ and relax the problem by allowing preemp-
tion only for small jobs. Let \( SOL_{relax} \) denote an optimal solution of \( I \) when small jobs can be preempted and let \( OPT_{relax}(I) \) denote its value. Clearly, \( OPT_{relax}(I) \leq OPT(I) \). Instance \( I^{fix} \) is obtained from \( I \) by fixing the processing times and costs of jobs: we show that \( OPT_{relax}(I) = OPT_{relax}(I^{fix}) \) (see Lemma 6). Instance \( I_{target} \) is obtained from \( I^{fix} \) by grouping small jobs and replacing them with compact jobs. We show that \( OPT_{relax}(I_{target}) = OPT_{relax}(I^{fix}) \) (see Lemma 7). Finally, by using instance \( I^{\#} \) we build a solution with \( OPT_{relax}(I^{\#}) \leq OPT_{relax}(I_{target}) + 4\varepsilon OPT(I) \) (see Lemma 8) and the claim follows. The proofs of the claimed results will suggest and make it clear the algorithm displayed in Figure 4.

Let us start defining instance \( I^{fix} \). According to \( SOL_{relax} \), let \( \delta_{relax} \) be the compression parameters of jobs, and let \( s_1^{relax}, s_2^{relax}, ..., s_n^{relax} \) and \( t_1^{relax}, t_2^{relax}, ..., t_n^{relax} \) be the job starting and completion times, respectively. If \( i \in N \), then let \( \rho_j^{relax} := i \varepsilon \). Consider the modified instance \( I^{fix} \) in which the processing time \( p_j^{fix} \) and cost \( c_j^{fix} \) of job \( J_j \) \((j = 1, ..., n)\) are fixed to \( p_j(\delta_j^{relax}) \) and \( c_j(\delta_j^{relax}) \), respectively, while release \( r_j^{fix} \) and delivery \( q_j^{fix} \) times are reset to the following values:

\[
\begin{align*}
    r_j^{fix} &= \rho_j^{relax}, & \text{for } J_j \in L, \\
    r_j^{fix} &= \max\{r_j, \max_{h:J_h \in Pre(j)} r_h^{fix}\}, & \text{for } J_j \in S, \\
    q_j^{fix} &= \max\{q_j, \max_{h:J_h^{fix} < t_h^{fix}} q_h^{fix}\}, & \text{for } J_j \in L, \\
    q_j^{fix} &= \max\{q_j, \max_{h:J_h \in Suc(j)} q_h^{fix}\}, & \text{for } J_j \in S. \\
\end{align*}
\]

If set \( Pre(j) \) is empty, we assume that \( \max_{h:J_h \in Pre(j)} r_h^{fix} = 0 \) (similarly if \( \{h : r_h^{fix} < r_j^{fix}\} \) or \( Suc(j) \) are empty sets). We will see that release and delivery times are modified in the described way to ensure that the optimal order of \( SOL_{relax} \) will be ‘almost’ preserved later.

The following lemma shows that condition (1) is still valid for instance \( I^{fix} \).

**Lemma 5** For each pair of jobs, \( J_j \) and \( J_k \), of instance \( I^{fix} \) the following condition holds: \( J_j \prec J_k \Rightarrow (r_j^{fix} \leq r_k^{fix} \text{ and } q_j^{fix} \geq q_k^{fix}) \).

**Proof.** We first prove by induction that

\[
J_j \prec J_k \implies r_j^{fix} \leq r_k^{fix}. \quad (3)
\]

Consider any fixed ordering \( J_1, ..., J_\nu \) of the jobs that is consistent with the precedence relation. Trivially, condition (3) holds for set \( \{J_1\} \). Assume that condition (3) holds for set \( N_{k-1} = \{J_1, ..., J_{k-1}\} \), then we prove that condition (3) holds for set \( N_k = \{J_1, ..., J_k\} \), for \( 2 \leq k \leq \nu \). If there is no job from set \( N_{k-1} \) that must be processed before \( J_k \), then condition (3) holds for set \( N_k \). Otherwise, we distinguish between the following cases:
(1) If $J_k \in L$
   (a) and there is a large job $J_j$ from set $N_{k-1}$ such that $J_j \prec J_k$, then $r_j^\text{fix} \leq r_k^\text{fix}$ since $\rho_j \leq \rho_k$ (see Step (S-2));
   (b) and there is a small job $J_j$ from set $N_{k-1}$ such that $J_j \prec J_k$, then $r_j \leq r_k \leq \rho_k = r_k^\text{fix}$ by definition of $R_k$ and by condition (1); furthermore $\text{Pre}(j) \subseteq \text{Pre}(k)$, since $J_j \prec J_k$, and $r_k^\text{fix} = \rho_k \geq \max_{h \in \text{Pre}(j)} \rho_h = \max_{h \in \text{Pre}(j)} r_h^\text{fix}$; therefore
   \[ r_k^\text{fix} \geq \max\{r_j, \max_{h:J_h \in \text{Pre}(j)} r_h^\text{fix}\} = r_j^\text{fix}. \]

(2) If $J_k \in S$
   (a) and there is a large job $J_j$ from set $N_{k-1}$ such that $J_j \prec J_k$, then $J_j \in \text{Pre}(k)$ and therefore $r_j^\text{fix} \leq r_k^\text{fix}$;
   (b) and there is a small job $J_j$ from set $N_{k-1}$ such that $J_j \prec J_k$, then $r_j \leq r_k$ by condition (1); furthermore $\text{Pre}(j) \subseteq \text{Pre}(k)$ since $J_j \prec J_k$, and $\max_{h \in \text{Pre}(k)} r_h^\text{fix} \geq \max_{h \in \text{Pre}(j)} r_h^\text{fix}$; therefore
   \[ r_k^\text{fix} = \max\{r_k, \max_{h:J_h \in \text{Pre}(k)} r_h^\text{fix}\} \geq \max\{r_j, \max_{h:J_h \in \text{Pre}(j)} r_h^\text{fix}\} = r_j^\text{fix}. \]

Hence, we have proved that if $J_j \prec J_k$ then $r_j^\text{fix} \leq r_k^\text{fix}$. This result guarantees that it is always possible to find an ordering of the jobs that is consistent with the precedence relation and such that $r_1^\text{fix} \leq \cdots \leq r_\nu^\text{fix}$. Let $J_1, \ldots, J_\nu$ denote this ordering. In the following, we prove by induction that

\[ J_j \prec J_k \implies q_j^\text{fix} \geq q_k^\text{fix}. \quad (4) \]

Trivially, condition (4) holds for set $\{J_\nu\}$. Assume that condition (4) is true for set $N_{j+1} = \{J_{j+1}, \ldots, J_\nu\}$, then we prove that condition (4) holds for set $N_j = \{J_j, \ldots, J_\nu\}$, for $1 \leq j \leq \nu - 1$. If there is no job from set $N_{j+1}$ that must be processed after $J_j$, then condition (3) holds for set $N_j$. Otherwise, we distinguish between the following cases:

(1) If $J_j \in L$
   (a) and there is a large job $J_k$ from set $N_{j+1}$ such that $J_j \prec J_k$, then $r_j^\text{fix} \leq r_k^\text{fix}$, $q_j \geq q_k$ by condition (1), and
   \[ \max_{h:J_h \prec J_j} q_h^\text{fix} \geq \max_{h:J_h \prec J_k} q_h^\text{fix}; \]
   it follows that
   \[ q_j^\text{fix} = \max\{q_j, \max_{h:J_h \prec J_j} q_h^\text{fix}\} \geq \max\{q_k, \max_{h:J_h \prec J_k} q_h^\text{fix}\} = q_k^\text{fix}; \]
   (b) and there is a small job $J_k$ from set $N_{j+1}$ such that $J_j \prec J_k$, then $\text{Suc}(k) \subseteq \text{Suc}(j)$, and since $J_1, \ldots, J_\nu$ denote an ordering of the jobs
that is consistent with the precedence relation, we have \( \text{Suc}(k) \subseteq \text{Suc}(j) \subseteq N_{j+1} \). In the previous case (1.a) we have shown that \( q_j^{\text{fix}} \geq \max_{h: J_h \in \text{Suc}(j)} q_h^{\text{fix}} \) and hence \( q_j^{\text{fix}} \geq \max_{h: J_h \in \text{Suc}(k)} q_h^{\text{fix}} \). By observing that \( q_j^{\text{fix}} \geq q_j \) and \( q_j \geq q_k \) by condition (1), we have

\[
q_j^{\text{fix}} \geq \max\{q_k, \max_{h: J_h \in \text{Suc}(k)} q_h^{\text{fix}}\} = q_k^{\text{fix}}.
\]

(2) If \( J_j \in S \)

(a) and there is a large job \( J_k \) from set \( N_{j+1} \) such that \( J_j \prec J_k \), then \( q_j^{\text{fix}} \geq q_k^{\text{fix}} \) since \( J_k \in \text{Suc}(j) \);

(b) and there is a small job \( J_k \) from set \( N_{j+1} \) such that \( J_j \prec J_k \), then \( q_j \geq q_k \) by condition (1); furthermore \( \text{Suc}(j) \supseteq \text{Suc}(k) \) since \( J_j \prec J_k \), and \( \max_{h: J_h \in \text{Suc}(k)} q_h^{\text{fix}} \geq \max_{h: J_h \in \text{Suc}(k)} q_h^{\text{fix}} \); therefore

\[
q_j^{\text{fix}} = \max\{q_j, \max_{h: J_h \in \text{Suc}(j)} q_h^{\text{fix}}\} \geq \max\{q_k, \max_{h: J_h \in \text{Suc}(k)} q_h^{\text{fix}}\} = q_k^{\text{fix}}.
\]

Let \( OPT_{\text{relax}}(I^{\text{fix}}) \) denote the optimum value for \( I^{\text{fix}} \) when small jobs can be preempted. Then we have the following

Lemma 6 \( OPT_{\text{relax}}(I^{\text{fix}}) = OPT_{\text{relax}}(I) \).

Proof. Instance \( I^{\text{fix}} \) is obtained from \( I \) by fixing job processing times and costs as in \( SOL_{\text{relax}} \), and by increasing (or leaving unchanged) release dates and delivery times. The latter follows by proving that release dates of large jobs cannot decrease: for \( J_j \in L \), if \( i \varepsilon \leq s_j^{\text{relax}} < (i+1)\varepsilon \), for some \( i \in \mathbb{N} \), then \( r_j^{\text{fix}} = r_j^{\text{relax}} = i\varepsilon \); moreover \( r_j = k\varepsilon \) for some \( k \in \mathbb{N} \) and, clearly, \( r_j \leq s_j^{\text{relax}} \); therefore \( r_j \leq r_j^{\text{fix}} \).

Therefore, when small jobs can be preempted, any feasible solution for \( I^{\text{fix}} \) is also a feasible solution for \( I \), since no job is processed before its release date. Moreover, \( OPT_{\text{relax}}(I^{\text{fix}}) \geq OPT_{\text{relax}}(I) \). The claim follows by proving that there exists a solution for \( I^{\text{fix}} \) of value at most \( OPT_{\text{relax}}(I) \).

Consider the optimal solution \( SOL_{\text{relax}} \). It is easy to check that \( s_j^{\text{fix}} \leq s_j^{\text{relax}} \) \( (j = 1, \ldots, n) \) and, therefore, we can schedule the jobs of instance \( I^{\text{fix}} \) exactly as in \( SOL_{\text{relax}} \): the starting and the completion time of job \( J_j \) are \( s_j^{\text{relax}} \) and \( t_j^{\text{relax}} \), respectively, for \( j = 1, \ldots, n \). Let \( J_c \) be the job whose delivery is completed last, then the value of this solution is equal to \( t_c^{\text{relax}} + q_c^{\text{fix}} \). If we prove that \( t_c^{\text{relax}} + q_c^{\text{fix}} \leq \text{OPT}_{\text{relax}}(I) \), then the claim follows.

We prove that \( t_c^{\text{relax}} + q_c^{\text{fix}} \leq \text{OPT}_{\text{relax}}(I) \) by induction. Let \( J_1, \ldots, J_n \) denote any fixed ordering of the jobs that is consistent with the precedence relation and such that \( r_1^{\text{fix}} \leq \cdots \leq r_n^{\text{fix}} \). (Note that this is possible by Lemma 5.) If
\(c = n\), then \(q^\text{fix}_c = q_c\) and \(t^\text{relax}_c + q^\text{fix}_c \leq \text{OPT}_{\text{relax}}(I)\). Otherwise, assume that \(t^\text{relax}_j + q^\text{fix}_j \leq \text{OPT}_{\text{relax}}(I)\) for every \(j = c + 1, \ldots, n\) (induction hypothesis).

If \(J_c\) is a large job, let \(J_h\) denote the job with \(r^\text{fix}_c < r^\text{fix}_h\) and \(q^\text{fix}_h = \max_{j : r^\text{fix}_j < r^\text{fix}_c} q^\text{fix}_j\) (ties are broken arbitrarily). Since \(r^\text{fix}_c < r^\text{fix}_h\) and \(r^\text{fix}_c = r^\text{fix}_c\), it follows that \(r^\text{fix}_c \leq s^\text{relax}_c < r^\text{fix}_h \leq s^\text{relax}_h\), and job \(c\) is completed before job \(h\) starts in \(\text{SOL}_{\text{relax}}\), since \(J_c\) starts before job \(h\) and a large job cannot be preempted, i.e. \(s^\text{relax}_h \geq t^\text{relax}_c\).

From induction hypothesis we know that \(t^\text{relax}_h + q^\text{fix}_h \leq \text{OPT}_{\text{relax}}(I)\), and hence \(t^\text{relax}_c + p^\text{fix}_h + q^\text{fix}_h \leq \text{OPT}_{\text{relax}}(I)\). Observe that \(t^\text{relax}_c + q_c \leq \text{OPT}_{\text{relax}}(I)\). It follows that

\[
t^\text{relax}_c + \max\{q^\text{fix}_c, q_c\} \leq t^\text{relax}_c + \max\{(p^\text{fix}_h + q^\text{fix}_h), q_c\} \leq \text{OPT}_{\text{relax}}(I).
\]

Note that \(q^\text{fix}_c = \max\{q^\text{fix}_c, q_c\}\), and we have \(t^\text{relax}_c + q^\text{fix}_c \leq \text{OPT}_{\text{relax}}(I)\).

Otherwise, if \(J_c\) is a small job, let \(J_h\) denote the job such that \(J_h \in \text{SUC}(c)\) and \(q^\text{fix}_h = \max_{j : J_h \in \text{SUC}(c)} q^\text{fix}_j\) (ties are broken arbitrarily). Since \(J_h \in \text{SUC}(c)\), it follows that job \(J_c\) must be completed before job \(J_h\) starts, i.e. \(s^\text{relax}_h \geq t^\text{relax}_c\).

From the induction hypothesis we know that \(t^\text{relax}_h + q^\text{fix}_h \leq \text{OPT}_{\text{relax}}(I)\), and hence \(t^\text{relax}_c + p^\text{fix}_h + q^\text{fix}_h \leq \text{OPT}_{\text{relax}}(I)\). It follows that

\[
t^\text{relax}_c + \max\{q^\text{fix}_h, q_c\} \leq t^\text{relax}_c + \max\{(p^\text{fix}_h + q^\text{fix}_h), q_c\} \leq \text{OPT}_{\text{relax}}(I),
\]

since \(t^\text{relax}_c + q_c \leq \text{OPT}_{\text{relax}}(I)\). The claim follows by observing that \(q^\text{fix}_c = \max\{q^\text{fix}_h, q_c\}\).

Consider the instance \(I^\text{target}\) obtained from instance \(I^\text{fix}\) by replacing all the jobs from \(S_i\) \((i = 1, \ldots, \pi)\) with a single job having the same release date and delivery time as the jobs from \(S_i\), and processing time and cost equal to \(\sum_{J_j \in S_i} p^\text{fix}_j\) and \(\sum_{J_j \in S_i} c^\text{fix}_j\), respectively.

**Lemma 7** \(\text{OPT}_{\text{relax}}(I^\text{fix}) = \text{OPT}_{\text{relax}}(I^\text{target})\).

**Proof.** Consider the preemptive version of the extended Jackson’s rule: schedule the jobs starting at the smallest \(r_j\)-value; at each decision point \(t\) given by a release date or a finishing time of some job, schedule a job \(j\) with the following properties: \(r_j \leq t\), all its predecessors are scheduled, and it has the largest delivery time. A preemption occurs at a release date if the newly released job has a bigger delivery time than the that of the job currently being processed. We shall denote this algorithm as PJR. PJR is known \([6]\) to solve optimally problem \(1|\sum_{j} r_j, \text{prec} | L_{\text{max}}\) when preemption is allowed (in the notation of Graham et al. \([2]\) this problem is denoted \(1|\sum_{j} r_j, \text{pmtm}, \text{prec} | L_{\text{max}}\)). Therefore, if we apply PJR to instance \(I^\text{fix}\) we obtain a solution whose value cannot be greater than \(\text{OPT}_{\text{relax}}(I^\text{fix})\) (since this algorithm solves the relaxed problem where every job can be preempted).
Interestingly, we can show that when we set the release date and delivery time values according to (2), no large job is preempted. To show this, assume for the purposes of contradiction, that there exists a large job $J_j$ that is preempted by using PJR. Then, there exists a job $J_k$ with $q_k^{\text{fix}} > q_j^{\text{fix}}$ that is released when $J_j$ is processed, thus $r_j^{\text{fix}} < r_k^{\text{fix}}$. But $J_j$ cannot be a large job since according to (2) if $r_j^{\text{fix}} < r_k^{\text{fix}}$ then $q_j^{\text{fix}} \geq q_k^{\text{fix}}$.

Recall that we have partitioned the set $S$ of small jobs into $\pi$ subsets, namely $S_1, S_2, \ldots, S_\pi$, such that jobs belonging to the same subset have the same execution profile. Now, according to (2), it is easy to check that jobs belonging to the same subset $S_i$ $(i = 1, \ldots, \pi)$ are reset to the same $r_j^{\text{fix}}, q_j^{\text{fix}}$-values. The latter follows from (2) and by recalling that jobs belonging to the same subset $S_i$ have the same $r_j, q_j, \text{Pre}(j)$ and $\text{Suc}(j)$. Moreover, if $J_h$ is a (front) back-neighbor of $S_i$ $(i = 1, \ldots, \pi)$ then by Lemma 5 we have $r_h^{\text{fix}} \leq r_j^{\text{fix}}$ and $q_h^{\text{fix}} \geq q_j^{\text{fix}}$ ($r_j^{\text{fix}} \leq r_h^{\text{fix}}$ and $q_j^{\text{fix}} \geq q_h^{\text{fix}}$), for every $J_j \in S_i$. Therefore, we do not change the value of the solution returned by PJR if we assume that job $J_h$ is processed before (after) the jobs from $S_i$. Observe that after these changes, if $J_j < J_k$ ($J_k < J_j$), $J_j \in S_i$ and $J_k \notin S_i$, then $J_j < J_k$ ($J_k < J_j$) for each $J_j \in S_i$. Again, we do not change the value of the solution returned by PJR if we assume that all the jobs from $S_i$ are processed one after the other, according to any fixed ordering that is consistent with the precedence relation. For simplicity, let us use again $I^{\text{fix}}$ to denote this modified instance. Now, it is easy to check that the solution value returned by PJR on instance $I^{\text{target}}$ is equal to $\text{OPT}_{\text{relax}}(I^{\text{fix}})$.

We now show that from $I^\#$ it is possible to obtain an instance $I^\reflectbox{A}$ that is nearly the same as $I^{\text{target}}$. Instance $I^\#$ is obtained from $I$ by replacing each large job $J_j$ with a job $J_j^\#$ having the same release date and delivery time of $J_j$, but processing times belonging to $P_j$ (see subsection 2.3). By definition of $P_j$, there exists a value $p_j \in P_j$ such that $0 \leq p_j - p_j^{\text{fix}} \leq \max\{\varepsilon^3, \varepsilon p_j^{\text{fix}}\}$. The corresponding cost value $C_j(p_j)$ is obtained by rounding up to the nearest value of set $V_L$ the cost of job $J_j$ when its processing time is $p_j$. Recall that we are assuming, without loss of generality, that $\ell_j \leq u_j$ and $c_{j}^u \leq c_j^\#$, for all jobs $J_j$. Therefore, since $p_j \geq p_j^{\text{fix}}$, we have that the cost of job $J_j$ when its processing time is $p_j$ is not greater than the cost of job $J_j$ when its processing time is $p_j^{\text{fix}}$. It follows that $C(p_j) - c_j^{\text{fix}} \leq \max\{\varepsilon, \varepsilon c_j^{\text{fix}}\}$. Moreover, the jobs from set $S_i$ $(i = 1, \ldots, \pi)$ have been replaced with a single job $J_i^\#$. Job $J_i^\#$ has the same release $r_i^\#$ and delivery time $q_i^\#$ as the jobs from $S_i$, but processing times belonging to $P_i$ (see subsection 2.3). By definition of $P_i$, there exists a value $p_i \in P_i$ such that $0 \leq p_i - \sum_{j \in S_i} p_j^{\text{fix}} \leq \max\{\varepsilon/\pi, \varepsilon \sum_{j \in S_i} p_j^{\text{fix}}\}$. The corresponding cost $C_i(p_i)$ is obtained by first computing the minimum sum of costs $L P(S_i, p_i)$ for jobs belonging to $S_i$, when the total sum of processing times is at most $p_i$, and then by rounding $L P(S_i, p)$ up to the nearest value of set $V_L$. Note that $L P(S_i, p_i)$ is not greater than $\sum_{j \in S_i} c_j^{\text{fix}}$ since $p_i \geq \sum_{j \in S_i} p_j^{\text{fix}}$. Therefore, it is easy to check that $C(p_i) - \sum_{j \in S_i} c_j^{\text{fix}} \leq \max\{\varepsilon/\pi, \varepsilon \sum_{j \in S_i} r_j^{\text{fix}}\}$. Now, let
us use $I^\uparrow$ to denote the instance obtained from $I^\#$ by fixing the processing time and cost of each job $J^\#_j$ to $p_j$ and $C(p_j)$, respectively; moreover, assume that the release dates and delivery times of jobs in $I^\uparrow$ are set as in $I^{\text{target}}$. Then, instance $I^\uparrow$ is “nearly the same” as $I^{\text{target}}$ and the optimal solution value of $I^\uparrow$ may be greater than $\text{OPT}_{\text{relax}}(I^{\text{fix}})$ by at most

$$\sum_{J^\#_j \in L^\#} \left( \max \left\{ \varepsilon^3, \varepsilon \tilde{p}_{j}^{\text{fix}} \right\} + \max \left\{ \varepsilon^3, \varepsilon \tilde{c}_{j}^{\text{fix}} \right\} \right) + \sum_{J^\#_j \in S^\#} \left( \max \left\{ \varepsilon/\pi, \varepsilon \sum_{J_j \in S_i} p_j^{\text{fix}} \right\} + \max \left\{ \varepsilon/\pi, \varepsilon \sum_{J_j \in S_i} c_j^{\text{fix}} \right\} \right) \leq 4\varepsilon \text{OPT}(I).$$

**Lemma 8** $\text{OPT}_{\text{relax}}(I^\#) \leq \text{OPT}_{\text{relax}}(I^{\text{target}}) + 4\varepsilon \text{OPT}(I)$.

By the previous arguments the optimal solution of instance $I^\#$ can be easily obtained by executing the preemptive extended Jackson’s rule on a constant number of instances obtained from $I^\#$ by changing the release dates, delivery times, and fixing processing times and costs of jobs; the best schedule generated is output. Without loss of generality, let us renumber jobs such that $L^\# = \{J^\#_1, \ldots, J^\#_\lambda\}$, where $\lambda = |L^\#|$. For $j = 1, \ldots, \lambda$, let $R_j = \{r_j^\# + i\varepsilon : i \in \mathbb{N}$ and $r_j^\# + i\varepsilon \leq 3 - \varepsilon\}$. The release dates of jobs $J^\#_1, \ldots, J^\#_\lambda$ are reset to new values taken from $R_1, \ldots, R_\lambda$, respectively. Depending on these values, the other release dates and delivery times may also change. Moreover, the processing requirements of each job $J^\#_j$ are fixed according to the set of alternative pairs $(P_j, C_j(P_j))$, as computed in subsection 2.3. More precisely, our main algorithm performs the steps displayed in Figure 4.

Since $\text{OPT}_{\text{relax}}(I^{\text{fix}}) \leq \text{OPT}(I) \leq 3$, and by definition of large jobs, in any optimal solution the starting time of each large job cannot be later than $3 - \varepsilon$. Therefore, $\rho_{j}^{\text{relax}} \in R_j$, for each $J^\#_j \in L^\#$, and in one of the iterations of step (S-2), we have job processing times, costs, release dates and delivery times such that the application of PJR on that instance returns the optimal solution.

4.0.0.1 **Acknowledgements.** Supported by Swiss National Science Foundation project 200021-104017/1, “Power Aware Computing”, and by the Swiss National Science Foundation project 200021-100539/1, “Approximation Algorithms for Machine scheduling Through Theory and Experiments”.

**References**


