Setting parameters for scheduling targets under emergent situations*

Jianhua Chen¹, Xiaoguang Yang^{2†}

¹ School of Economics, Anhui University, Hefei230039, Anhui, P. R. China

² Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing100080, P. R. China

(Received December 2 2006, Accepted January 9 2007)

Abstract. The parameter setting problems are to modify the initial parameters of a system optimally such that some pre-specific objective of the system can be reached. The principle behind parameter setting could be regarded as a target-first philosophy. The parameter setting problems can be found applications in the many real production systems, particularly in the emergency situation. In this paper, we consider the parameter setting problems for some classical single machine scheduling problems. We present $O(n^2)$ algorithms for the processing-time setting problem of $1||\sum C_i$, the processing-time setting problem of $1||T_{\max}$, the processing-time setting problem of $1|r_i|C_{\max}$, and the release time setting problem of $1|r_i|T_{\max}$ with fixed lower bounds and fixed unit cost, respectively. We also show that the due-time setting problem of $1||T_{\max}$ is NP-hard.

Keywords: parameter setting, scheduling target, emergent situation

1 Introduction

Emergency management has attracted more and more attention in recent years^[5]. When an emergent case happens, time is the most important issue. A task which can be done longer might be required to finish in a very short time under an emergent situation. To realize this goal, some parameters are required to be changed. For example, when a fire breaks out in a gym, people are required to be evacuated in 5 minutes by running while the normal exiting time might be half an hour by walking. As the aim of such problems is to determine the suitable parameters to reach a pre-specific targets under emergent situation, we may call these problems **parameter setting problems under emergent situation**. Naturally, such modification will lead the parameters deviate from their normal values which may incur some cost. Thus a cost function is associated with each changing. The parameter setting problems are to find a minimum cost modification to realize the given objective.

For the classical scheduling problems, the parameters of the discussed production systems are given, the task of scheduling is to assign the operations of jobs to each machine and to determine the starting time of each operation, with goal to optimize an objective function of the production system. The parameter setting problems for scheduling are required to modify the parameters of the production system such that some pre-specific objective can be reached, and the modification costs are minimum.

Scheduling is a basic management activity in operation management. When an emergent situation occurs, the normal schedule for job can not be implemented, moreover the normal production parameters have to be modified such that new targets have been reached. For example, an ordinary goal of production planning is to minimize the maximum tardiness of jobs; while in some emergency case, the optimal maximum tardiness with respect to original parameters might exceed the tolerance required in the emergent case. For instance, all

* The research is supported by National Key Research and Development Program of China (Grant No. 2002CB312004) and National Natural Science Foundation of China (Nos. 70425004700221001).

† Corresponding author. Tel.: +86-10-62651380; E-mail address: xgyang@iss.ac.cn.

jobs are required to be completed not later a fixed time than their due times. Thus it might need to modify the parameters of the production system such that the maximum tardiness of the production system will be compressed within the target value. Solving this parameter setting problem, we can obtain an emergency plan. These by setting different targets, we can get a set of emergency plans. If an emergent case happens, the decision-maker can choose one emergent plan according to his estimation for the target. Thus people can have a fast response to handle the emergent case.

A class of closely related scheduling problems is called the controllable (or compressible) scheduling problems, for example, see [1–3, 12–16]. Similar to our models, the parameters, such as processing times, setup times and release dates, are changeable with costs in those models. The difference between our models and their models is that the aim of their models is about the trade-off between the objective of scheduling and the cost incurred by changing parameters, while the aim of our model is to realize some given objective of scheduling with minimum cost.

In this paper, we consider parameter setting problems for some classical single machine scheduling models. The paper is organized as follows. In next section we present an $O(n^2)$ time algorithm for the processing-time setting problem of $1||T_{\max}$. In Section 3, we give an $O(n^2)$ time algorithm for the processing-time setting problem of $1||\sum C_i$. In Section 4, we design an $O(n^2)$ time algorithm for the processing-time setting problem of $1|r_i|C_{\max}$. In Section 4, we present an $O(n^2)$ time algorithm for the release time setting problem of $1|r_i|C_{\max}$ when the lower bounds of release times are same and the unit costs are identical. In Section 6, we show that the due-time setting problem of $1||T_{\max}$ is NP-hard.

2 Setting processing times for $1||T_{\max}$

The $1||T_{\max}$ problem is to sequence the independent jobs on a single machine to minimize the maximum tardiness, where associated with each job J_i are processing time p_i and due time d_i . The tardiness T_i of job J_i is defined as $\max\{C_i - d_i, 0\}$ with C_i denoted by completion time of job J_i . The problem is solved by the EDD (earliest due date) rule owe to Jackson^[7].

$1||T_{\max}$ is to find a schedule to minimize the maximum tardiness with respect to the normal processing times. But sometimes the maximum tardiness with respect to the normal processing times is not acceptable for the system. People may require that the maximum tardiness can not exceed a given bound. For example, all jobs must be finished before their due time. In order to achieve this goal, one way is to compress the processing times. This raises what we call the processing-time setting problem of $1||T_{\max}$. It can be stated as follows: Given a cost function c on the reduction of the initial processing time vector p , a lower bound vector l , and an upper bound U on the maximum tardiness, we try to modify the processing time vector from p to p^* such that $l_i \leq p_i^* \leq p_i$ for $1 \leq i \leq n$, $T_{\max}(p^*) \leq U$, and $\sum_{i=1}^n c_i(p_i - p_i^*)$ is minimum. Where $T_{\max}(p^*)$ stands for the maximum tardiness with respect to processing time vector p^* .

The feasibility of the processing-time setting problem is straightforward. Without loss of generality, we only consider the feasible instances. Here below, we present a polynomial algorithm to solve the problem.

Algorithm A:

Step 0. Sort the jobs according to EDD rule. Say $d_1 \leq d_2 \leq \dots \leq d_n$. Let $y_i = p_i$ for all $i \in I$. Let $T_i(y)$ denote the tardiness of job i under EDD rule with respect to processing time vector y . Thus $T_{\max}(y) = \max_{1 \leq i \leq n} T_i(y)$.

Step 1. If $T_{\max}(y) \leq U$, halt, $p^* = y$. Otherwise continue.

Step 2. Find i^* , the first job such that $T_{i^*}(y) = T_{\max}(y)$.

Step 3. Find i^+ , the job among $\{1, 2, \dots, i^*\}$ with minimum c_i such that $y_i > l_i$, i.e., $i^+ = \arg \min\{c_i \mid y_i > l_i, 1 \leq i \leq i^*\}$.

Step 4. Denote $T_{\max}^-(y)$ be the largest tardiness among jobs in $\{1, 2, \dots, i^+ - 1\}$ with respect to y . Let $\delta = \min\{T_{\max}(y) - U, T_{\max}(y) - T_{\max}^-(y), y_{i^+} - l_{i^+}\}$, update $y_{i^+} \leftarrow y_{i^+} - \delta$ and return Step 1.

Let us show the validity of Algorithm A.

First we claim that the updating at Step 4 is the optimal strategy to reduce $T_{\max}(y)$ by δ .

Obviously to reduce $T_{\max}(y)$ by δ , we have to shorten the sum of processing times of jobs in $\{1, 2, \dots, i^*\}$ by at least δ . We note that the tardiness of each job in $\{i^* + 1, i^* + 2, \dots, n\}$ is reduced by δ or becomes zero, depending on whether $T_i(y) \geq \delta$ or $T_i(y) < \delta$. Therefore we need not to shorten the processing times of jobs in $\{i^* + 1, i^* + 2, \dots, n\}$. As any over-reduction is unnecessary, we may assume that the total reduction of processing times of jobs in $\{1, 2, \dots, i^*\}$ is exactly δ .

Since c_{i^+} is the minimal one among $\{1, 2, \dots, i^*\}$ and $\delta \leq y_{i^+} - l_{i^+}$, let us shift all the reduction of processing times of jobs in $\{1, 2, \dots, i^*\}$ to the processing time of job i^+ . Clearly, this is the cheapest way to make the largest tardiness among job from i^* to n reduced by δ .

Further we can see that the maximum tardiness of all jobs is reduced exactly by δ too. In fact, by shortening p_{i^+} by δ , the completion times of all jobs in $\{i^+, i^+ + 1, \dots, n\}$ is reduced exactly by δ . As $T_{i^*}(y) = T_{\max}(y)$, the new tardiness of jobs in $\{i^+, i^+ + 1, \dots, n\}$ are still not greater than the new tardiness of job i^* . Moreover, as $\delta \leq T_{\max}(y) - T_{\max}^-(y)$, then the largest tardiness of jobs in $\{1, 2, \dots, i^+ - 1\}$ is also not greater than the new tardiness of job i^* . Hence the updating strategy at Step 4 is the cheapest way to reduce $T_{\max}(y)$ by δ .

Algorithm A is indeed a greedy algorithm. Next we prove that the algorithm converges globally. To this end, we need only show that y'' is the optimal solution to make $T_{\max}(y)$ be reduced by $\delta' + \delta''$ if $l \leq y' \leq y$ is the optimum to make $T_{\max}(y)$ be reduced by δ' and $l \leq y'' \leq y'$ is the optimum to make $T_{\max}(y')$ be reduced by δ'' .

In fact, write the cost of changing y to y' as $C(y, y')$ and the cost of changing y' to y'' as $C(y', y'')$, i.e. $C(y, y') = \sum_{i=1}^n c_i(y_i - y'_i)$ and $C(y', y'') = \sum_{i=1}^n c_i(y'_i - y''_i)$, we have $C(y, y'') = \sum_{i=1}^n c_i(y_i - y''_i) = C(y, y') + C(y', y'')$ since $l \leq y'' \leq y' \leq y$.

Now let $l \leq y^+ \leq y$ is a solution to make $T_{\max}(y)$ be reduced by $\delta' + \delta''$. It is clear that the tardiness can be reduced continuously, i.e. there exists y^- with $l \leq y^- \leq y^+$ such that y^- makes $T_{\max}(y)$ be reduced by δ' , and hence y^+ makes $T_{\max}(y^-) = T_{\max}(y) - \delta' = T_{\max}(y')$ be reduced by δ'' .

By the definitions of y' and y'' , we have

$$C(y, y^-) \geq C(y, y'), \quad C(y^-, y^+) \geq C(y', y'').$$

Hence we obtain that $C(y, y^+) \geq C(y, y'')$, i.e. y'' is the optimal solution to make $T_{\max}(y)$ be reduced by $\delta' + \delta''$.

In such a way, step by step, we know that Algorithm A produces an optimal solution at each iteration for total reduction of $T_{\max}(p)$ till this iteration.

Now let us show the algorithm will halt after finite iterations. At each iteration, we have three cases:

Case 1. $\delta = T_{\max}(y) - U$, then the algorithm stops after this iteration.

Case 2. $\delta = y_{i^+} - l_{i^+}$, then the reduction on job i^+ is saturated. Thus Case 2 happens at most n times.

Case 3. $\delta = T_{\max}(y) - T_{\max}^-(y)$. Let $Q(y) = \{i \mid T_i(y) = T_{\max}(y)\}$. Write the updated processing time vector as $\bar{y}$. Clearly $Q(y) \subset Q(\bar{y})$. As $\delta = T_{\max}(y) - T_{\max}^-(y)$, we have $T_{\max}(\bar{y}) = T_{\max}^-(y)$, thus there are some jobs entering into $Q(\bar{y})$, namely $Q(\bar{y}) \setminus Q(y) \neq \emptyset$. Therefore we also have Case 3 happens at most n time.

So we can conclude that Algorithm A has at most $2n$ iterations. Let us count the complexity of Algorithm A. As sorting in Step 0 can be done in $O(n \log(n))$, and finding $i^*, i^+, T_{\max}^-(y)$ can be done in $O(n)$ times, and the total number of iterations are at most $2n$, the total complexity of Algorithm B is $O(n^2)$. We conclude that

Theorem 1. The processing time setting problem of $1||T_{\max}$ can be solved in $O(n^2)$ times.

3 Setting processing times for $1||\sum C_i$

The $1||\sum C_i$ problem is to sequence the independent jobs on a single machine to minimize the total flow time. The total flow time is widely accepted as a good measurement of overall quality of service. It can also be interpreted as total work in-processing on the production line. When the optimal total flow time under normal

condition is beyond the real requirement, say the overall quality of service demanded by the public can not be satisfied under current condition, or the production line does not have enough buffer capacity to carry the work in-processing, it is natural to change the processing times to reach the goal. The processing-time setting problem of $1||\sum C_i$ arises from this situation.

Let $\{p_1, p_2, \dots, p_n\}$ the initial processing times of jobs, and denote $I = \{1, 2, \dots, n\}$ the job set. Given a target value $U > 0$, and a lower bound vector $l : I \rightarrow R_+$ such that $l_i \leq p_i$, and a cost vector $c : I \rightarrow R_+$, the processing-time setting problem of $1||\sum C_i$ is to find a modified processing time vector p^* such that

- (1) $\sum C_i(p^*)$, the total flow time with respect to p^* , is not greater than U ;
- (2) $l_i \leq p_i^* \leq p_i$, for $i = 1, 2, \dots, n$;
- (3) $\sum_{i=1}^n c_i(p_i - p_i^*)$ is minimized.

A well-known result by Smith^[17] states that:

Lemma 1. A permutation ω is an optimal permutation for $1||\sum C_i$ if and only if

$$p_{\omega(1)} \leq p_{\omega(2)} \leq \dots \leq p_{\omega(n)}.$$

Smith's scheduling strategy is called SPT (shortest processing time) rule. Obviously, the processing time setting problem for $1||\sum C_i$ is feasible if and only if the total completion times under SPT rule with respect to the lower bounds is not greater than U . Here below, we always assume that the problem is feasible and the optimal total completion time with respect to p is greater than U . That is, a reduction on the processing times is necessary. As any over-reduction is unnecessary, we will have $\sum C_i(p^*) = U$, where $C_i(y)$ stands for the completion time under SPT rule with respect to a processing time vector p^* .

To reduce the total completion time, we need to reduce the processing times of some jobs. Let us analyze how to do it economically.

Suppose that the current processing time vector is y , and $\delta > 0$ is a small amount we like to reduce for the total completion time. Intuitively, when δ is sufficient small, we need only reduce the processing time of one job to realize this goal, because we can always shift the reductions on other jobs to the job which incurs the least cost.

Note that by sorting the jobs in the non-decreasing order of y_i (without loss of generality, assume that the non-decreasing order is in the natural order, i.e., $y_1 \leq y_2 \leq \dots \leq y_n$), we call the group of jobs with equal processing times a *stair*. Consider one stair of processing times, say $\{y_i, y_{i+1}, \dots, y_j\}$ (implying that $y_{i-1} < y_i = \dots = y_j < y_{j+1}$.) Clearly, among a stair, the order of jobs is irrelevant to the total completion time.

Let us consider how to reduce the processing times of the jobs within this stair to make the total completion time reduced by δ . First, the jobs with $y_k = l_k$ in the stair have their lower bounds saturated, we can not reduce the processing times of those jobs further. So we need only to consider the jobs with $y_k > l_k$. If all the jobs in the stair have their lower bounds saturated, we can not shorten the total completion time by reducing the processing times of the jobs in this stair. Thus we need only consider the stairs in which there is a job such that $y_k > l_k$.

We always put the modified job at the first position in the stair, that is, job i is the job we will modify. As we have already supposed that δ is sufficient small, which we may assume that the needed reduction for a job is not greater than $y_i - y_{i-1}$. Thus after the reduction, the order of jobs with respect to the optimal sequence keeps unchanged. This makes us easy to compute the total completion after this modification.

Assume that we reduce the processing time of job i to shorten the total completion time by δ , then the reduction is $\frac{\delta}{n-i}$. Thus we can choose job i which has the smallest cost among the jobs with $y_k > l_k$ in the stair. Moreover, to make the reduction as large as possible, we like to choose job i with the smallest lower bound among the jobs with same cost.

To determine which stair should be processed first, we can define the 'cost' of a stair by the smallest cost of the job with $y_k > l_k$ within the stair, and define the 'cost' of a stair as $+\infty$ if all its jobs are saturated. Then we select the stair with the least 'cost'. To summarize, we can describe the above discussion formally as the following algorithm.

Algorithm B:

Step 0. Let $y_i = p_i$ for all $i \in I$.

Step 1. If $\sum C_i(y) \leq U$, halt, $p^* = y$. Otherwise let $\Delta = \sum C_i(y) - U$

Step 2. Sort $\{y_1, y_2, \dots, y_n\}$ in lexicographic order as follows:

first according to the non-decreasing order of value y_i ;

second according to the non-decreasing order of cost c_i ;

third according to the non-decreasing order of lower bound l_i .

Step 3. Partition $\{y_1, y_2, \dots, y_n\}$ into groups such that each group contains all jobs having same y values, say we have

$$\underbrace{y_1 = y_2 = \dots = y_{k_1}}_{<} < \underbrace{y_{k_1+1} = y_{k_1+2} = \dots = y_{k_2}}_{<} < \dots < \underbrace{y_{k_{m-1}+1} = y_{k_{m-1}+2} = \dots = y_{k_m}}_{<}$$

Step 4. Let $k_0 = 0$. For each group j from 1 to m , define the cost of the group c'_j in the following way: if $y_i = l_i$ for all jobs in the group, then $c'_j = +\infty$; otherwise let $j(i)$ be the first job in the group such that $y_{j(i)} > l_{j(i)}$, then define $c'_j = \frac{c_{j(i)}}{n - k_{j-1}}$.

Step 5. Find the group with minimum cost c'_j , say j^* .

Step 6. Compute

$$\delta = \min\left\{\frac{\Delta}{n - k_{j^*-1}}, y_{k_{j^*}} - y_{k_{j^*-1}}, y_{j^*(i)} - l_{j^*(i)}\right\}.$$

Step 7. Upgrade $y_{j^*(i)} \leftarrow y_{j^*(i)} - \delta$ and return Step 1.

Let us show that Algorithm B converges finitely. In fact, after each iteration, if the algorithm doesn't halt, either we have that $\delta = y_{j^*(i)} - l_{j^*(i)}$, thus job $j^*(i)$ is saturated; or we have that $\delta = y_{k_{j^*}} - y_{k_{j^*-1}}$, which indicates that job $j^*(i)$ enters the immediate lower stair. We may call the iteration with $\delta = y_{j^*(i)} - l_{j^*(i)}$ the first type iteration, and the iteration with $\delta = y_{k_{j^*}} - y_{k_{j^*-1}}$ the second type iteration. Obviously, the iteration of the first type occur at most n times. For the second type iterations, since $c'_{j^*} = \frac{c_{j^*(i)}}{n - k_{j^*-1}}$ is the smallest among c'_j and $k_{j^*-2} < k_{j^*-1}$, we have that job $j^*(i)$ is the job with the lowest cost after it enters the immediate lower stair. Moreover it is unique one, that is, there is no other job in the new stair which has same cost. Hence by the algorithm, we need to continue reducing the processing time of job $j^*(i)$.

Once again, Algorithm B is a greedy algorithm. Using the similar argument as last section, we can prove that Algorithm B terminates at the optimal solution.

Now let us analyze the complexity of Algorithm B. First, as there are at most $n - 1$ lower stairs before job $j^*(i)$, there are at most $n - 1$ second type iterations between any two first type iterations. Therefore the total number of iterations is $O(n^2)$. Second, we need to sort the jobs in lexicographic order and determine the 'cost' of each stair. Note that after each iteration, one job leaves the current working stair, and either the job itself becomes a new stair (that is $\delta = y_{j^*(i)} - l_{j^*(i)}$ and $l_{j^*(i)} > y_{k_{j^*-1}}$, and we define the 'cost' of this new stair as $+\infty$); or job $j^*(i)$ becomes the first job in the immediate lower stair, and the number of jobs of the current working stair is reduced by 1 and the the number of jobs of the current working stair is increased by 1. Hence we need only to introduce a dynamic index vector to point the first job with $y_i > l_i$ in each stair, and all the computation within each iteration can be done in $O(1)$ times. Only the sorting at the initial iteration is needed, and it can be done in $O(n \log(n))$ times. Hence we can conclude that

Theorem 2. The processing time setting problem of $1||\sum C_i$ can be solved in $O(n^2)$ times.

4 Setting processing times for $1|r_i|C_{\max}$

Associating each job J_i with a processing time p_i and a release time r_i for $i \in I$ where $I = \{1, 2, \dots, n\}$ is the set of jobs, $1|r_i|C_{\max}$ problem is to arrange the starting time for each job, which can not be earlier than its release time, such that the maximum completion time $C_{\max}$ is minimum. It is trivial to see that we need only to sort the jobs according to their release times in the non-decreasing order, say the jobs are ordered in $r_1 \leq r_2 \leq \dots \leq r_n$, and let $C_0 = 0$, then set $S_{i+1} = \max\{C_i, r_{i+1}\}$ and $C_{i+1} = S_{i+1} + p_{i+1}$. We have $C_{\max} = C_n$.

For any subset $I' \subset I$, and any processing time vector p' , denote $C_{\max}(I', p')$ be the optimal maximum completion time of the sub-instance I' with processing time p' . Now we define the processing time setting problem of $1|r_i|C_{\max}$, which is to modify the processing times p_i to p'_i such that $l_i \leq p'_i \leq p_i$, the maximum completion time $C_{\max}(I', p')$ under p' satisfies $C_{\max}(I', p') \leq U$ for some given value U , and the total cost $\sum_{i=1}^n c_i(p_i - p'_i)$ is minimum. Where l_i is the lower bound of processing time, and c_i is the unit cost of the reduction. Obviously, the problem is feasible if the maximum completion time is not greater than U when the processing times of all jobs reach their lower bounds. Here below we assume that any given instance is a feasible one.

To solve the problem, let us group jobs according to their release times. Suppose that there are $m(\leq n)$ different release times, say $r_{[1]} < r_{[2]} < \dots < r_{[m]}$. We write S_j be the subset of jobs which have release times $r_{[j]}$. For any $1 \leq k \leq m$, we use I_k to denote the sub-instance of $1|r_i|C_{\max}$ which contains the first k groups of jobs, i.e., $I_k = \bigcup_{j=1}^k S_j$.

When $C_{\max}(I_k, \hat{p}) \leq r_{[k+1]}$, a direct observation is that there is no contribution to shorten the maximum completion time of whole instance if we reduce the processing times of jobs in I_k but keep the processing times in $I \setminus I_k$ unchanged. Therefore, we need only consider reduce the processing times of jobs in $I \setminus I_k$ for the largest k such that $C_{\max}(I_k, \hat{p}) \leq r_{[k+1]}$.

It is straightforward to see that the cheapest way to shorten the maximum completion time by a small amount is to reduce the processing time of the cheapest job which has not reach its lower bound.

Consider the largest possible reduction by a job i . First it can not be reduced beyond its lower bound. Second, it is not necessary to reduce it further if the completion time of one of sub-instance including this job reaches its next release time.

Based on all above observations, we can design an algorithm as follows to solve the processing time setting problem of $1|r_i|C_{\max}$.

Algorithm C:

Step 1. Group the jobs according to their release times, and sort the groups in the increasing order, say $r_{[1]} \leq r_{[2]} \leq \dots \leq r_{[m]}$. Let $r_{[m+1]} = U$, and $p'_i = p_i$ for all $i \in I$.

Step 2. Find the largest k such that $C_{\max}(I_k, p') \leq r_{[k+1]}$. If $k = m$, halt.

Step 3. Find a job i , such that $i = \arg \min\{c_i \mid i \in I \setminus I_k, \text{ and } p'_i > l_i\}$. Assume that job i belongs to group S_h . For each group j from h to m , let $\Delta_j = C(I_j, p') - r_{[j+1]}$.

Step 4. Let $\delta = \min\{p'_i - l_i, \min\{\Delta_j \mid h \leq j \leq m\}\}$.

Step 5. Update p'_i by $p'_i - \delta$, and go to Step 2.

At each iteration, either the job i reaches its lower bound, or there is some sub-instance I_j ($j \geq h > k$) such that $C_{\max}(I_j, p') \leq r_{[j+1]}$ with respect to the new p' . Hence there are at most $2n$ iteration.

Consider the computation in each iteration. It is not difficult to see that grouping jobs can be done in $O(n)$ time, sorting groups can be done in $O(n \log(n))$ times, finding the cheapest job can be done in $O(n)$, computing all $C_{\max}(I_j, p')$ and Δ_j can be done in $O(n)$ time too. Hence we obtain that

Theorem 3. Algorithm C solves the processing time setting problem of $1|r_i|C_{\max}$ in $O(n^2)$ times.

5 Setting release times for $1|r_i|C_{\max}$ with equal lower bounds and unit costs

In this section, we consider the release time setting problem of $1|r_i|C_{\max}$, which keeps the processing times unchanged but try to modify the release times to make the maximum completion time no longer than a given value U under modified r' such that the total modification is minimum. In the sequel, we only consider the simple version that all release times are allowed to be reduced to a common lower bound $\underline{r}_i = \underline{r}$, and the unit costs of the reduction for all jobs are same, i.e., $c_i = c$ for all $i \in I$. It is easy to see that the problem is feasible if only $\underline{r} + \sum_{i \in I} p_i \leq U$.

For a given release time vector $\hat{r}$, we group jobs according to the different release times, say $\hat{r}_{[1]} < \hat{r}_{[2]} < \dots < \hat{r}_{[m(\hat{r})]}$. We write $S_j(\hat{r})$ be the subset of jobs which have release times $\hat{r}_{[j]}$. For any $1 \leq k \leq m(\hat{r})$,

we use $I_k(\hat{r})$ to denote the sub-instance of $1|\hat{r}_i|C_{\max}$ which contains the first k groups of jobs, i.e., $I_k(\hat{r}) = \bigcup_{j=1}^k S_j(\hat{r})$. Denote $C_{\max}(I_k(\hat{r}))$ be the optimal maximum completion time of the sub-instance $I_k(\hat{r})$.

Now let us analyze how to compress the maximum completion of $1|\hat{r}_i|C_{\max}$ further.

As the processing times are not able to be shortened, if there are no idle time slots from $\underline{r}$ to $C_{\max}(I_{m(\hat{r})}(\hat{r}))$, we have $C_{\max}(I_{m(\hat{r})}(\hat{r})) = \underline{r} + \sum_{i \in I} p_i$. Thus the maximum completion time of $1|\hat{r}_i|C_{\max}$ can not be compressed further. Therefore, if $C_{\max}(I_{m(\hat{r})}(\hat{r}))$ is compressible, then there must exist some idle time slots between $\underline{r}$ and $C_{\max}(I_{m(\hat{r})}(\hat{r}))$, that is, there exists some $1 \leq k \leq m(\hat{r})$ such that $C_{\max}(I_k(\hat{r})) < \hat{r}_{[k+1]}$.

Let k be the largest index such that $C_{\max}(I_k(\hat{r})) < \hat{r}_{[k+1]}$. Clearly, if we only reduce the release times in $I_k(\hat{r})$, but keep the release times in $I \setminus I_k(\hat{r})$ untouched, the maximum completion time $C_{\max}(I_{m(\hat{r})}(\hat{r}))$ will not change. Therefore, we need to reduce the release times of jobs in $I \setminus I_k(\hat{r})$ in order to reduce the maximum completion time.

Suppose Δ , the reduction amount of the maximum completion time of whole instance satisfying that $\Delta \leq \hat{r}_{[k+1]} - C_{\max}(I_k(\hat{r}))$, then we need only to reduce the release times in $I \setminus I_k(\hat{r})$ to realize this goal. Let us consider how to realize this goal cheaply. First as $\underline{r}$ is a common lower bound for all release times, it is feasible to reduce the release time of any job in $I \setminus I_k(\hat{r})$. Second as the unit costs are same, it is not difficult to deduce by the interchange technique that the best strategy is to reduce the release time of a job which has smallest $\hat{r}_i$ in $I \setminus I_k(\hat{r})$ by Δ , and then reduce the release times in the order of $\hat{r}$ from small to big for other jobs in $I \setminus I_k(\hat{r})$ such that the undated release time is equal to the smaller one between the completion time of its job and the its current release time.

If the the remotest idle time slot $\hat{r}_{[k+1]} - C_{\max}(I_k(\hat{r}))$ is big enough to cover the difference between the current maximum completion time and the target value U , the above procedure has already produced an optimal solution. If not, we go to process next remote idle time slot. In this way, we can generate an optimal solution in the end.

To summarize, we have following algorithm to solve the release time setting problem of $1|r_i|C_{\max}$ when all the release times have a common lower bound, and the unit modification costs are equal.

Algorithm D:

Step 1. Sort the job in the non-decreasing order of their initial release time. Let $r' = r$. Group the jobs according to the different values of r'_i , and sort the groups in the increasing order, say $r'_{[1]} < r'_{[2]} < \dots < r'_{[m(r)]}$. Let $C_{\max}(I_0(r')) := \underline{r}$, where $I_0(r') := \emptyset$.

Step 2. If $C_{\max}(I_{m(r')}(r')) \leq U$, halt.

Step 3. Find the largest k such that $C_{\max}(I_k(r')) < r'_{[k+1]}$. If such k does not exist, stop. The instance is not feasible.

Step 4. Let i be the first job such that $r'_i = r'_{[k+1]}$. Let $\delta = \min\{C_{\max}(I_{m(r')}(r')) - U, r'_{[k+1]} - C_{\max}(I_k(r'))\}$.

Step 5. Update $r'_i = r'_i - \delta$. For $h = i$ to n let $C_h = C_h - \delta$, and for $h = i$ to n let $r'_h = \min\{C_{h-1}, r'_h\}$.

Step 6. Generate new groups, and go to Step 2.

After each iteration, either the one idle time slot disappears, or the algorithm will terminates. As there are at most $n - 1$ idle time slots, the number of iterations is at most $n - 1$.

Consider the computation in each iteration. It is not difficult to see that sorting jobs and groups can be done in $O(n \log(n))$ times, computing new completion times C_h and updating r'_h for $h \in I \setminus I_k(r')$ can be done in $O(n)$ time too. Note that the order of release times keeps untouched during all iterations, hence we need only one sorting operation for jobs. Moreover, new groups only grow out from $I \setminus I_k(r')$, and its order is same as the order of jobs, hence generating the new groups and sorting them can be done in $O(n)$ time. Therefore we conclude that

Theorem 4. Algorithm D solves the release time setting problem of $1|r_i|C_{\max}$ in $O(n^2)$ times if $\underline{r}_i = \underline{r}$ and $c_i = c$ for all $i \in I$.

6 Setting due times for $1||T_{\max}$

The due time setting problem of $1||T_{\max}$ can be stated as follows: Given a cost function w on the increment of due time d , given an upper bound U on the maximum tardiness $T_{\max}$, we try to modify the due time d with least cost such that the maximum tardiness of new instance doesn't exceed U .

In this section, we show that the due time setting problem of $1||T_{\max}$ is NP-hard. First we show that it is strongly NP-hard when there are different costs on the modification of due times involved in the problem. The proof is based on a polynomial reduction from $1||\sum w_i T_i$, which is strongly NP-hard by Lawler^[2], to the due time setting problem of $1||T_{\max}$.

Theorem 5. *The due time setting problem of $1||T_{\max}$ is strong NP-hard if the different costs on the modification of due times involved in.*

Proof. Let $\mathcal{J} = (p, d, w, B)$ be an instance of decision problem of $1||\sum w_i T_i$, where p is the vector of processing times, d is the vector of due times, w is the weight vector on tardiness, and B is an upper bound on the total weighted tardiness.

We construct an instance $\mathcal{J}^+$ of the decision problem of the due time setting problem of $1||T_{\max}$ as follows: the vectors of the processing times and due times are same as the instance of $1||\sum w_i T_i$, w is the cost vector of modification of due times, $U = 0$ (that is, we require to modify due times such that all jobs can be finished before the modified due times), and B is the upper bound on the total modification cost.

We claim that the instance $\mathcal{J}$ has a feasible solution if and only if the instance $\mathcal{J}^+$ has a feasible solution.

In fact, suppose that $\mathcal{J}$ has a feasible solution, i.e. there is a schedule such that $\sum w_i T_i \leq B$. Let us define a modified due time vector $d'_i = d_i + T_i$. Clearly by the definition T_i , we have $C_i \leq d_i + T_i = d'_i$. Namely each job i can be finished before the modified due time. Moreover $\sum w_i (d'_i - d_i) = \sum w_i T_i \leq B$. Thus we obtain a feasible solution of $\mathcal{J}^+$.

On the contrary, suppose that $\mathcal{J}^+$ has a feasible solution, i.e. there is a modified due time vector $d' \geq d$ such that $C_i \leq d'_i$ for each i and $\sum w_i (d'_i - d_i) \leq B$. Then we have $\sum w_i T_i = \sum w_i \max\{C_i - d_i, 0\} \leq \sum w_i (d'_i - d_i) \leq B$. Thus we get a feasible solution for $\mathcal{J}$. The proof is finished.

Remark 1. From the proof, it is clear that an instance of $1||\sum T_i$ can be transformed into an instance of the due time setting problem of $1||T_{\max}$ with $w_i = 1$ for each job. Since $1||\sum T_i$ is NP-hard by Du and Leung^[2], we can conclude the due time setting problem of $1||T_{\max}$ with $w_i = 1$ for each job is also NP-hard.

7 Concluding remarks

In this paper, we discuss the parameter setting problems under emergent situations for some classical scheduling models. We present polynomial algorithms for three processing-time setting problems, and one release time setting problem. We also show one due-time setting problem is NP-hard. But we failed to solve the release time setting problem of $1|r_i|C_{\max}$ if the lower bounds are different or the different unit costs are involved. We like to note that the scheduling models discussed in this paper are simplest ones among scheduling models, and the modification strategy under our consideration is relatively simple in which we only consider changing one type of parameters. Studying the parameter setting problems for more complicated scheduling models, or studying the parameter setting problems with jointly modification strategy, say not only changing processing times but also changing release times, might be challenge since the scheduling models do not have a good combinatorial characterization as the network models in general. We believe that the research on this subject has a big potential.

References

- [1] Z. L. Chen, Q. Lu, G. Tang. Single machine scheduling with discretely control lable processing times. *Operations Research Letters*, 1997, **21**: 69–76.

- [2] T. Cheng, C. Oguz, X. Qi. Due-date assignment and single machine scheduling with compressible processing times. *International Journal of Production Economics*, 1996, **43**: 29–35.
- [3] T. Cheng, N. Shakhlevich. Single machine scheduling of unit-time jobs with controllable release dates. *Journal of Global Optimization*, 2003, **27**: 293–311.
- [4] J. Du, J. Leung. Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research*, 1990, **15**: 483–495.
- [5] A. Farazmand, A. F. Farazmand. *Handbook of Crisis and Emergency Management*. Marcel Dekker, 2001.
- [6] T. Gonzalez, S. Sahni. Open shop scheduling to minimize finish time. *SIAM Journal*, 1977, **6**: 155–166.
- [7] J. R. Jackson. Scheduling a production line to minimize maximum tardiness, research report. **in**: *Management Science Research Project*, vol. 43, Los Angeles, 1954. University of California.
- [8] S. M. Johnson. Optimal two-and-three-stage production schedules. *Naval Research Logistics Quarterly*, 1954, **1**: 61–68.
- [9] E. L. Lawler. A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. *Ann. Discrete Math.*, 1977, **1**: 331–342.
- [10] E. L. Lawler, J. K. Lenstra, et al. Sequencing and scheduling: algorithms and complexity. *Handbooks in Operation Research and Management Science*, 1993.
- [11] J. K. Lenstra, A. Kan, P. Brucker. Complexity of machine scheduling problems. *Ann. Discrete Math.*, 1977, **1**: 343–362.
- [12] C. Ng, X. Cai. A new model for completion time variance problem with job-dependent weights and controllable processing times. *International Journal of Modelling and Simulation*, 1997, **17**: 306–309.
- [13] C. Ng, T. Cheng, et al. Single machine scheduling with a variable common due date and resource-dependent processing times. *Computers and Operations Research*, 2003, **30**: 1173–1185.
- [14] C. Ng, T. Cheng, M. Kovalyov. Batch scheduling with controllable setup and processing times to minimize total completion time. *Journal of the Operational Research Society*, 2003, **54**: 499–506.
- [15] X. Qi, G. Yin, J. R. Birge. Single-machine scheduling with randomly compressible processing times. *Stochastic Analysis and Applications*, 2002, **20**: 591–613.
- [16] X. D. Qi, G. Yin, J. R. Birge. Single-machine scheduling with random machine breakdowns and randomly compressible processing times. *Stochastic Analysis and Applications*, 2000, **18**: 635–653.
- [17] W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 1956, **3**: 59–66.