Theoretical Computer Science 412 (2011) 4536–4544

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Tight complexity analysis of the relocation problem with arbitrary release dates

Sergey V. Sevastyanov [a], Bertrand M.T. Lin [b,*], Hsiao-Lan Huang [b]

[a] Sobolev Institute of Mathematics, Novosibirsk State University, Novosibirsk, Russia

[b] Institute of Information Management, Department of Information and Finance Management, National Chiao Tung University, Hsinchu, Taiwan

ARTICLE INFO

Article history:
Received 2 May 2010
Received in revised form 20 February 2011
Accepted 21 April 2011
Communicated by G. Ausiello

Keywords:
Relocation problem
Resource constraints
Release dates
Makespan
NP-hardness
Multi-parametric dynamic programming

ABSTRACT

The paper considers makespan minimization on a single machine subject to release dates in the relocation problem, originated from a resource-constrained redevelopment project in Boston. Any job consumes a certain amount of resource from a common pool at the start of its processing and returns to the pool another amount of resource at its completion. In this sense, the type of our resource constraints extends the well-known constraints on resumable resources, where the above two amounts of resource are equal for each job. In this paper, we undertake the first complexity analysis of this problem in the case of arbitrary release dates. We develop an algorithm, based on a multi-parametric dynamic programming technique (when the number of parameters that undergo enumeration of their values in the DP-procedure can be arbitrarily large). It is shown that the algorithm runs in pseudo-polynomial time when the number m of distinct release dates is bounded by a constant. This result is shown to be tight: (1) it cannot be extended to the case when m is part of the input, since in this case the problem becomes strongly NP-hard, and (2) it cannot be strengthened up to designing a polynomial time algorithm for any constant $m > 1$, since the problem remains NP-hard for $m = 2$. A polynomial-time algorithm is designed for the special case where the overall contribution of each job to the resource pool is nonnegative. As a counterpart of this result, the case where the contributions of all jobs are negative is shown to be strongly NP-hard.

1. Introduction

Scheduling and planning in an organization are decision-making processes that deploy mathematical analysis and computing methods to allocate limited resources to optimize the objectives and goals of the organization [17]. Resource constraints are one of the most commonly addressed critical issues in project scheduling (see for example [15,7,2]). This paper studies a single-machine scheduling problem, called *the relocation problem*, in which resource constraints are involved. The resource constraints in the relocation problem differ from the traditional (resumable) resource constraints for scheduling problems [1,6] with regard to the amount of resource released by a completed activity. In the relocation problem, the amount of resource returned at the completion of an activity is independent of the amount of resource that the activity demands and consumes for starting its processing.

Formally, we consider a set of jobs $N = \{1, \ldots, n\}$ to be processed on a single machine. Initially, Q_0 units of the resource are provided in a common resource pool. Each job $i \in N$ has a release date r_i and a processing time p_i, requires α_i units of

* Corresponding author. Tel.: +886 35131472.
E-mail addresses: bmtlin@mail.nctu.edu.tw, bmtlin@iim.nctu.edu.tw (B.M.T. Lin).

0304-3975/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2011.04.034

the resource to commence its processing, and returns β_i units of the resource back to the common resource pool when it is completed. Without loss of generality, we may assume that the values of all these parameters are nonnegative. The resource that job i acquires is immediately consumed when its processing starts. The machine can process at most one job at a time, and no preemption is allowed. Thus, job i can be processed only if (1) its release date is reached, (2) the resource level in the pool is sufficient for the requirement α_i, and (3) the machine is not occupied. All parameters cited in this paper are nonnegative integers, except for the contributions of jobs that will be defined later and may be of an arbitrary sign. Given an initial level of the resource pool, a schedule is called *feasible* if it satisfies the release-date constraints and no job is blocked due to insufficiency of the resource. The problem is to determine a feasible schedule with the minimum makespan ($C_{\max}$).

Adopting the three-field notation [5], we use $\langle 1 \mid rp, r_i \mid C_{\max}\rangle$ to denote the problem under study. The notation dictates that we consider a single-machine scheduling problem with the minimum makespan objective. In the second field, rp specifies the resource constraints of the relocation problem, and r_i specifies the release-date constraints.

The relocation problem was proposed and formulated by Kaplan and his colleagues [9,16,11] for a redevelopment project in Boston. In the redevelopment area, there was a set of old buildings to demolish and new buildings to erect within the same area. During the redevelopment session, tenants of the old buildings needed to be evacuated and temporarily housed. The authority provided an initial budget for the temporary housing units. The project team wanted to determine a redevelopment sequence of the buildings such that all tenants could be successfully housed during the project session. In its basic setting, the relocation problem considered only Q_0, $\{\alpha_i\}$ and $\{\beta_i\}$, and sought for a feasible redevelopment sequence subject to an initial resource level. An optimization counterpart of the problem is to find the minimum initial resource level, called *resource requirement*, that guarantees the existence of a feasible sequence.

The new problem setting studied in this paper is motivated by the following scenario. Each job in the relocation problem corresponds to a sub-region undergoing an independent reconstruction. It contains a set of old buildings that have to be demolished. As a second phase of the reconstruction job, another set of new buildings should be erected within the same sub-region. It is assumed that both phases of this process have to be performed by the same team (associated in our model with the single machine), and that the team can move to another sub-region and start another job only after the whole reconstruction job within one sub-region is completed. The release date of a job is due to an expiration date before which the corresponding sub-region has certain functions and cannot be demolished. The expiration dates might be distinct for different sub-regions. We thereby come to a more general problem setting with distinct release dates, suggesting more potential applications in practice.

The study of the relocation problem started in the 1980s. However, it did not draw much research attention. Kaplan and Amir [10] established the equivalence of the resource requirement minimization problem to the classical two-machine flowshop scheduling problem of makespan minimization [8]. This equivalence gives rise to a new interpretation of two-machine flowshop scheduling and of Johnson's algorithm in terms of the relocation problem. Such an interpretation is more intuitive than those which could be found in most textbooks. For potential applications of the problem equivalence, please refer to [3]. In addition to the redevelopment project, the relocation problem exhibits applications in other areas such as memory management [10] and financial planning [18]. In the interest of theoretical development, the relocation problem also suggests many new challenging research problems. The relocation problem with multiple parallel machines to minimize makespan was addressed by Kaplan [9] and Kaplan and Amir [10], and its complexity remained open until Kononov and Lin [12] showed that the problem is strongly NP-hard even when there are only two machines and $\alpha_i \leq \beta_i, p_i = 1$ hold for all jobs. Kononov and Lin [12] proposed approximation algorithms and analyzed their ratio performance in the worst case. A single-machine problem to minimize total weighted completion time was studied by Kononov and Lin [13]. They established NP-hardness of the problem and designed two approximation algorithms with ratio performance guarantees. Lin and Liu [14] investigated the relocation problem with generalized due dates to maximize the total rewards, defined in terms of installments. They proved the problem to be strongly NP-hard and gave lower bounds and dominance rules for the development of branch-and-bound algorithms.

This paper starts the research on the relocation problem with release-date constraints and proposes the following results. First, we establish some basic properties of feasible and optimal sequences of jobs for a given instance of our problem. Based on those properties, a combinatorial algorithm is designed for the general case. We next transform the algorithm into a dynamic program with $3m$ independent dynamic parameters, where m is the number of distinct release dates. Thus, when the number of distinct release dates m is bounded by a constant, the DP-algorithm runs in pseudo-polynomial time. As a negative counterpart, it is proved that for variable m the problem becomes strongly NP-hard. Thus, no pseudo-polynomial time algorithm may exist for the case of variable m, unless $P = NP$. Furthermore, it is shown that our result cannot be improved up to a polynomial time algorithm for a constant m, because even for $m = 2$ the problem remains NP-hard (which is proved in Section 5). Next, the problem is investigated with respect to positive/negative contributions $\delta_i \doteq \beta_i - \alpha_i$ of jobs to the resource pool. We show that the case when all jobs have nonnegative contributions is polynomially solvable, and that the opposite case when all jobs have negative contributions is strongly NP-hard.

The remainder of this paper is organized as follows. Section 2 introduces notation and several preliminary results that will be used in the subsequent sections. In Section 3, we design an exact combinatorial algorithm for solving the general problem. Section 4 is dedicated to the development of the dynamic programming approach. In Section 5, we discuss the complexity of two special cases: when all jobs have nonnegative contributions and when all jobs have negative contributions. Concluding remarks are presented in Section 6.

2. Notation and preliminary results

In this section, for the convenience of the reader, we introduce the notation used in our discussion and formulate several preliminary properties of feasible and optimal solutions.

$\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_{|N|})$ a permutation of jobs in set N,
$s_i(S)$, $c_i(S)$ the starting and completion times of job i in schedule S.

Let σ be a permutation of jobs in $N' \subseteq N$. Kaplan and Amir [10] established that the resource requirement of sequence σ is equal to $R(\sigma) \doteq \max_{j=1,\ldots,|N'|} \left(\sum_{i=1}^{j} \alpha_{\sigma_i} - \sum_{i=1}^{j-1} \beta_{\sigma_i} \right)$. A sequence σ is called *feasible*, if $R(\sigma) \leq Q_0$.

We note that the *feasibility of a sequence* of jobs is defined exclusively with respect to the resource constraints, while temporal parameters, such as job release dates and processing times, are not involved. On the contrary, the *feasibility of a schedule* should take into account both the release-date constraints and the resource constraints.

Given a sequence σ of jobs in N, S_σ will denote the corresponding *active schedule*, *i.e.*, the one in which each job is processed at the earliest possible time among all feasible schedules specified by σ. Clearly, if σ is a feasible sequence, then S_σ is a feasible schedule, and if σ is an optimal sequence (at which the minimum makespan is attained), then S_σ is an optimal schedule. Thus, it suffices to consider only active schedules.

We next present some known results concerning the properties of solutions to the basic relocation problem with $r_j = 0$ for all jobs $j \in N$.

Lemma 1 (*Kaplan and Amir [10]*). *Given an instance I of problem $\langle 1 \mid rp \mid C_{\max} \rangle$ and a permutation of jobs σ, the resource requirement $R(\sigma)$ is equal to the total idle time of the second machine in the active permutation schedule S_σ for the two-machine flowshop problem, using α_i and β_i as the processing times of job $i \in N$ on the first machine and the second machine, respectively.*

Thus, Lemma 1 establishes a one-to-one correspondence between the amount of resource required by a job permutation σ in problem $\langle 1 \mid rp, r_i \mid C_{\max} \rangle$, on the one hand, and the length of the active permutation schedule S_σ in the corresponding $\langle F2 \mid\mid C_{\max} \rangle$ problem, on the other hand: the more is the length of schedule S_σ in the two-machine flowshop scheduling problem $\langle F2 \mid\mid C_{\max} \rangle$, the more resource is required by the permutation σ in the relocation problem.

As is well known, the $\langle F2 \mid\mid C_{\max} \rangle$ problem of minimizing the schedule length was solved by Johnson [8] in polynomial time. As follows from his result, the following sequence of jobs can be taken for the optimal one: the jobs with $\delta_i \geq 0$ are processed first and sequenced in non-decreasing order of α_i, while the jobs with $\delta_i < 0$ are sequenced after them in non-increasing order of β_i. Such an order of jobs will be further referred to as *Johnson's order*, or *Johnson's permutation* of jobs. This permutation can be found in $O(n \log n)$ time.

It follows from Johnson's result and Lemma 1 that the job permutation that requires the minimum amount of resource in problem $\langle 1 \mid rp, r_i \mid C_{\max} \rangle$ can also be found in polynomial time — it suffices to sequence the jobs in Johnson's order. In other words, if for a given instance of problem $\langle 1 \mid rp, r_i \mid C_{\max} \rangle$ there exist feasible permutations of jobs, then one of such permutations can be found by applying Johnson's rule. And since in the case of **identical release dates** r_j all feasible permutations of jobs define active schedules of the same length, this implies that any feasible permutation can be taken for an optimal one. In particular, Johnson's permutation is optimal for problem $\langle 1 \mid rp \mid C_{\max} \rangle$, unless there are no feasible permutations at all (which was also observed by Kaplan and Amir [10]).

In the general case, Johnson's rule does not provide an optimal solution for problem $\langle 1 \mid rp, r_i \mid C_{\max} \rangle$. However, for the convenience of our further argumentation, we will assume that **the jobs are indexed in Johnson's order**, unless otherwise stated. We will say that two jobs σ_i, σ_j $(i < j)$ are arranged in a given sequence σ in *anti-Johnson's order*, if $\sigma_i > \sigma_j$.

As an auxiliary result, Johnson observed [8] that if in some permutation of jobs σ two consecutive jobs $i = \sigma_k$, $j = \sigma_{k+1}$ are arranged in anti-Johnson's order with respect to each other, then their transposition does not increase the length of schedule S_σ in problem $\langle F2 \mid\mid C_{\max} \rangle$. In terms of the relocation problem, this yields the following

Remark 2. Let σ be a feasible sequence of jobs in problem $\langle 1 \mid rp, r_i \mid C_{\max} \rangle$ such that two consecutive jobs $i = \sigma_k$, $j = \sigma_{k+1}$ stand in anti-Johnson's order, while $r_j \leq s_i(S_\sigma)$. Then the transposition of jobs i and j yields also a feasible sequence of jobs σ', and the length of the active schedule $S_{\sigma'}$ is not greater than that of schedule S_σ.

In the studied problem we assume that there are m distinct release dates: $r_{(1)} < r_{(2)} < \cdots < r_{(m)}$. Due to Remark 2, we can always re-arrange a given schedule so that the jobs starting within the interval $[r_{(i)}, r_{(i+1)})$ (for each $i = 1, \ldots, m$, where $r_{(m+1)} = \infty$) are ordered by Johnson's rule. Thus, we may restrict ourselves by considering only such job permutations σ that can be represented as a concatenation of m sub-intervals, each being ordered by Johnson's rule:

$$\sigma = \sigma_{(1)} \oplus \cdots \oplus \sigma_{(m)}. \tag{1}$$

Such sub-intervals will be referred to as *J-blocks*. This idea provides a basis for a combinatorial algorithm that will be described in the next section.

In the remainder of this section we prove a heredity property of feasible sequences of jobs in problem $\langle 1 \mid rp, r_i \mid C_{\max} \rangle$. Given a sequence σ of jobs indexed by Johnson's rule, we will use σ^k to denote the subsequence of σ obtained by deleting the jobs $k+1, \ldots, n$ from σ.

Lemma 3. *Let σ be a feasible sequence of jobs $\{1, \ldots, K\}$ for $K \leq n$. Then, every subsequence σ^k $(k \leq K)$ is also feasible.*

Proof. The proof is based upon induction on decreasing k. Suppose that a subsequence σ^k, $2 \le k \le K$ of sequence σ is feasible. Let us show that so is subsequence σ^{k-1}. Suppose that job k is not the last job in sequence σ^k. By Remark 2, the transpositions of job k toward the last position of σ^k provide a feasible sequence $\hat{\sigma}^k = \sigma^{k-1} \oplus (k)$ of jobs $\{1, \ldots, k\}$. Since, next, every prefix of a feasible sequence is feasible, σ^{k-1} is feasible. □

3. Exact algorithm $\mathcal{A}_1$

This section presents an exact algorithm for the studied problem. Algorithm $\mathcal{A}_1$ generates a pool Σ_n of candidate sequences of jobs $\{1, \ldots, n\}$ that should contain at least one optimal sequence, if it exists. The pool is constructed iteratively by increasing $k = 1, \ldots, n$, where at step k we create the pool Σ_k of some feasible sequences of jobs $\{1, \ldots, k\}$ by augmenting the existing sequences from pool Σ_{k-1} with the newly inserted job k. In addition, it is assumed that each sequence $\sigma \in \Sigma_k$ is represented as a concatenation of its J-blocks in the form of (1). The algorithm consists of $n + 2$ steps.

Step 0. Define pool Σ_0 that consists of the single *null* permutation $\sigma^0 = \sigma_{null}$, which has only *null* J-blocks $\sigma^0_{(i)}$.

Steps $k = 1, \ldots, n$. Pool Σ_k being formed at step k consists of all feasible sequences of jobs generated from any sequence $\sigma' \in \Sigma_{k-1}$ by adding job k to the end of any J-block $\sigma'_{(i')}$ with number $i' \ge i(k)$, where the function $i(k)$ is defined by the equality $r_{(i(k))} = r_k$. In addition, job k is added to a J-block $\sigma'_{(i')}$ with number $i' > i(k)$, only if it is non-empty.

Step $n + 1$. If $\Sigma_n = \varnothing$ then {Output "No feasible schedules"; **STOP**} else select from pool Σ_n the sequence σ_{opt} with the minimum value of the objective function. □

The next lemma directly follows from the description of the algorithm.

Lemma 4. *At each step, $k = 0, \ldots, n$ any sequence $\sigma \in \Sigma_k$ meets the following properties (further referred to as "properties $(*)$"):*
$(a*)$ σ is feasible;
$(b*)$ σ can be represented as a concatenation $\sigma = \sigma_{(1)} \oplus \cdots \oplus \sigma_{(m)}$ of m J-blocks (i.e., increasing sequences of job indices) some of which may be empty;
$(c*)$ each non-empty J-block $\sigma_{(i)}$ of σ consists of jobs $\{j\}$ with $r_j \le r_{(i)}$ and starts with job j such that $r_j = r_{(i)}$. □

To justify the optimality of the algorithm, we need to prove two more lemmas.

Lemma 5. *If for a given instance of problem $\langle 1 \mid rp, r_i \mid C_{\max}\rangle$ there are feasible sequences of jobs, then there exists an optimal sequence that meets properties $(*)$.*

Proof. Suppose that for a given instance of problem $\langle 1 \mid rp, r_i \mid C_{\max}\rangle$ there are feasible sequences of jobs, and therefore, there are optimal ones. Let $\sigma = (\sigma_1, \ldots, \sigma_n)$ be the optimal sequence with the minimum number of pairs of jobs standing in anti-Johnson's order. Permutation σ meets property $(a*)$ by our choice. Let us show how σ can be represented in the form (1) so as to meet the remaining properties $(*)$. In the following procedure, we split permutation σ into pieces $\sigma_{(i)}$, $i = 1, \ldots, m,$, and next prove that those pieces are J-blocks.

We consecutively look through jobs $\sigma_1, \ldots, \sigma_n$ as they follow in sequence σ. To begin with, job σ_1 is placed at the beginning of piece $\sigma_{(i(\sigma_1))}$, while all pieces $\sigma_{(i')}$ for $i' < i(\sigma_1)$ will remain empty till the end of this representation.

Suppose that all jobs $\sigma_{j'}$ for $j' < j$ have been examined, that the formation of pieces $\sigma_{(i')}$ for $i' < i$ has been completed, and a non-empty piece $\sigma_{(i)}$ is under formation. The next in turn job σ_j either extends piece $\sigma_{(i)}$ (if $r_{\sigma_j} \le r_{(i)}$), or initiates a new non-empty piece $\sigma_{(i'')}$ (with $i'' = i(\sigma_j)$) in case $r_{\sigma_j} > r_{(i)}$. In the latter case, we assume that the formation of all pieces $\sigma_{(i')}$ for $i' < i''$ is completed, although some of them may be empty.

Clearly, the algorithm described above provides a representation of permutation σ as a concatenation of m pieces. It is also clear, by construction, that each piece $\sigma_{(i)}$ meets property $(c*)$. To complete the proof of Lemma 5, it remains to show that each piece is a J-block.

Suppose that for some $j > 1$ the inequality $\sigma_{j-1} > \sigma_j$ holds, and σ_{j-1} belongs to piece $\sigma_{(i)}$. We will show that job σ_j must belong to a different piece $\sigma_{(i')}$, $i' > i$. Indeed, once the pair of jobs σ_{j-1}, σ_j stands in anti-Johnson's order, then the inequality $r_{\sigma_j} > s_{\sigma_{j-1}}(S_\sigma)$ must hold. Otherwise the transposition of these two jobs could yield a permutation σ' which, by Remark 2, is also optimal. Besides, it has a less, by one, number of anti-Johnson's pairs of jobs, which contradicts the choice of permutation σ. Thus, we obtain the relations

$$r_{(i)} \le s_{\sigma_k}(S_\sigma) \le s_{\sigma_{j-1}}(S_\sigma) < r_{\sigma_j}$$

(where σ_k is the first job of piece $\sigma_{(i)}$), and so, by description of the algorithm, job σ_j should fall into a different piece $\sigma_{(i')}$, $i' > i$. □

Lemma 6. *At each step $k = 0, \ldots, n$ algorithm $\mathcal{A}_1$ generates all permutations of jobs $\{1, \ldots, k\}$ that meet properties $(*)$.*

Proof. The statement is definitely valid for $k = 0, 1$. Suppose, it is true for step $k-1$, $k \geq 2$. Let us prove that pool Σ_k contains all permutations of jobs $\{1, \ldots, k\}$ that meet properties $(*)$. Let σ be such a permutation. We will prove that $\sigma \in \Sigma_k$.

It is clear that job k in representation (1) of permutation σ is the ending job of a J-block $\sigma_{(i)}$. Hence, removing job k from σ yields a permutation $\sigma' = \sigma'_{(1)} \oplus \cdots \oplus \sigma'_{(m)}$, where $\sigma_{(i)} = \sigma'_{(i)} \oplus (k)$ and $\sigma_{(i')} = \sigma'_{(i')}$ for all $i' \neq i$. Let us show that σ' meets properties $(*)$.

Indeed, by Lemma 3, permutation σ' is feasible, while properties $(b*)$ and $(c*)$ are inherited from permutation σ. Thus, σ' meets properties $(*)$, and by the induction hypothesis, it is contained in Σ_{k-1}.

Once σ meets $(*)$, we have $r_k \leq r_{(i)}$. Therefore, at the kth step of algorithm $\mathcal{A}_1$ in one of the variants of augmenting the permutation σ' by job k, the latter is placed exactly to the end of J-block $\sigma'_{(i)}$. And since the resulting permutation (coinciding with σ) is feasible, it must be added to pool Σ_k. □

Now suppose that for a given instance I of problem $\langle 1 \mid rp, r_i \mid C_{\max}\rangle$ there exist feasible permutations of jobs. By Lemma 6, the optimal permutation σ_{opt} (whose existence is proved in Lemma 5) must be generated at the nth step of algorithm $\mathcal{A}_1$. And since pool Σ_n contains no infeasible permutations (by Lemma 4), no permutation $\sigma \in \Sigma_n$ may provide a less value of the objective function that σ_{opt} does. It follows that at the $(n+1)$th step algorithm $\mathcal{A}_1$ will choose one of the optimal permutations of jobs. Thus, we have proved the following.

Theorem 7. *Given an instance of problem $\langle 1 \mid rp, r_i \mid C_{\max}\rangle$, algorithm $\mathcal{A}_1$ either finds an optimal sequence of jobs, or determines that no feasible sequences exist.* □

Therefore, all sequences of jobs constructed at all steps of algorithm $\mathcal{A}_1$ meet properties $(*)$. These properties enable us to create a more efficient procedure of checking the feasibility of any sequence $\sigma \in \Sigma_k$ and for computing the length of its active schedule S_σ (applicable for permutations $\sigma \in \Sigma_n$).

As follows from property $(c*)$, for any sequence $\sigma = \sigma_{(1)} \oplus \cdots \oplus \sigma_{(m)} \in \Sigma_k$ in representation (1), all jobs in the J-block $\sigma_{(i)}$ are processed in the active schedule S_σ as an "active block", *i.e.*, without an inner idle time. In addition, the J-block $\sigma_{(i)}$ starts processing in schedule S_σ either in its proper release date $r_{(i)}$, or later, immediately after the completion of the previous J-block. Thus, for computing the length of the active schedule S_σ, it suffices to know just the length $P_{(i)}(\sigma)$ of each J-block $\sigma_{(i)}$ (*i.e.*, the total length of all its jobs). Consequently, the active schedules for any two permutations of jobs σ', σ'' having identical vectors $(P_{(1)}(\sigma'), \ldots, P_{(m)}(\sigma')) = (P_{(1)}(\sigma''), \ldots, P_{(m)}(\sigma''))$ have the same length.

Furthermore, for checking the feasibility of any permutation $\sigma \in \Sigma_k$ obtained from a feasible permutation $\sigma' \in \Sigma_{k-1}$ by adding job k to the end of a J-block $\sigma'_{(i)}$, it is sufficient to know:

- the amount of resource (denoted as $\bar{Q}_{(i')}(\sigma')$) in the resource pool at the completion of each J-block $\sigma_{(i')}$;
- the minimum level of resource (denoted as $Q_{(i')}(\sigma')$) attained in the pool while processing the jobs of the (i')th J-block.

(For a feasible permutation of jobs, all these parameters should be nonnegative.) Thus, for checking the feasibility of newly obtained sequences of jobs $\sigma \in \Sigma_k$ in the course of algorithm $\mathcal{A}_1$ and for computing the lengths of their active schedules, it is sufficient to keep only their *schemes*, *i.e.*, $(3m)$-dimensional vectors of parameters

$$Y(\sigma) = ((Q_{(1)}(\sigma), \bar{Q}_{(1)}(\sigma), P_{(1)}(\sigma)), \ldots, (Q_{(m)}(\sigma), \bar{Q}_{(m)}(\sigma), P_{(m)}(\sigma))).$$

This remark enables us to describe a more efficient DP-algorithm for determining an optimal permutation of jobs in the $\langle 1 \mid rp, r_i \mid C_{\max}\rangle$ problem. Given an instance of our problem, the algorithm accumulates step-by-step distinct schemes of feasible job permutations, rather than the permutations themselves. But under this approach, we need to append to our algorithm an additional *restoration step* aimed at restoring the optimal permutation of jobs from its scheme.

It is well known from the theory of Dynamic Programming that such a restoration of an optimal solution (at the reverse trace of the DP-algorithm) can be performed in different ways, realizing a tradeoff between the running time and the memory requirement of the DP-algorithm. In this paper, we chose the version of DP with the minimum memory requirement (which even in this case is huge enough), while losing somewhat concerning the running time.

So, instead of collecting job sequences (permutations of the first k jobs indexed in Johnson's order) in pools Σ_k, we will collect different schemes of those sequences in pools $\mathcal{Y}_k$ at steps $k = 0, 1, \ldots, n$. Furthermore, at each step $k = 1, \ldots, n$ we will keep in memory only two pools: $\mathcal{Y}_{k-1}$ and $\mathcal{Y}_k$. At step $n+1$ for each scheme $Y \in \mathcal{Y}_n$ by means of its parameters $\{P_{(1)}, \ldots, P_{(m)}\}$ we calculate the length of the corresponding active schedule and choose the scheme with the minimum makespan. And next, to obtain the optimal job permutation, we apply the restoration step consisting of n-times repeated Forward Trace of our DP-algorithm.

In a more detailed description of the algorithm presented in the next section, it will be shown how to compute the scheme of each job sequence $\sigma \in \Sigma_k$ obtained at step k from a job sequence $\sigma' \in \Sigma_{k-1}$, knowing only the scheme of the latter sequence.

4. Multi-parametric dynamic program

As is known, in the classical scheme of a DP-algorithm, applied, for example, for solving the Knapsack problem, at each step $k = 1, 2, \ldots$ of the algorithm we collect solutions with different values of a single dynamic parameter. (For the latter,

either the capacity of the knapsack, or the total cost of its contents is chosen.) As a result, the process of collecting solutions at steps $k = 1, 2, \ldots$ can be described on a plane sheet of paper as a two-dimensional table.

For the dynamic programming algorithm described below, this approach is not viable, because the number of dynamic parameters may be arbitrarily large (for different instances of our problem). Besides, the running time of such a DP-algorithm exponentially depends on the number of dynamic parameters. Such algorithms will be further referred to as *Multi-Parametric Dynamic Programs*. The preliminary analysis of properties of optimal solutions performed in the previous section enables us to reduce the number of dynamic parameters to $3m$, where m is the number of distinct job release dates in a given problem instance. Thus, the running time of the algorithm essentially depends on that, whether the parameter m is bounded by a constant, or alternatively, it is part of the input and may take arbitrarily large values in the framework of the problem under consideration.

Algorithm $\mathcal{A}_{DP}$

The algorithm consists of two parts: *Forward Trace* and *Reverse Trace*.

Forward Trace

This phase is aimed at finding the *scheme of an optimal schedule*.

Step 0. $Y_0 := ((Q_0, Q_0, -1), \ldots, (Q_0, Q_0, -1))$; $\mathcal{Y}_0 := \{Y_0\}$;
$(* P_{(i)}(\sigma) = -1$ provides the information that J-block $\sigma_{(i)}$ of permutation σ is empty,
while $P_{(i)}(\sigma) = 0$ does not necessarily mean that, keeping in mind that there may be
zero-length jobs $*)$
$\delta_k := \beta_k - \alpha_k$, for every $k = 1, \ldots, n$.

Steps $k = 1, \ldots, n$. Enumerate all schemes $Y' \in \mathcal{Y}_{k-1}$, and for each scheme $Y' \in \mathcal{Y}_{k-1}$ ($Y' = ((Q'_{(1)}, \bar{Q}'_{(1)}, P'_{(1)}), \ldots, (Q'_{(m)}, \bar{Q}'_{(m)}, P'_{(m)}))$) generate a new scheme $Y \in \mathcal{Y}_k$ by adding job k to the end of the i'th J-block of Y' (with i' sequentially taking the values from $i(k)$ to m), except for those J-blocks where $i' > i(k)$ and $P'_{(i')} = -1$:

$Y := Y'$;
$P_{(i')} := P'_{(i')} + p_k$; $Q_{(i')} := \min\{Q'_{(i')}, \bar{Q}'_{(i')} - \alpha_k\}$;
$Q_{(i'')} := Q'_{(i'')} + \delta_k$, $i'' = i' + 1, \ldots, m$;
$\bar{Q}_{(i'')} := \bar{Q}'_{(i'')} + \delta_k$, $i'' = i', \ldots, m$.
If all components of vector Y are nonnegative and $Y \notin \mathcal{Y}_k$, then add Y to pool $\mathcal{Y}_k$.

Step $n + 1$. If $\mathcal{Y}_n = \varnothing$ then {Output "No feasible schedules"; **STOP**}
else select from pool $\mathcal{Y}_n$ the scheme Y_{opt} with the minimum length of its active schedule.
end $(*$ of the Forward Trace $*)$

Reverse Trace

This phase is aimed at retrieving an optimal sequence σ_{opt} corresponding to the optimal scheme Y_{opt}.
$\widetilde{Y} := Y_{\text{opt}}$; $\sigma_{(i)} := \sigma_{null}$, $i = 1, \ldots, m$;
for $K := n$ **downto** 1 **do begin**
Repeat **Steps on $k = 0, \ldots, K$** of the Forward Trace until at step $k = K$ we find a scheme $Y \in \mathcal{Y}_K$ equal to $\widetilde{Y}$. Fix the scheme $Y' \in \mathcal{Y}_{K-1}$ from which the scheme $\widetilde{Y}$ was obtained by adding job K. Fix also the number i' of the J-block to which job K was appended, put $\sigma_{(i')} := (K) \oplus \sigma_{(i')}$; $\widetilde{Y} := Y'$.
end $(*$ Steps on $K *)$
$\sigma_{\text{opt}} := \sigma_{(1)} \oplus \cdots \oplus \sigma_{(m)}$; Output (σ_{opt}).
end $(*$ of the Reverse Trace $K *)$ □

The optimality of algorithm $\mathcal{A}_{DP}$ follows from the previous discussion. Now let us analyze the required memory space and the running time of the algorithm.

Since the total amount of job processing times can be divided into m integral parts (*i.e.*, J-blocks) in $O\left(\left(\sum_{j=1}^{n} p_j\right)^m / m!\right)$ ways, and each of the parameters $Q_{(i)}, \bar{Q}_{(i)}$ of a feasible scheme Y may take only integral values from the interval $[0, Q_0 + \sum_{j=1}^{n} \beta_j]$, the number of different schemes Y in each pool $\mathcal{Y}_k$ can be bounded by

$$M \doteq \left(\sum_{j=1}^{n} p_j\right)^m \left(Q_0 + \sum_{j=1}^{n} \beta_j\right)^{2m} \Big/ m! \tag{2}$$

Since at any step of the algorithm we keep in memory only two pools of schemes, and each scheme depends on $3m$ parameters, the required memory does not exceed $O(mM)$.

As for the running time of $\mathcal{A}_{DP}$, the dominant phase is the reverse trace, where we have to repeat the steps on k of the Forward Trace n times. In turn, at each step on $k = 1, \ldots, n$ we have to enumerate the schemes of the pool $\mathcal{Y}_{k-1}$, and for each scheme $Y \in \mathcal{Y}_{k-1}$ examine $O(m)$ possible variants of insertion of job k. For each newly obtained scheme, we have to check

if it already exists in the pool $\mathcal{Y}_k$, and if not, we should add the new scheme to the pool. This can be done (by organizing the pool as a binary search tree) in $O(m \log M)$ time. Summing up, we come up with the overall running time $O(n^2m^2M \log M)$, and conclude the results in the following theorem.

Theorem 8. *The $\langle 1 \mid rp, r_i \mid C_{\max}\rangle$ problem with n jobs and m distinct release dates can be solved by algorithm $\mathcal{A}_{DP}$ in $O(n^2m^2M \log M)$ time with $O(mM)$ memory requirement, where M is defined in (2). When m is fixed, the algorithm is pseudo-polynomial in the input size.*

As will be shown below, the result conveyed by the multi-parametric dynamic program is tight at two points. First, it cannot be extended to the case when m is part of the input, because the problem becomes strongly NP-hard (as proved later in Theorem 10). And second, it cannot be strengthened down to a polynomial-time algorithm, since, due to Theorem 11 proved below, the $\langle 1 \mid rp, r_i \mid C_{\max}\rangle$ problem remains NP-hard even if there are only two distinct release dates.

5. Complexity results for special cases

In this section we consider two cases where the contributions of all jobs have the same sign.

1. $\langle 1 \mid rp\ (\delta_i \geq 0), r_i \mid C_{\max}\rangle$ (all jobs have nonnegative contributions)
2. $\langle 1 \mid rp\ (\delta_i < 0), r_i \mid C_{\max}\rangle$ (all jobs have negative contributions).

It will be shown that the first case can be solved in polynomial time, while the second one is NP-hard in the strong sense. First, we describe a greedy procedure for problem $\langle 1 \mid rp\ (\delta_i \geq 0), r_i \mid C_{\max}\rangle$. In this procedure the jobs will be considered in non-decreasing order of their release dates, so, there is no need for indexing them in Johnson's order.

Greedy Procedure (GP)
Whenever the machine becomes vacant, process an arbitrary *feasible job*, *i.e.*, a job that is released and there is sufficient resource in the pool for its processing. If the set of such jobs is empty, but still there are jobs not yet released, wait for the next job to be released. Once all jobs have been released and none of the unscheduled jobs can be processed, output the obtained (maybe, partial) sequence $\hat{\sigma}$ and STOP. □

Theorem 9. *For any instance of the $\langle 1 \mid rp\ (\delta_i \geq 0), r_i \mid C_{\max}\rangle$ problem with n jobs, GP either finds an optimal sequence of jobs or establishes that no feasible sequence exists. The algorithm can be implemented in $O(n \log n)$ time.*

Proof. First, assume that there are no feasible sequences of all n jobs (and so, there are no optimal ones). Since GP can only produce feasible sequences, in this situation it cannot produce a complete feasible sequence of n jobs, therefore, it will output an incomplete sequence $\hat{\sigma}$, enabling us to conclude that there are no optimal sequences of n jobs.

Assume now that the set of optimal sequences of n jobs is non-empty, and let $\tilde{\sigma}$ be the optimal sequence having the longest prefix (σ') in common with sequence $\hat{\sigma}$. Let k be the length of that prefix. If $k = n$, sequence $\hat{\sigma}$ is optimal and we are done.

Suppose now that $k < n$. We will show that this leads to a contradiction. Notice that prefix σ' (common for sequences $\tilde{\sigma}$ and $\hat{\sigma}$) is identically scheduled in the corresponding active schedules ($S_{\tilde{\sigma}}$ and $S_{\hat{\sigma}}$), and after time t when it is completed in both schedules we have a job $\tilde{\sigma}_{k+1}$ that can be feasibly commenced at some point in time $t'' \geq t$ (the time when it is started in schedule $S_{\tilde{\sigma}}$). This implies that GP also cannot stop at this position, once after time t some new jobs appear that can be feasibly commenced. Let so $\hat{\sigma}_{k+1}$ be the job chosen by GP for the $(k+1)$th position. By the logic of the greedy algorithm (that compels one to start every next job at the earliest possible time, when there are feasible jobs), we may conclude that job $\hat{\sigma}_{k+1}$ starts in schedule $S_{\hat{\sigma}}$ at time $t' \in [t, t'']$.

By the definition of k, we know that job $\hat{\sigma}_{k+1}$ stands in sequence $\tilde{\sigma}$ in a later position $k' > k+1$. Let us transform sequence $\tilde{\sigma}$ to sequence $\tilde{\sigma}'$ by moving job $\hat{\sigma}_{k+1}$ to the $(k+1)$th position. It can be easily shown that sequence $\tilde{\sigma}'$ is also optimal. Indeed, the sequence is feasible, since, first, job $\hat{\sigma}_{k+1}$ is feasible at this position in sequence $\hat{\sigma}$, and second, the insertion of a job with nonnegative contribution into a feasible sequence cannot violate the feasibility of the subsequent jobs. Furthermore, the above transformation of sequence $\tilde{\sigma}$ cannot increase the length of the corresponding active schedule, since the fragment of sequence $\tilde{\sigma}'$ from $(k+1)$th to k'th position starts in the active schedule even earlier (at time t' instead of t'') than in the optimal schedule $S_{\tilde{\sigma}}$. Thus, $\tilde{\sigma}'$ is optimal.

But as can be seen, sequence $\tilde{\sigma}'$ has at least $(k+1)$-length prefix in common with sequence $\hat{\sigma}$, which contradicts the choice of the original optimal sequence $\tilde{\sigma}$. The contradiction proves the optimality of sequence $\hat{\sigma}$.

Now we derive the bound on running time. Although GP is able to choose any feasible job at every step, to simplify the choice of the feasible job from among the set of released jobs, we keep the information on this set as a heap prioritized by Johnson's rule. We can thereby maintain the set within $O(n \log n)$ running time, and this, clearly, can be taken for the upper bound on the running time of the whole Greedy Procedure. □

After solving the case with positive contributions, we now move to the case in which $\delta_i < 0$ for all jobs i. In the corresponding two-machine flowshop scheduling problem of makespan minimization, we may reverse the processing order of the two operations of each job and consider the Gantt chart from right to left (thereby converting the jobs with negative contributions to the ones with positive contributions). The schedule remains feasible within the same makespan. In other

words, the symmetry, or the *mirror property*, can be established. However, it is not the case in the relocation problem with release dates. In the following, we will show the computational intractability of the case with negative contributions. The proof is based upon a reduction from 3-PARTITION, which is known to be strongly NP-complete [4].

3-PARTITION:
INSTANCE: A positive integer B and a set of items $A = \{1, 2, \ldots, 3u\}$ with positive integer weights $x_i,\ i \in A$, such that $B/4 < x_i < B/2,\ i \in A$, and $\sum_{i=1}^{3u} x_i = uB$.
QUESTION: Is there a partition of set A into u subsets $A_1, \ldots, A_u$ such that $\sum_{i \in A_j} x_i = B$ for each subset A_j?

Theorem 10. *Problem* $\langle 1 \mid rp\ (\delta_i < 0), r_i \mid C_{\max} \rangle$ *is strongly NP-hard.*

Proof. Given an instance of 3-PARTITION, we construct a scheduling instance I of $4u$ jobs with the initial resource level $Q_0 = uB + u - 1$ as follows.

Ordinary jobs $i = u+1, \ldots, 4u$: $p_i = \alpha_i = x_{i-u}, \beta_i = r_i = 0$.
Enforcer jobs $i = 1, \ldots, u-1$: $\alpha_i = (u-i)(B+1),\ \beta_i = \alpha_i - 1,\ p_i = 0,\ r_i = iB$.
Enforcer job u (introduced for consistency): $\alpha_u = \beta_u = p_u = 0,\ r_u = uB$.

Assume w.l.o.g. $u \geq 2$. We claim that the desired partition exists for 3-PARTITION if and only if a feasible schedule S for instance I exists and has a makespan $C_{\max}(S) \leq uB$.

Suppose subsets $A_1, \ldots, A_u$ constitute a desired partition. We will show that for instance I there exists a feasible schedule S of length uB. Let $\mathcal{J}_j$ denote the set of ordinary jobs defined by the elements from $A_j,\ 1 \leq j \leq u$, and let $\sigma(\mathcal{J}_j)$ stand for an arbitrary sequence of jobs in $\mathcal{J}_j$. It is not difficult to check that the sequence

$$\sigma' = \sigma(\mathcal{J}_1) \oplus (1) \oplus \sigma(\mathcal{J}_2) \oplus (2) \oplus \cdots \oplus (u-1) \oplus \sigma(\mathcal{J}_u) \oplus (u)$$

is feasible and the makespan of its active schedule $S_{\sigma'}$ is equal to uB.

We proceed to the *if* part of the claim. Assume that there is a feasible sequence σ' for jobs in I with $C_{\max}(S_{\sigma'}) = uB$. Since the total processing time of jobs in I is equal to uB, no idle time is allowed in schedule $S_{\sigma'}$. From the values of β_i and r_i we can conclude that the enforcer jobs are sequenced in schedule σ' in the order $1, \ldots, u$ (which means, in the order as they are released). Let $\mathcal{J}_1$ denote the subset of the ordinary jobs preceding enforcer job 1. If $\sum_{i \in \mathcal{J}_1} \alpha_i > B$, then the resource level before the processing of job 1 is

$$Q_0 - \sum_{i \in \mathcal{J}_1} \alpha_i < (u-1)(B+1).$$

The resource requirement of job 1 is $(u-1)(B+1)$. Infeasibility thus arises. On the other hand, if $\sum_{i \in \mathcal{J}_1} \alpha_i < B$, the total processing time of the ordinary jobs in $\mathcal{J}_1$ is smaller than B, implying idle time before $r_1 = B$. Therefore, $\sum_{i \in \mathcal{J}_1} \alpha_i = B$ must hold. The items corresponding to the ordinary jobs in $\mathcal{J}_1$ constitute a subset A_1 with $\sum_{i \in A_1} x_i = B$. Successively applying this argument to enforcer jobs 2 to u, we come up with subsets of ordinary jobs $\mathcal{J}_2, \ldots, \mathcal{J}_u$, where $\mathcal{J}_j$ is the subset of the ordinary jobs scheduled in σ' between enforcer job $j-1$ and job j, such that $\sum_{i \in \mathcal{J}_j} \alpha_i = B$. This provides the desired partition $A_1, \ldots, A_u$ for 3-PARTITION and completes the proof of Theorem 10. □

As shown in the next theorem, further restriction of the problem $\langle 1 \mid rp\ (\delta_i < 0), r_i \mid C_{\max} \rangle$ to the case with only two distinct release dates still retains the problem NP-hard.

Theorem 11. *Problem* $\langle 1 \mid rp\ (\delta_i < 0), r_i \in \{0, r\} \mid C_{\max} \rangle$ *is ordinary NP-hard.*

Proof. The proof can be given using a reduction from the PARTITION problem.

PARTITION:
INSTANCE: A positive integer B and a set of items $A = \{1, 2, \ldots, u\}$ with positive integer weights $x_i,\ i \in A$, such that $\sum_{i=1}^{u} x_i = 2B$.
QUESTION: Is there a partition of set A into subsets A_1 and A_2 such that $\sum_{i \in A_1} x_i = \sum_{i \in A_2} x_i = B$?

Given an instance of the PARTITION problem, an instance of problem $\langle 1 \mid rp\ (\delta_i < 0), r_i \in \{0, r\} \mid C_{\max} \rangle$ is defined as follows.

Initial resource level $Q_0 = 2B + 1$.
Ordinary jobs $i = 1, \ldots, u$: $p_i = \alpha_i = x_i,\ \beta_i = r_i = 0$.
Enforcer job $i = 0$: $\alpha_0 = B + 1,\ \beta_0 = B,\ p_0 = 0,\ r_0 = B$.

Using the same argument as in Theorem 10, we can show that there is a desired partition in the PARTITION problem if and only if there exists a feasible schedule of length $2B$. □

Remark 12. Note that the result of Theorem 11 cannot be strengthened to a strong NP-hardness result, since the problem with two distinct release dates can be solved in pseudo-polynomial time by the dynamic programming algorithm $\mathcal{A}_{DP}$.

Table 1
Complexity of special cases of problem $\langle 1 \mid rp, r_i \mid C_{\max} \rangle$.

Conditions	Complexity	Source
$\delta_i \geq 0$	$O(n \log n)$	Theorem 9
$\delta_i < 0$	Strongly NP-hard	Theorem 10
$r_i \in \{r_{(1)}, \ldots, r_{(m)}\}$ for fixed m	Pseudo-polynomial DP-algorithm	Theorem 8
$m = 2,\ \delta_i < 0$	Ordinary NP-hard	Theorem 11

6. Conclusion

In this paper, we presented first results for the single-machine relocation problem with release-date constraints. Through a series of lemmas and statements, we established some general properties of feasible and optimal solutions of the problem. On the basis of those properties, we designed several algorithms shown to be efficient in certain special cases. As a counterpart to those positive results, some NP-hardness results were established. The results obtained are summarized in Table 1.

As one can learn from the table, the polynomially solvable case with positive contributions (Theorem 9) and the counterpart case with negative contributions (which is strongly NP-hard, by Theorem 10) do not pose the symmetry property that might be expected for a flow shop problem with the minimum makespan objective (which is explainable due to the presence of release dates). Theorem 10 establishes the strong NP-hardness of the case with a variable number m of distinct release dates. On the other hand, when m is fixed, the problems can be solved in pseudo-polynomial time (Theorem 8). As shown by Theorem 11, the pseudo-polynomial time algorithm is the best of what one could do while solving this problem, because it remains NP-hard even if there are only two distinct release dates.

The contributions of this paper are two folds. First, we have studied, with tight complexity analysis, a new scheduling problem that extends the relocation problem to the case of arbitrary release dates. We thereby extended the horizon of study of this important resource-constrained scheduling problem. Second, we have demonstrated a powerful technique for determining tight boundaries between ordinary and strongly NP-hard cases of hard optimization problems by means of the tool of the so-called multi-parametric dynamic programs.

Now we would like to discuss further possible development of these results. Interesting directions for further research on the relocation problem with release dates include the cases with unit processing times or a limited number of different processing times, as well as the case with a limited number of different contributions. Application of the multi-parametric dynamic programming approach for determining boundaries between ordinary and strongly NP-hard cases of other hard optimization problems could also be a worthy direction of further research.

Acknowledgements

The research is partially supported by the National Science Council of Taiwan (grant nos 98-2410-H-009-011 and 98-2811-H-009-003), the Russian Foundation for Basic Research (grant nos 08-01-00370 and 08-06-92000-HHC) and the Federal Target Grant "Scientific and educational personnel of innovation Russia" for 2009–2013 (government contract No. 14.740.11.0362). The authors are grateful to the anonymous referees for their constructive comments.

References

[1] J. Blazewicz, J.K. Lenstra, A.H.G. Rinnooy Kan, Scheduling subject to resource constraints: classification and complexity, Discrete Applied Mathematics 5 (1983) 11–24.
[2] P. Brucker, A. Drexl, R. Möhring, K. Neumann, E. Pesch, Resource-constrained project scheduling: notation, classification, models, and methods, European Journal of Operational Research 112 (1) (1999) 3–41.
[3] T.C.E. Cheng, B.M.T. Lin, Johnson's rule, composite jobs and the relocation problem, European Journal of Operational Research 192 (3) (2009) 1008–1013.
[4] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freedman, San Francisco, CA, 1979.
[5] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnoy Kan, Optimization and approximation in determenistic sequencing and scheduling: a survey, Annals of Discrete Mathematics 5 (1979) 287–326.
[6] P.L. Hammer, Scheduling under resource constraints — deterministic models, Annals of Operations Research 7 (1986).
[7] W. Herroelen, B. De Reyck, E. Demeulemeester, Resource-constrained project scheduling: a survey of recent developments, Computers and Operations Research 25 (4) (1998) 279–302.
[8] S.M. Johnson, Optimal two- and three-stage production schedules with setup times included, Naval Research Logistics Quarterly 1 (1954) 61–67.
[9] E.H. Kaplan, Relocation models for public housing redevelopment programs, Planning and Design 13 (1) (1986) 5–19.
[10] E.H. Kaplan, A. Amir, A fast feasibility test for relocation problems, European Journal of Operational Research 35 (1988) 201–205.
[11] E.H. Kaplan, O. Berman, Orient Heights housing projects, Interfaces 18 (6) (1988) 14–22.
[12] A.V. Kononov, B.M.T. Lin, On the relocation problems with multiple identical working crews, Discrete Optimization 21 (4) (2006) 368–381.
[13] A.V. Kononov, B.M.T. Lin, Minimizing the total weighted completion time in the relocation problem, Journal of Scheduling 13 (2) (2010) 123–129.
[14] B.M.T. Lin, S.T. Liu, Maximizing the reward in the relocation problem with generalized due dates, International Journal of Production Economics 115 (1) (2008) 55–63.
[15] A. Mingozzi, V. Maniezzo, S. Ricciardelli, L. Bianco, An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation, Management Science 44 (5) (1998) 714–729.
[16] PHRG. New lives for old buildings: Revitalizing public housing project, Technical Report, Public Housing Group, Department of Urban Studies and Planning, MIT, Cambridge, MASS, 1986.
[17] M. Pinedo, Planning and Scheduling in Manufacturing and Services, Springer Verlag, New York, 2005.
[18] J.-X. Xie, Polynomial algorithms for single machine scheduling problems with financial constraints, Operations Research Letters 21 (1) (1997) 39–42.