Acta Cybernetica **15** (2001) 369–378.

A PTAS for single machine scheduling with controllable processing times

Petra Schuurman* and Gerhard J. Woeginger†

Abstract

We deal with a single machine scheduling problem in which each job has a release date, a delivery time and a controllable processing time. The fact that the jobs have a controllable processing time means that it is allowed to compress (a part of) the processing time of the job, in return for compression cost. The objective is to find a schedule that minimizes the total cost, that is, the latest delivery time of any job plus the total compression cost. In this note we discuss how the techniques of Hall and Shmoys [3] and Hall [1] can directly be applied to design a polynomial time approximation scheme for this problem.

Keywords. Scheduling, worst case analysis, approximation algorithm, approximation scheme, controllable processing time.

1 Introduction

We consider a scheduling problem in which n jobs, $J_1, \ldots, J_n$, have to be scheduled on a single machine. Each job J_j has a processing requirement p_j and it becomes available for processing at a specific point in time, which we call its release date r_j. After its processing, J_j needs some delivery time (independent of the machine) before it is completed (e.g. cooling off or transportation time); we denote this delivery time by q_j. We assume that no *preemption* is allowed, i.e. once a job has been started, it must be completed without interruptions. The goal is to minimize the latest job delivery completion time, which we call the *length* of the schedule. This scheduling problem with release dates and delivery times is usually denoted by $1 \mid r_j \mid L_{\max}$.

In this paper we consider a more difficult variation of problem $1 \mid r_j \mid L_{\max}$: There are situations where one can and wishes to board out part of the work. In case part of the work of job J_j is boarded out, we say that J_j is *compressed.* The

*Email: petra@win.tue.nl. Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

†Email: gwoegi@opt.math.tu-graz.ac.at. Institut für Mathematik B, TU Graz, Steyrergasse 30, A-8010 Graz, Austria, and Department of Mathematics, University of Twente, 7500 AE Enschede, The Netherlands. Supported by the START program Y43-MAT of the Austrian Ministry of Science.

amount J_j is compressed is denoted by x_j, and the maximum possible compression of J_j is denoted by u_j. Here x_j is a decision variable with $0 \leq x_j \leq u_j$; note that x_j does not need to be integral. We call the amount of processing time of a job J_j after compression, its *shortened processing time* a_j. Clearly, $a_j = p_j - x_j$. Of course compressing a job J_j results in extra costs, so-called *compression costs*. We denote the compression cost per unit of J_j by c_j. The total compression cost, denoted by C, satisfies $C = \sum_{j=1}^{n} c_j x_j$. The objective is now to find a schedule σ that minimizes the total cost $T(\sigma)$, that is the length $L(\sigma)$ of the schedule plus the total compression cost $C(\sigma)$.

For the computational complexity of the single-machine problem, the following is known. Let us first discuss the cases where all delivery times are equal. Then determining a schedule with minimal length of an instance of $1 \,|\, r_j \,|\, L_{\max}$ comes down to finding a schedule with minimal makespan, where the makespan is the completion time of the latest job. In this case, determining a schedule with minimal makespan can be done in polynomial time by ordering the jobs in nondecreasing order of release date. The compressible processing time variant also admits a polynomial time algorithm in case of equal delivery times: After sorting the jobs in nondecreasing order of release date, we start compressing the jobs with compression cost less than 1 in order of nonincreasing compression costs, where we only compress a job in case the length of the schedule thereby decreases. Now let us turn to the cases with arbitrary delivery times. Lenstra, Rinnooy Kan & Brucker [5] proved that $1 \,|\, r_j \,|\, L_{\max}$ is $\mathcal{NP}$-hard in the strong sense. Therefore, the single machine problem with compressible processing times, which encloses $1 \,|\, r_j \,|\, L_{\max}$ as a special case, is also strongly $\mathcal{NP}$-hard.

These later observations justify the search for approximate solutions by means of polynomial time approximation algorithms. We say that such an approximation algorithm has *worst case performance guarantee* ρ, or is a ρ-approximation algorithm for short, if it always delivers a solution with value at most $\rho \cdot \mathrm{OPT}$. Here, OPT denotes the value of an optimal solution, which will also be denoted by T^* in our case. For a strongly $\mathcal{NP}$-hard problem, the best that we can hope for, is the existence of a polynomial time approximation scheme, a PTAS for short. A PTAS is a family of polynomial time $(1+\varepsilon)$-approximation algorithms for all $\varepsilon > 0$.

Already in 1971 Schrage [8] developed a 2-approximation algorithm for $1 \,|\, r_j \,|\, L_{\max}$ based on a simple heuristic. During the eighties various improvements upon Schrage's heuristic were designed to obtain better performance guarantees. First Potts [7] modified the heuristic of Schrage into a $\frac{3}{2}$-approximation algorithm, and then Hall and Shmoys [3] on their turn improved Potts' heuristic to get a $\frac{5}{4}$-approximation algorithm. In 1989, Hall and Shmoys [2] developed a new idea, that, among others, helped them in designing an approximation scheme for $1 \,|\, r_j \,|\, L_{\max}$. Hereby the approximability status of $1 \,|\, r_j \,|\, L_{\max}$ was determined. The results of Hall and Shmoys [2] were even stronger, since they extended to the problem with precedence constraints.

In 1991, Zdrzalka [9] designed a $(\frac{1}{2}+\tau)$-approximation algorithm for the single-machine problem with compressible processing times. Here, τ is the performance guarantee of the best approximation algorithm for $1 \,|\, r_j \,|\, L_{\max}$. Hence the approx-

imation algorithm of Zdrzalka has performance guarantee arbitrarily close to $\frac{3}{2}$. Nowicki [6] improved on this result by constructing a $(\frac{4}{3}+\varepsilon)$-approximation algorithm, where $\varepsilon > 0$ can be arbitrarily small.

In this note we apply the ideas of Hall and Shmoys [2] for $1 \mid r_j \mid L_{\max}$ and some of the ideas of Hall [1] for the flowshop problem, to design an approximation scheme for minimizing the latest delivery time under controllable processing times.

2 The approximation scheme

Clearly the total cost is a combination of two opposing objectives: On the one hand we want to minimize the total length and on the other hand we wish to minimize the total compression cost. Therefore, it seems logical to separate these costs when attacking the problem. It comes in hand to express the minimal compression costs as a function of the length of the schedule; to that end we define $C_{opt}(L)$ to be the minimal compression costs of a schedule of length L. Note that the minimal total cost, T^*, equals $\min_L(L + C_{opt}(L))$. We start with the following observation.

Observation 2.1 *The minimal compression costs of a schedule $C_{opt}(L)$ is nonincreasing in its length L.* □

Before giving a detailed description of the approximation scheme, we start with a global explanation of the ideas behind the scheme. Our approach consists of finding a schedule with nearly optimal costs given a fixed length. We construct an algorithm $A_\varepsilon(L)$ that, given a feasible schedule length L, produces a schedule of length at most $(1+\frac{\varepsilon}{3})L$ and costs at most $C_{opt}(L)$.

As introduced by Hall and Shmoys in [2] and used in various papers later on, we make use of a so called *outline scheme*. An outline scheme is a partition of the feasible schedules into sets. This partition is done in such a way that schedules in the same sets, share the same characteristics. The idea of the outline scheme is to generate a good schedule for each set and then to take the best among these generated schedules to be the approximate solution. For this idea to work, the following two conditions are necessary: First, the partition has a polynomial number of sets, and second, for each set, we are able to find a schedule that is nearly as good as the best schedule within that set, in polynomial time.

To satify the first condition, as one usually does in designing a PTAS, we distinguish between big jobs and small jobs. We call a job *big* if its shortened processing time is at least $\delta^2 L$, otherwise we call a job *small*. Note that, in contrary to the usual definition, this definition is schedule dependent.

Each set in the outline scheme is characterized by, what we call, a *skeleton*. Roughly speaking, this skeleton determines the approximate position of the big jobs in a schedule; an exact characterization of a skeleton is given in Section 2.2. Our algorithm $A_\varepsilon(L)$, which is based on building a good schedule for each skeleton, consists of two stages. Given a candidate length L, we first guess the approximate position of the big jobs in an optimal schedule by enumerating all possible skeletons. In the second stage, we construct a schedule for each skeleton by fitting in the small

jobs. The best among all those schedules will be the output of our approximation algorithm.

Below we describe the various steps of our approximation scheme in detail: First, in Section 2.1, we indicate for which lengths L we construct an approximate solution; the characteristics of a skeleton are described in Section 2.2; in Section 2.3 we show how we have incorporated the small jobs to obtain good schedules. Finally, in Section 2.4, we show that our algorithm indeed produces a near-optimal solution, i.e. a solution with cost at most $(1+\varepsilon)T^*$.

2.1 Fixing the length of the schedule

As we want our algorithm to have polynomial running time, we only evaluate a constant number of different lengths L. In order to determine suitable lengths, we start by computing natural lower and upper bounds on the length of a schedule. By means of the approximation scheme of Hall and Shmoys [2], we get a lower bound on the length of a schedule by computing an approximate schedule in case all jobs are compressed upon their maximal compression u_j. We choose the parameters in the approximation scheme of Hall and Shmoys such that the approximate length of this schedule, which we denote by $L_0(\varepsilon)$, is at most $(1+\frac{\varepsilon}{3})$ times the length of an optimal schedule in which all jobs are maximaly compressed.

Analogously, we determine a natural upper bound on the length of a schedule by finding an approximate schedule for the problem in which no job is compressed. The approximate length of this schedule is denoted by $L_\infty(\varepsilon)$, where $L_\infty(\varepsilon)$ is less than $(1+\frac{\varepsilon}{3})$ times the length of an optimal schedule in which no job is compressed. Clearly, we only consider schedules with length between $L_0(\varepsilon)$ and $L_\infty(\varepsilon)$.

We would like subsequent lengths to differ at most a multiplicative factor of $1+\frac{\varepsilon}{3}$. For this purpose the range of $[L_0(\varepsilon), L_\infty(\varepsilon)]$ could be too wide, therefore we need to construct better lower and upper bounds.

The work of Zdrzalka [9] and Nowicki [6] gives us good lower and upper bounds on the total cost T^* of an optimal schedule. Consider a $\frac{3}{2}$-approximation algorithm for the problem, obtained by the approach of Nowicki. Let $T(\sigma_{\mathcal{N}})$ be the cost of schedule $\sigma_{\mathcal{N}}$ produced by this algorithm. Clearly, we do not execute $A_\varepsilon(L)$ for lengths L with $L > T(\sigma_{\mathcal{N}})$. Furthermore, in case $L < \frac{4}{9}\varepsilon T(\sigma_{\mathcal{N}})$, that is, $L < \frac{2}{3}\varepsilon T^*$, we also do not execute $A_\varepsilon(L)$.

Concluding: We execute algorithm $A_\varepsilon(L)$ for a constant number of lengths L, starting with $\max(L_0(\varepsilon), \frac{4}{9}\varepsilon T(\sigma_{\mathcal{N}}))$, increasing the length every time with a factor $(1+\frac{\varepsilon}{3})$ until the value exceeds $\min(L_\infty(\varepsilon), T(\sigma_{\mathcal{N}}))$. The structure of our scheme is as follows.

Scheme

INPUT: A number $\varepsilon < 1$.

Compute schedules σ_0 and σ_∞ with lengths $L_0(\varepsilon)$ and $L_\infty(\varepsilon)$ respectively, by means of the approximation scheme of Hall and Shmoys.
Construct a schedule $\sigma_{\mathcal{N}}$, with cost $T(\sigma_{\mathcal{N}})$, by a $\frac{3}{2}$-approximation algorithm obtained by the approach of Nowicki.

$L := \max(L_0(\varepsilon), \frac{4}{9}\varepsilon T(\sigma_{\mathcal{N}}))$;
WHILE $L < \min(L_\infty(\varepsilon), T(\sigma_{\mathcal{N}}))$ DO
    Construct an approximate schedule σ by algorithm $A_\varepsilon(L)$;
    $L := L(1 + \frac{\varepsilon}{3})$
END WHILE
OUTPUT: A schedule σ for which the total cost $T(\sigma)$ is minimal among all constructed schedules.

The algorithm $A_\varepsilon(L)$ consists of two stages, which are explained in the next two sections.

2.2 Stage 1: Characterising the skeleton of a schedule

Following the idea of the outline scheme, the first stage of $A_\varepsilon(L)$ consists of grouping together schedules with the same skeleton. We first characterize such a skeleton, after which we enumerate all feasible skeletons in order to build one good schedule from each of these skeletons in the second stage.

Given a candidate length L and a constant ε, we divide the interval $[0, L)$ in $\frac{1}{\delta}$ intervals I_i, $i = 1, \ldots, \frac{1}{\delta}$, of equal length, i.e. $I_i = [(i-1)L\delta, iL\delta)$, where $\delta = \frac{1}{18}\varepsilon$. For each interval I_i, we would like to know which jobs are started in this interval and the amount of time that they occupy the machine. As in e.g. Hall & Shmoys [3] and Hall [1], we do not determine the corresponding starting interval for all jobs, but only for the big jobs. The only difference between our problem and for example the flowshop problem studied by Hall, is that, due to the possibility of compressing, we do not know beforehand which jobs are big and which jobs are small. However, we can easily overcome this difference: a straightforward extension of the approach in Hall and Shmoys [3], leads to a PTAS.

We characterize a skeleton by the following:

- For each interval I_i, $i = 1, \ldots, \frac{1}{\delta}$, we specify a set of big jobs B_i. The cardinality of each set B_i is at most $\frac{1}{\delta}$ and $B_i \cap B_l = \emptyset$ for all i and l.
- For each job J_j in a set B_i, we specify its approximate shortened processing time $\hat{a}_j$; $\hat{a}_j$ is between $\delta^2 L$ and L, and is a multiple of $\delta^3 L$.
- For each interval I_i, $i = 1, \ldots, \frac{1}{\delta}$, we specify the approximate total shortened processing time $\hat{A}_i$ of the small jobs in the interval I_i; $\hat{A}_i$ is a multiple of $\delta^2 L$.

We can represent a skeleton by a vector $\bar{y}$, where $y_i = (B_i, \{\hat{a}_j | J_j \in B_i\}, \hat{A}_i)$. Although the number of different skeletons is polynomial (see the proof of Lemma 2.2 in Section 2.4), we restrict our attention to those skeletons for which there possibly exists a feasible schedule, i.e. so-called *feasible skeletons* that satisfy the following conditions.

- Every big job can be compressed up to $\hat{a}_j$, that is, for every job J_j in a set B_i:

$$p_j - u_j \leq \hat{a}_j \leq \left\lceil \frac{p_j}{\delta^3 L} \right\rceil \delta^3 L.$$

- Every big job is assigned to a compatible starting interval, i.e. for every job $J_j \in B_i$:

$$r_j < i\delta L \text{ and } (i-1)\delta L + \hat{a}_j + q_j \leq \left\lceil \frac{1}{\delta^3} \right\rceil \delta^3 L.$$

- No interval is overloaded, i.e. for every interval I_i and for all $1 \leq l \leq i$

$$\sum_{k=l}^{i} (\hat{A}_k + \sum_{j \in B_k} \hat{a}_j) - \max_{j \in B_i} \hat{a}_j \leq (i-l+1)(\delta L + 2\delta^2 L),$$

the additional $2\delta^2 L$ is due to the rounding of the shortened processing times.

Our approach consists in enumerating all feasible skeletons. In the next section we explain how to transform a feasible skeleton $\bar{y}$ into a feasible schedule $\sigma(\bar{y})$.

2.3 Stage 2: Incorporating the small jobs

Given a feasible skeleton $\bar{y}$, we know the approximate total amount of shortened processing time of the small jobs for each interval I_i and both the starting interval and the approximate shortened processing time of the big jobs. Since we have rounded the shortened processing times, we need to enlarge the intervals I_i. We therefore define intervals $\hat{I}_i$, $i = 1, \ldots, \frac{1}{\delta}$, where $\hat{I}_i = [(i-1)(\delta L + 3\delta^2 L), i(\delta L + 3\delta^2 L))$. We now determine the starting interval $\hat{I}_i$ and the shortened processing time for the small jobs. Analogous to Hall [1], we determine this data by means of a linear program.

To that end, we define decision variables a_{ij}, where a_{ij} represents the shortened processing time of job J_j in the interval $\hat{I}_i$. The set of small jobs is denoted by S. In fact our linear program assigns different pieces of the same job to different intervals, which corresponds to the construction of a preempted schedule. The first two inequalities in the LP-formulating below, express the bounds on the amount of compression x_j. Equalities three and four impose natural constraints on the intervals each (piece of) job is processed in, whereas inequalities five and six bound the total amount of shortened processing time for each machine and each job, respectively. The goal is of course to minimize the total compression cost.

LP

$$\begin{array}{lrcll}
\min & \sum_j c_j(p_j - \sum_i a_{ij}) & & & \\
\text{s.t.} & p_j - \sum_i a_{ij} & \geq & 0 & \forall j \in S \\
 & p_j - \sum_i a_{ij} & \leq & u_j & \forall j \in S \\
 & a_{ij} & = & 0 \quad \text{if } r_j \geq i\delta L & \forall i = 1, \ldots, \frac{1}{\delta}, \forall j \in S \\
 & a_{ij} & = & 0 \quad \text{if } L - q_j < (i-1)\delta L & \forall i = 1, \ldots, \frac{1}{\delta}, \forall j \in S \\
 & \sum_j a_{ij} & \leq & \hat{A}_i & \forall i = 1, \ldots, \frac{1}{\delta} \\
 & \sum_i a_{ij} & \leq & \delta^2 L & \forall j \in S \\
 & a_{ij} & \geq & 0 & \forall i = 1, \ldots, \frac{1}{\delta}, \forall j \in S
\end{array}$$

The LP may not give a solution, in this case it is clear that there is no schedule corresponding to the skeleton $\bar{y}$. Otherwise, we construct a feasible schedule as follows.

Instead of assigning different pieces of a small job to different intervals, as the LP-solution suggests, we assign the job as a whole, that is, the total shortened processing time $\sum_i a_{ij}$ as determined by the LP, to a single interval $\hat{I}_i$. We recursively determine the jobs that have starting interval $\hat{I}_1, \hat{I}_2, \ldots \hat{I}_{\frac{1}{\delta}}$. In each step i, we first compute R_i, which denotes the subset of small jobs that have not been assigned to $\hat{I}_1, \hat{I}_2, \ldots \hat{I}_{i-1}$ and for which there is a $l \leq i$ with $a_{lj} > 0$. Then, we order the jobs in R_i in increasing order of their delivery time. Finally, we assign the jobs in R_i one by one to $\hat{I}_i$ until the total shortened processing time exceeds $\hat{A}_i$.

Given the starting intervals of the jobs, we order the jobs in each interval $\hat{I}_i$ as follows. First we schedule the small jobs (in arbitrary order), then we schedule the big jobs in order of nondecreasing shortened processing time. We only allow idle time between two jobs with different starting intervals. In order to obtain a feasible schedule, that is, to ensure that each job is started at or after its release time, we introduce an idle interval with length δL at the beginning of the schedule. Hence the intervals $\hat{I}_i$ are shifted by δL time units. We call the schedule constructed above $\sigma(\bar{y})$. It is easy to check that $\sigma(\bar{y})$ is a feasible schedule.

Summarizing: the structure of our algorithm $A_\varepsilon(L)$ is as follows.

Algorithm $A_\varepsilon(L)$

INPUT: A number $\varepsilon < 1$ and an integer L.

$\delta := \frac{1}{18}\varepsilon$;
Divide the interval $[0, L)$ into $\frac{1}{\delta}$ intervals of equal length;
Enumerate all feasible skeletons $\bar{y}$;

FOR each feasible skeleton $\bar{y}$ DO
    Compute the compression cost $C_B(\bar{y})$ for the big jobs in $\bar{y}$
    Solve the LP;
    IF the LP has a solution with value $C_S(\bar{y})$
        THEN construct a schedule $\sigma(\bar{y})$ with length at most
        $(1 + \frac{\varepsilon}{3})L$ and compression cost $C_B(\bar{y}) + C_S(\bar{y})$.
END FOR

OUTPUT: A schedule σ with total minimal cost among all constructed schedules $\sigma(\bar{y})$.

2.4 The analysis

We start this section by showing that $A_\varepsilon(L)$ runs in polynomial time. Then, we conclude that, since the number of executions of $A_\varepsilon(L)$ is constant, as stated in Lemma 2.3, our scheme has polynomial running time. Finally, by means of Lemmas 2.4 and 2.5, we prove that our scheme produces a near-optimal solution.

Lemma 2.2 *Algorithm $A_\varepsilon(L)$ runs in polynomial time.*

Proof. Since the LP described in Section 2.3 clearly runs in polynomial time and the procedure to construct $\sigma(\bar{y})$ is also polynomial, the key factor is the number of different skeletons. If the latter is polynomial, then so is $A_\varepsilon(L)$.

As the number of big jobs per interval is at most $\frac{1}{\delta}$, we have at most $n^{\frac{1}{\delta}}$ different B_i's. For each job J_j we have at most $\frac{1}{\delta^3}$ choices for its approximate shortened processing time $\hat{a}_j$; hence, there are at most $\frac{1}{\delta^3}^{\frac{1}{\delta}}$ different sets $\{\hat{a}_j | J_j \in B_i\}$. Finally, there are $\frac{1}{\delta}$ different choices for $\hat{A}_j$. Concluding: the number of different skeletons is at most

$$\left((n\frac{1}{\delta^3})^{\frac{1}{\delta}}\frac{1}{\delta}\right)^{\frac{1}{\delta}},$$

and therefore $A_\varepsilon(L)$ runs in polynomial time. □

Lemma 2.3 *The number of times $A_\varepsilon(L)$ is executed is a constant that depends on ε.*

Proof. Let κ be the number of times we execute algorithm $A_\varepsilon(L)$ and let L_{first} and L_{last} be the first and last length, respectively, for which algorithm $A_\varepsilon(L)$ is executed. Clearly, $L_{first} \geq \frac{2}{3}\varepsilon T(\sigma_{\mathcal{N}})$ and $L_{last} = (1+\frac{\varepsilon}{3})^{\kappa-1}L_{first} < T(\sigma_{\mathcal{N}})$. Hence,

$$(1+\frac{\varepsilon}{3})^{\kappa-1}L_{first} < T(\sigma_{\mathcal{N}}) \leq \frac{3}{2\varepsilon}L_{first},$$

that is,

$$\kappa < \frac{\log(\frac{3}{2\varepsilon})}{\log(1+\frac{\varepsilon}{3})} + 1.$$ □

From the previous two lemmas it follows that our scheme runs in polynomial time. It now remains to prove that σ_ε, the output of our scheme, has value at most $(1+\varepsilon)T^*$. To that end, we first prove that algorithm $A_\varepsilon(L)$ outputs a near-optimal schedule.

Lemma 2.4 *The algorithm $A_\varepsilon(L)$ produces a schedule with length at most $(1+\frac{\varepsilon}{3})L$ and cost at most $C_{opt}(L)$.*

Proof. Let σ be a schedule with length at most L and cost at most $C_{opt}(L)$. We divide $[0, L)$ in $\frac{1}{\delta}$ intervals I_i of equal length and we define S_i to be the set of small jobs that start in I_i. Next, we construct a skeleton $\bar{y}$, with $y_i = (B_i, \{\hat{a}_j | J_j \in B_i\}, \hat{A}_i)$, where

- B_i is defined to be the set of big jobs that start in I_i;
- for each job $J_j \in B_i$, we define its approximate shortened processing time $\hat{a}_j$ to be $\lceil \frac{a_j}{\delta^3 L} \rceil \delta^3 L$;
- $\hat{A}_i$ is defined to be equal to $\left\lceil (\sum_{J_j \in S_i} a_j)/(\delta^2 L) \right\rceil \delta^2 L$.

After first enlarging each interval I_i by a multiplicative factor of $1+\delta$ (i.e. an absolute increase of δL), to compensate the rounding of the big jobs, and then enlarging each I_i by δL, to compensate for the rounding of the small jobs, it is clear that every big job can still be started within its assigned interval.

Given the skeleton $\bar{y}$, we construct a feasible schedule, $\sigma_\varepsilon(\bar{y})$, as described in Section 2.3. Thanks to the additional $\delta^2 L$ units of space in each interval $\hat{I}_i$, we can also cope with an additional small job that is possibly assigned to $\hat{I}_i$. Furthermore, since no big job is compressed more than in the original schedule (we have rounded up the shortened processing times) and the small jobs are compressed in such a way that the compression cost are minimized, the compression cost of $\sigma_\varepsilon(\bar{y})$ is at most $C_{opt}(L)$.

Let us now compute the length of $\sigma_\varepsilon(\bar{y})$. We have already argued that no interval $\hat{I}_i$ is overloaded, that is, the jobs assigned to $\hat{I}_i$ can actually be started in $\hat{I}_i$. Because our algorithm has shifted all jobs δL units to the right, no job is started before its release time. But a job might complete after L. Let us reason how much this additional delay might be. Consider a big job in $\sigma_\varepsilon(\bar{y})$ that starts at a time $\hat{t}_j \in \hat{I}_i$. In σ this job starts at time $t_j \in I_i$ i.e. $t_j \geq (i-1)\delta L$. As $t_j + a_j + q_j \leq L$,

$$\begin{aligned}
\hat{t}_j + \hat{a}_j + q_j &\leq \delta L + i(\delta L + 3\delta^2 L) + \hat{a}_j + q_j \\
&\leq 2\delta L + 3i\delta^2 L + t_j + (1+\delta)a_j + q_j \\
&\leq 5\delta L + (1+\delta)L \\
&\leq (1+6\delta)L \\
&= (1+\frac{\varepsilon}{3})L.
\end{aligned}$$

Hence, the delivery completion time of each big job is at most $(1+\frac{\varepsilon}{3})L$. A similar analysis can be made for each small job. Concluding: $L(\sigma_\varepsilon(\bar{y})) \leq (1+6\delta)L = (1+\frac{\varepsilon}{3})L$ and $C(\sigma_\varepsilon(\bar{y})) \leq C_{opt}(L)$. □

Lemma 2.5 *The scheme proposed in the previous sections computes a solution with value at most* $(1+\varepsilon)T^*$.

Proof. Let σ^* be an optimal schedule. We now distinguish two cases.

In case (i), the length $L(\sigma^*)$ of schedule σ^* is less than $\frac{2}{3}\varepsilon T^*$. We know that the algorithm $A_\varepsilon(\frac{2}{3}T^*)$ outputs a schedule σ_1 with total cost at most $(1+\frac{\varepsilon}{3})\frac{2}{3}\varepsilon T^* + C_{opt}(L(\sigma^*)) \leq \varepsilon T^* + T^* = (1+\varepsilon)T^*$. This settles the first case.

In case (ii), the length $L(\sigma^*)$ of schedule σ^* is at least $\frac{2}{3}\varepsilon T^*$. Then there exists an integer $k \geq 0$ such that $(1+\frac{\varepsilon}{3})^{k-1} L_{first} \leq L(\sigma^*) \leq (1+\frac{\varepsilon}{3})^k L_{first}$, where L_{first} is defined as in Lemma 2.3. Our scheme computes $A_\varepsilon((1+\frac{\varepsilon}{3})^k L_{first})$, which delivers a solution σ_2 with length at most $(1+\frac{\varepsilon}{3})^{k+1} L_{first} \leq (1+\frac{\varepsilon}{3})^2 L(\sigma^*) < (1+\varepsilon)L(\sigma^*)$ and cost at most $C_{opt}(L(\sigma^*))$.

To conclude, in either case our approximation algorithm outputs a schedule with cost at most $(1+\varepsilon)T^*$. □

Our final theorem summarizes the main result of this paper.

Theorem 2.6 *The single-machine problem with release dates, delivery times and compressible processing times with objective to minimize the maximal job delivery completion time possesses a PTAS.* □

References

[1] L.A. HALL. Approximability of flow shop scheduling. *Mathematical Programming 82*, 1998, 175–190.

[2] L.A. HALL AND D.B. SHMOYS, Approximation schemes for constrained scheduling problems, in *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, 1989, 134–140.

[3] L.A. HALL AND D.B. SHMOYS. Jackson's rule for single-machine scheduling: Making a good heuristic better. *Mathematics of Operations Research 17*, 1992, 22–35.

[4] D.S. HOCHBAUM AND D.B. SHMOYS. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM 34*, 1987, 144–162.

[5] J.K. LENSTRA, A.H.G. RINNOOY KAN, AND P. BRUCKER. Complexity of machine scheduling problems. *Annals of Discrete Mathematics 1*, 1977, 343–362.

[6] E. NOWICKI. An approximation algorithm for a single-machine scheduling problem with release times, delivery times and controllable processing times. *European Journal on Operations Research 72*, 1994, 74–81.

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

[8] L. SCHRAGE. Obtaining optimal solutions to resource constrained network scheduling problems. Unpublished manuscript, 1971.

[9] S. ZDRZALKA. Scheduling jobs on a single machine with release dates, delivery times and controllable processing times: Worst-case analysis. *Operations Research Letters 10*, 1991, 519–523.

Received May, 2002