Batch scheduling and common due-date assignment on a single machine

T.C. Edwin Cheng\textsuperscript{a,}\textsuperscript{*}, 1, Mikhail Y. Kovalyov\textsuperscript{b,2}

\textsuperscript{a}Office of the Vice President, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
\textsuperscript{b}Institute of Engineering Cybernetics, Belarus Academy of Sciences, Surganova 6, 220012 Minsk, Belarus

Received 21 November 1994; revised 13 September 1995

Abstract

We consider the problem of scheduling \( n \) groups of jobs on a single machine where three types of decisions are combined: scheduling, batching and due-date assignment. Each group includes identical jobs and may be split into batches; jobs within each batch are processed jointly. A sequence independent machine set-up time is needed between each two consecutively scheduled batches of different groups. A due-date common to all jobs has to be assigned. A schedule specifies the size of each batch, i.e. the number of jobs it contains, and a processing order for the batches. The objective is to determine a value for the common due-date and a schedule so as to minimize the sum of the due date assignment penalty and the weighted number of tardy jobs. Several special cases of this problem are shown to be ordinary NP-hard. Some cases are solved in \( O(n \log n) \) time. Two pseudopolynomial dynamic programming algorithms are presented for the general problem, as well as a fully polynomial approximation scheme.

Keywords: Scheduling; Batching; Due-date assignment; Dynamic programming; Fully polynomial approximation scheme

1. Introduction

We consider the following due-date assignment and batch scheduling problem. There are \( n \) groups of jobs to be scheduled for processing on a single machine. All jobs are available at time zero. Each group \( j \) consists of \( q_j \geq 1 \) identical jobs with a same processing requirement \( p_j \geq 0 \) and a weight \( w_j \geq 0 \). Each job is completed immediately when its processing is finished. Each group may be partitioned into batches containing contiguous scheduled jobs. A set-up time \( s_j \geq 0 \) is required before a batch of group \( j \) is processed if it is processed first on the machine or immediately after a batch of
another group. Thus, set-up times are sequence independent. The machine can handle only one job at a time and cannot process any job whilst a set-up is performed. A schedule specifies the size of each batch, i.e. the number of jobs it contains, and a processing order for the batches. A due-date value $d \geq 0$ common to all jobs has to be determined. For any schedule and any due-date value $d$, a job $i$ with completion time $C_i$ is *early* if $C_i \leq d$ and it is *tardy* if $C_i > d$. We denote the number of tardy jobs of group $j$ by $U_j$.

The objective is to determine a value for the common due-date $d$ and a schedule so as to minimize the sum of the due-date assignment penalty and the weighted number of tardy jobs: $r(d) + \sum_{j=1}^{n} w_j U_j$. The due-date assignment penalty function $r(d)$ is defined as follows: given a threshold value $R \geq 0$ and a coefficient $\alpha > 0$, $r(d) = 0$ if $d \leq R$ and $r(d) = \alpha d$ if $d > R$. All data are assumed to be integers.

The problem formulated above combines three types of decisions: scheduling, batching and due-date assignment. The significance of assigning accurate due-dates to jobs is well recognized by researchers. A vast body of the literature is devoted to the common due-date assignment problem (see, for example, a survey of Cheng and Gupta [3]). There has also been recent interest in batch scheduling problems (see papers of Bruno and Downey [2], Dobson et al. [6], Naddef and Santos [13], Coffman et al. [4], Monma and Potts [12], Coffman et al. [5], Albers and Brucker [1]). In many practical situations of production scheduling, batching, scheduling and due-date assignment decisions are strongly inter-related. Using the advent of computer aided manufacturing, these decisions can be integrated and computer-controlled. Our paper initiates the study of problems where scheduling, batching and due-date assignment decisions are combined.

An example of a practical situation involving these three types of decisions can be given as follows. Consider a production line where orders consisting of identical items are processed. If no shipment of items is possible until the entire order is completed, then the customer may be out of stock while awaiting delivery. However, customer service is improved if a few items are produced in the near future to cover the customer's immediate demand and the remaining part of the order is produced at some later time. As for due-date assignment, the common due-date assignment method has been extensively studied in the scheduling literature [3]. This method of due-date assignment is commonly in use when a uniform due-date is quoted on all orders so as to project an image of fair treatment to all customers [14]. Clearly, this situation can be modelled by the problem where scheduling, batching and due-date assignment decisions are combined.

It should be noted that, instead of batch scheduling, preemptive scheduling can be considered. However, in the literature on preemptive scheduling, there is no penalty when a job is preempted that is rather unusual in practice. It is more likely that some machine set-up time is incurred when a job is preempted.

We now comment on the structure of an optimal schedule for any fixed due-date value $d$. Without loss of generality, we assume that all jobs of a group which are scheduled to be tardy can be placed in a single batch called a *tardy batch*. Furthermore,
if there are two or more batches of the same group containing early jobs, then, without
increasing the weighted number of tardy jobs, they may be combined to form a single early batch. Thus, we may restrict our search to schedules which contain at most one early and at most one tardy batch for each group. Clearly, tardy batches can be scheduled in an arbitrary order after the last early batch. Moreover, since there is a common due date, early batches can also be scheduled in an arbitrary order.

The rest of the paper is organized as follows. In the next section, we study the computational complexity of various special cases of the problem. We show that the problem is NP-hard for the following cases:

• equal weights, unit job processing times;
• equal weights, equal set-up times;
• a single job for each group, zero set-up times;
• a single job for each group, zero job processing times.

We present $O(n \log n)$ algorithms for the following cases:

• equal weights, equal numbers of jobs for each group;
• equal weights, equal set-up times, equal job processing times;
• equal set-up times, equal job processing times, equal number of jobs for each group.

The complexity of the problem with equal set-up times and equal job processing times remains an open question.

In Section 3, two dynamic programming algorithms with $O(\sum_{j=1}^{n} q_j \sum_{i=1}^{J} w_i q_i)$ and $O(\sum_{j=1}^{n} q_j \sum_{i=1}^{J} (s_i + p_i q_i))$ running times are presented for the general problem. In Section 4, a fully polynomial approximation scheme is derived. The paper concludes with some remarks and suggestions for further research.

It should be noted that the problem with equal weights and externally given due-dates has been studied by Kovalyov [11]. Some ideas of this paper are used in our NP-hardness proofs. The ideas of our polynomial time algorithms are new, as well as the dynamic programming formulations and the approach to developing a fully polynomial approximation scheme.

2. Special cases

In this section, we study the computational complexity of various special cases of the original problem.

We adopt the three field notation of Graham et al. [9] to denote our type of problems. In this notation $1/\beta/\gamma$, the first field denotes the single machine processing system. The second field, $\beta \subset \{s_j = s, p_j = p, q_j = q\}$, specifies some group characteristics (equal set-up times, equal job processing times and equal number of jobs for all groups, respectively). The third field, $\gamma \in \{\sum w_j U_j + r(d), \sum U_j + r(d)\}$, refers to the optimality criterion. Here $\sum w_j U_j$ is the weighted number of tardy jobs and $\sum U_j$ arises when all weights are equal. Our original problem is represented by $1/1/\sum w_j U_j + r(d)$.

We first study the problem with equal weights. We show that the problems $1/p_j = 1/\sum U_j + r(d)$ and $1/s_j = s/\sum U_j + r(d)$ are both NP-hard and present $O(n \log n)$
algorithms for the problems \( 1/q_j = q_j/\sum U_j + r(d) \) and \( 1/s_j = s, p_j = p_j/\sum U_j + r(d) \). Our first NP-hardness proof is rather straightforward. It is as follows.

**Theorem 1.** The problem \( 1/p_j = 1/\sum U_j + r(d) \) is NP-hard.

**Proof.** We show that the decision version of the above problem is NP-complete by a transformation from the known NP-complete problem PARTITION [8]: given positive integers \( a_1, \ldots, a_n \), is there a set \( S \subseteq \{1, \ldots, n\} \) such that \( \sum_{j \in S} a_j = A/2 \), where \( A = \sum_{j=1}^{n} a_j \)? Given any instance of PARTITION, we construct the following instance of our problem. There are \( n \) groups with \( s_j = a_j, q_j = a_j \) and \( p_j = 1 \) for \( j = 1, \ldots, n \). We define \( R = A \) and \( a = 1 \), i.e. \( r(d) = 0 \) if \( 0 \leq d \leq A \) and \( r(d) = d \) if \( d > A \). We show that there exists a set \( S \) for which \( \sum_{j \in S} a_j = A/2 \) if and only if there is a solution to our problem with an objective value not exceeding \( A/2 \).

If there is a set \( S \) for which \( \sum_{j \in S} a_j = A/2 \), then we define \( d = A \) and schedule all jobs of the groups of \( S \) to be early. This yields \( r(d) = 0 \) and exactly \( A/2 \) tardy jobs.

Conversely, if there is a solution to our problem with an objective value not exceeding \( A/2 \), then \( d = A \) and there are at least \( A/2 \) early jobs. Since \( s_j = q_j p_j \) for \( j = 1, \ldots, n \), the time spent on set-ups in any interval is at least as large as the time devoted to processing. Thus, in our schedule with at least \( A/2 \) early jobs, the total set-up time within the interval \([0, d]\) is at least \( A/2 \). Since \( d \leq A \), we conclude that early batches in our schedule correspond to a set \( S \) for which \( \sum_{j \in S} a_j = A/2 \). □

Our next NP-hardness proof is based on the result of Gaizer [7] obtained for the case of an externally given due-date.

**Theorem 2.** The problem \( 1/s_j = s_j/\sum U_j + r(d) \) is NP-hard.

**Proof.** We show that the decision version of the above problem is NP-complete by a transformation from the NP-complete problem EQUAL CARDINALITY PARTITION [8]: given positive integers \( a_1, \ldots, a_n \) where \( n \) is even, is there a set \( S \subseteq \{1, \ldots, n\} \) such that \( |S| = n/2 \) and \( \sum_{j \in S} a_j = A/2 \) where \( A = \sum_{j=1}^{n} a_j \)? Given any instance of EQUAL CARDINALITY PARTITION, we construct the following instance of our problem. Set \( B = (A + a_1) \ldots (A + a_n) \). There are \( n \) groups with \( s_j = s = 2nAB, q_j = A + a_j \) and \( p_j = B(A+a_j-1)/(A+a_j) \) for \( j = 1, \ldots, n \). We define \( R = n^2 AB + R(nA + A - n)/2 \) and \( \alpha = 1 \), i.e. \( r(d) = 0 \) if \( 0 \leq d \leq R \) and \( r(d) = d \) otherwise. We show that there exists a set \( S \) for which \( |S| = n/2 \) and \( \sum_{j \in S} a_j = A/2 \) if and only if there is a solution to our problem with an objective value not exceeding \( (n + 1)A/2 \).

If there is a set \( S \) for which \( |S| = n/2 \) and \( \sum_{j \in S} a_j = A/2 \), then we define \( d = R \) and schedule all jobs of the groups of \( S \) to be early. This yields \( r(d) = 0 \) and exactly \( (n + 1)A/2 \) tardy jobs.

Conversely, suppose there is a solution to our problem with an objective value not exceeding \( (n + 1)A/2 \). Firstly, we note that \( d \leq R \) and \( r(d) = 0 \). Secondly, there are exactly \( n/2 \) early batches: there cannot be more than \( n/2 \) early batches since the
completion time of the last early batch will then be at least \((n/2+1)s > R\), and there cannot be less than \(n/2\) early batches since there will then be at least \((n/2 + 1)A\) tardy jobs. Let \(S\) be the set of groups having early batches and let \(e_j\) be the number of early jobs of group \(j, \ j = 1, \ldots, n\). Since \(r(d) = 0\), and there are no more than \((n + 1)A/2\) tardy jobs, the following inequality holds:

\[
\sum_{j \in S} e_j \geq (n + 1)A/2. \tag{1}
\]

For feasibility, we have

\[
\sum_{j \in S} (s + p_j e_j) = n^2AB + B \sum_{j \in S} e_j - B \sum_{j \in S} e_j/(A + a_j) \
\leq n^2AB + B(nA + A - n)/2. \tag{2}
\]

Substituting (1) into (2), we obtain

\[
\sum_{j \in S} e_j/(A + a_j) \geq n/2.
\]

Since \(e_j \leq q_j = A + a_j\) and \(|S| = n/2\), we deduce that \(e_j = A + a_j\) for \(j \in S\). Substituting \(A + a_j\) for \(e_j\) in (1), we deduce that \(\sum_{j \in S} a_j \geq A/2\); a similar substitution in (2) yields \(\sum_{j \in S} a_j \leq A/2\). Thus, \(\sum_{j \in S} a_j = A/2\), as required.

We now begin to study the problem \(1/q_j = q/\sum U_j + r(d)\). We first establish some properties of an optimal solution of this problem.

It is convenient to introduce some terminology.

The aggregate processing time of a group is the total processing time of all jobs of this group plus the corresponding set-up time. We define the SAPT (Shortest Aggregate Processing Time) sequence as a sequence \((j_1, \ldots, j_n)\) of non-split groups in nondecreasing order of their aggregate processing times: \(s_{j_i} + q p_{j_i} \leq s_{j_{i+1}} + q p_{j_{i+1}}\) for \(i = 1, \ldots, n - 1\).

A group is early if all its jobs are early; alternatively, it is tardy.

A group straddles the due-date if it has an early batch and a tardy batch which are processed contiguously. Clearly, there is no set-up time between these two batches; therefore, we consider the corresponding group as a non-split one.

Our algorithm for the problem \(1/q_i = q/\sum U_j + r(d)\) is based on the following lemma.

**Lemma 1.** For \(1/q_i = q/\sum U_j + r(d)\) there exists an optimal solution in which no group is split and all groups excluding the straddling group, if any, are sequenced in the SAPT order. The corresponding optimal due-date value is either equal to \(R\) or coincides with the completion time of one of the groups. In the former case, an optimal sequence of the groups has as many early groups (not jobs) as the SAPT sequence has. In the latter case, the SAPT sequence is optimal.
Proof. We first prove the existence of an optimal sequence of non-split groups in which all groups excluding the straddling group, if any, are sequenced in the SAPT order.

Let $e_j$ and $t_j$ denote the number of early and tardy jobs of group $j$, respectively, for $j = 1, \ldots, n$. Given an optimal solution, we consider the sequence of early batches and assume that there are at least two groups $i$ and $j$ such that $0 < e_i < q$ and $0 < e_j < q$. Assume without loss of generality $p_i \leq p_j$. Compute $\Delta = \min\{e_j, q - e_i\}$ and set $e_i = e_i + \Delta, t_i = t_i - \Delta, e_j = e_j - \Delta, t_j = t_j + \Delta$. For the new sequence, we have that the number of early jobs is not changed, the completion time of the last early batch is not increased and the number of non-split groups is increased by one. We may repeat this interchange process until we obtain an optimal sequence of early batches in which at most one early batch contains less than $q$ jobs. Clearly, this batch can be placed to be the last early batch and the corresponding group, say $k$, will be non-split as well as the others.

It is apparent that the SAPT sequence delivers the earliest completion time for any number of non-split groups. Hence, if we fix group $k$ to be the straddling group and schedule all other groups in the SAPT order, then there will be as many early jobs as in the original optimal sequence. Thus, the first statement of the lemma is proved.

We now consider an optimal solution in which no group is split. Without loss of generality assume that the optimal due-date value $d$ coincides with the completion time of a certain job of a certain group $k$ and $d > R$.

We show that there exists an optimal solution where $d$ is equal to $R$, or $d$ coincides with the completion time of group $k$ or with the completion time of the group preceding $k$ in the initial sequence of non-split groups.

If $\alpha p_k \leq 1$, then set $d = d + p_k$; otherwise, set $d = \max\{R, d - p_k\}$. It is apparent that this modification of the due-date value does not increase the optimal objective value. We repeat this modification until the value of $R$ or the start time of the first job of group $k$ is reached or the completion time of group $k$ is reached. In the former case, we can set $d$ to be equal to the completion time of the group preceding $k$ or $d = R$.

The latter statements of the lemma are easily proved, recalling that the SAPT sequence delivers the earliest completion time for any number of non-split groups. ☐

Lemma 1 provides the basis for the following algorithm for solving the problem $1/q = q/\sum U_j + r(d)$. In this algorithm 1, the groups are numbered in the SAPT order so that $s_1 + q p_1 \leq \cdots \leq s_n + q p_n$. Two possible choices for the optimal due-date value $d$ indicated in Lemma 1 are considered.

If $d = R$, then the first tardy group in the SAPT sequence $(1, \ldots, n)$ is found. If this group is $k$, then the SAPT sequence has $k - 1$ early groups. According to Lemma 1, a search for an optimal sequence is limited to group sequences $(1, \ldots, j - 1, j + 1, \ldots, k, j, k + 1, \ldots, n)$ for $j = 1, \ldots, k$ and $(1, \ldots, k - 1, j, k, \ldots, j - 1, j + 1, \ldots, n)$ for $j = k + 1, \ldots, n$, where group $j$ is assumed to be the straddling group. For each such a sequence, a value of $f_j$ is calculated which is the number of tardy jobs if the sequence has $k - 1$ early groups. No sequence can have more than $k - 1$ early groups. If it has
less than $k - 1$ early groups, then the value of $f_j$ is set to infinity. A sequence with
the minimal $f_j$ value is optimal if $d = R$.

If $d$ coincides with the completion time of one of the groups, then the SAPT sequence
is optimal. In this case, the optimal due-date value is found in a straightforward manner.
A formal description of the algorithm is given below.

**Algorithm 1**

**Step 1:** Number groups in the SAPT order so that $s_1 + q p_1 \leq \cdots \leq s_n + q p_n$. Set
$C_0 = 0$. For $k = 1, \ldots, n$, compute $C_k = \sum_{j=1}^{k} (s_j + q p_j)$. If $C_n \leq R$, then stop: the SAPT
sequence and $d = R$ is an optimal solution.

**Step 2:** If $C_{k-1} \leq R$ and $C_k > R$ for a certain $1 \leq k \leq n$, then compute

$$f_j = \begin{cases} q(n - k) + \left[ (C_k - \max\{R, C_k - q p_j\})/p_j \right] & \text{if } C_k - (s_j + q p_j) \leq R, \\ \infty & \text{otherwise}, \end{cases}$$

for $j = 1, \ldots, k$, and

$$f_j = q(n - k) + \left[ (C_{k-1} + s_j + q p_j - \max\{R, C_{k-1} + s_j\})/p_j \right]$$

for $j = k + 1, \ldots, n$.

Compute $f_j - \min\{f_j|j = 1, \ldots, n\}$.

**Step 3:** Compute $q(n - m) + r(C_m) = \min\{q(n - j) + r(C_j)|j = k, \ldots, n\}$.

**Step 4:** If $q(n - m) + r(C_m) \leq f_l$, then the SAPT sequence and $d = C_m$ is an optimal
solution with value $q(n - m) + r(C_m)$. Otherwise, sequence $(1, \ldots, l - 1, l + 1, \ldots, k, l, k + 1, \ldots, n)$ if $l \leq k$ or sequence $(1, \ldots, k - 1, l, k, \ldots, l - 1, l + 1, \ldots, n)$ if $l > k$ and $d = R$
is an optimal solution with value $f_l$.

**Theorem 3.** Algorithm 1 solves the problem $1/q_j = q/ \sum U_j + r(d)$ in $O(n \log n)$ time.

**Proof.** Step 1 of algorithm 1 constructs the SAPT sequence. Clearly, if the completion
time of the last group in this sequence does not exceed $R$, then the SAPT sequence
and $d = R$ is an optimal solution.

In Step 2, $d = R$ is assumed and values of $f_j$ are computed for all sequences where
$\text{group } j \text{ is fixed to be the straddling group and all other groups are in the SAPT order.}$

In Step 3, the optimal due-date value is determined assuming that the SAPT sequence
is optimal. Lemma 1 shows that there is an optimal solution among those constructed
in Steps 2 and 3. Hence, an optimal solution is found in Step 4.

Step 1 requires $O(n \log n)$ time. Steps 2–4 require $O(n)$ time. Thus, the overall time
complexity is $O(n \log n)$.

We now present an $O(n \log n)$ algorithm for the problem $1/s_j = s, p_j = p/ \sum U_j +
r(d)$. Our algorithm is based on the following lemma.

**Lemma 2.** For $1/s_j = s, p_j = p/ \sum U_j + r(d)$, there exists an optimal solution in which
no group is split and the groups are sequenced in nonincreasing order of $q_j$. The
corresponding optimal due-date value is equal to $R$ or coincides with the completion time of one of the groups.

**Proof.** The interchange argument used in Lemma 1 shows that there exists an optimal solution where no group is split. Moreover, since there are equal set-up times and equal processing times, the sequence of groups in nonincreasing order of $q_j$ contains the maximum number of early jobs for any value of $d$. The remaining statement of the lemma is proved following the same argumentation as in Lemma 1. □

Our algorithm for the problem $1/s_j = s$, $p_j = p/\sum U_j + r(d)$ is as follows.

**Algorithm 2**

**Step 1:** Number groups so that $q_1 \geq \ldots \geq q_n$. Set $C_0 = 0$. For $k = 1, \ldots, n$, compute $C_k = \sum_{j=1}^{k} (s + q_j p)$. If $C_n \leq R$, then stop: sequence $(1, \ldots, n)$ and $d = R$ is an optimal solution.

**Step 2:** If $C_{k-1} \leq R$ and $C_k > R$ for a certain $1 \leq k \leq n$, then compute

$$f^0 = \sum_{j=k+1}^{n} q_j + \left( (C_k - \max \{R, C_{k-1} + s\})/p \right).$$

**Step 3:** Compute $\sum_{j=m+1}^{n} q_j + r(C_m) = \min \{ \sum_{j=m+1}^{n} q_j + r(C_l) \mid l = k, \ldots, n \}$.

**Step 4:** If $\sum_{j=m+1}^{n} q_j + r(C_m) \leq f^0$, then $d = C_m$ is the optimal due-date value. Otherwise, $d = R$. In both cases, sequence $(1, \ldots, n)$ is optimal.

**Theorem 4.** Algorithm 2 solves the problem $1/s_j = s$, $p_j = p/\sum U_j + r(d)$ in $O(n \log n)$ time.

**Proof.** To prove this theorem we can employ the same argumentation as in Theorem 3. □

We now begin to study the problem with arbitrary weights.

The evident transformation from the known NP-complete problem KNAPSACK [8] shows that the problems $1/q_j = 1, s_j = 0/\sum w_j U_j + r(d)$ and $1/q_j = 1, p_j = 0/\sum w_j U_j + r(d)$ are both NP-hard. We now present an $O(n \log n)$ algorithm for the problem $1/q_j = q, s_j = s, p_j = p/\sum w_j U_j + r(d)$. This algorithm 3 is similar to algorithm 2. Its formal description is given below.

**Algorithm 3**

**Step 1:** Number groups so that $w_1 \geq \ldots \geq w_n$. Set $C_0 = 0$. For $k = 1, \ldots, n$, compute $C_k = k(s + q p)$. If $C_n \leq R$, then stop: sequence $(1, \ldots, n)$ and $d = R$ is an optimal solution.

**Step 2:** If $C_{k-1} \leq R$ and $C_k > R$ for a certain $1 \leq k \leq n$, then compute

$$f^0 = q \sum_{j=k+1}^{n} w_j + \left( (C_k - \max \{R, C_{k-1} + s\})/p \right) w_k.$$
Step 3: Compute \( q \sum_{j=m+1}^{n} w_j + r(C_m) = \min \{ q \sum_{j=l+1}^{n} w_j + r(C_l) \mid l = k, \ldots, n \} \).

Step 4: If \( q \sum_{j=m+1}^{n} w_j + r(C_m) \leq f^0 \), then \( d = C_m \) is the optimal due-date value. Otherwise, \( d = R \). In both cases, sequence \((1, \ldots, n)\) is optimal.

**Theorem 5.** Algorithm 3 solves the problem \( 1/q_j = q, s_j = s, p_j = p/\sum w_j U_j + r(d) \) in \( O(n \log n) \) time.

**Proof.** The proof is evident. \( \square \)

Thus, the complexity of almost all special cases of the problem with equal weights, equal set-up times, equal job processing times or an equal number of jobs for each group is resolved. The only open question is the complexity of the problem with equal set-up times and equal job processing times.

### 3. Dynamic programming

In this section, we present two dynamic programming algorithms to solve the general problem \( 1/\sum w_j U_j + r(d) \).

To facilitate discussion, we formulate our problem as a knapsack-type problem. By defining \( U = (U_1, \ldots, U_n) \), where \( U_j \) is a variable representing the number of tardy jobs for group \( j \), our formulation is the following:

\[
\begin{align*}
\text{Minimize} & \quad F(d, U) = r(d) + \sum_{j=1}^{n} w_j U_j, \\
\text{subject to} & \quad U_j \in \{0, 1, \ldots, q_j\}, j = 1, \ldots, n, d \geq 0, \\
& \quad C_n(U) \leq d,
\end{align*}
\]

where \( C_j(U) = \sum_{i=1}^{j} (s_i \text{sign}(q_i - U_i) + p_i(q_i - U_i)), j = 1, \ldots, n, \text{sign}(x) = 1 \text{ if } x > 0 \text{ and } \text{sign}(x) = 0 \text{ if } x = 0. \)

Constraint (5) ensures that all early jobs are completed by the due date. Let \((d^*, U^*)\) denote an optimal solution of the problem (3)–(5).

We now show that there are two essentially equivalent dynamic programming formulations for the problem (3)–(5). In the first, the weighted number of tardy jobs is a state variable and the completion time of the last early batch is a function value. However, their roles are switched in the alternative formulation.

Our first dynamic programming algorithm \( DP1 \) finds solutions \((d(f), U(f)) \) for \( f \in \{0, 1, \ldots, \sum_{j=1}^{n} w_j q_j\} \) such that \( \sum_{j=1}^{n} w_j U_j(f) = f, d(f) = C_n(U(f)) \) and \( C_n(U(f)) \leq C_n(U) \) for all solutions satisfying \( \sum_{j=1}^{n} w_j U_j = f \). In this algorithm, we recursively
compute the value of $c_j(f)$, which represents the minimum value of $C_j(U)$, subject to $\sum_{i=1}^{j} w_i U_i = f$. A formal statement of this algorithm is as follows.

**Algorithm DP1**

**Step 1:** (Initialization) Set $c_j(0) = 0$ for $j = 0$, $F = 0$ and set $c_j(f) = \infty$, otherwise. Set $j = 1$.

**Step 2:** (Recursion) Compute the following for $f = 0, 1, \ldots, \sum_{j=1}^{j} w_j q_j$:

\[
c_j(f) = \min \{ c_{j-1}(f - w_j U_j) + s_j \text{sign}(q_j - U_j) + p_j(U_j - U_j) \mid U_j = 0, 1, \ldots, q_j \}.
\]

If $j = n$, then go to Step 3; otherwise set $j = j + 1$ and repeat Step 2.

**Step 3:** (Optimal solution) For each $c_n(f) < \infty$, $f \in \{0, 1, \ldots, \sum_{j=1}^{n} w_j q_j\}$, use backtracking to find the corresponding values $U^{(f)}_1, \ldots, U^{(f)}_n$. An optimal solution is found as

\[
F(d^*, U^*) = \min \left\{ F(c_n(f), U^{(f)}) \mid f \in \{0, 1, \ldots, \sum_{j=1}^{n} w_j q_j\} \right\}.
\]

**Theorem 6.** Algorithm DP1 solves the problem (3)–(5) in $O(\sum_{j=1}^{n} q_j \sum_{i=1}^{j} w_i q_i)$ time.

**Proof.** We first show that algorithm DP1 finds solutions $(d^{(f)}, U^{(f)})$ for $f \in \{0, 1, \ldots, \sum_{j=1}^{n} w_j q_j\}$ such that $\sum_{j=1}^{n} w_j U^{(f)}_j = f$ and $C_n(U^{(f)}) \leq C_n(U)$ for all solutions satisfying $\sum_{j=1}^{n} w_j U_j = f$.

In Step 2 of the algorithm, all possible values of $U_j$ are analyzed for every group $j$. Only a partial solution having a smaller function value is retained for further consideration. Clearly, in the last iteration of Step 2, for each possible value of the total weighted number of tardy jobs $f$, we obtain values $U^{(f)}_j$ for $j = 1, \ldots, n$, such that $\sum_{j=1}^{n} w_j U^{(f)}_j = f$ and $C_n(U^{(f)}) \leq C_n(U)$ for all solutions satisfying $\sum_{j=1}^{n} w_j U_j = f$.

We now show that (6) is satisfied. Consider a certain optimal solution $(d^*, U^*)$. Set $g = \sum_{j=1}^{n} w_j U_j^*$. We have shown that there is a solution $(d^{(g)}, U^{(g)})$ such that $\sum_{j=1}^{n} w_j U^{(g)}_j = g$ and $C_n(U^{(g)}) \leq C_n(U^*)$. Due to nondecreasing of $r(d)$, algorithm DP1 sets $d^{(a)} = C_n(U^{(a)})$ for each value of $a$. Since $d^{(a)} = C_n(U^{(a)})$ and $C_n(U^*) \leq d^*$, we obtain $d^* \geq d^{(a)}$. Recalling that $(d^*, U^*)$ is an optimal solution, we have $F(d^*, U^*) = F(d^{(a)}, U^{(a)})$, i.e. (6) is satisfied. Thus, algorithm DP1 solves the problem (3)–(5).

For each group $j$, Step 2 is executed $\sum_{j=1}^{j} w_j q_j + 1$ times, each of which requires $O(q_j)$ time. Thus, the overall time complexity of DP1 is $O(\sum_{j=1}^{n} q_j \sum_{i=1}^{j} w_i q_i)$.

We now develop an alternative dynamic programming algorithm DP2 for the problem (3)–(5). In this algorithm, the completion time of the last early batch is a state variable and the weighted number of tardy jobs is a function value. More precisely, we recursively compute the value of $f_j(c)$ which represents the minimum value of $\sum_{i=1}^{j} w_i U_i$ subject to $C_j(U) = c$. Our alternative dynamic programming algorithm is as follows.
Algorithm DP2

Step 1: (Initialization) Set $f_j(c) = 0$ for $j = 0$, $c = 0$ and set $f_j(c) = \infty$, otherwise. Set $j = 1$.

Step 2: (Recursion) Compute the following for $c = 0, 1, \ldots, \sum_{i=1}^j (s_i + p_i q_i)$:

$$f_j(c) = \min \{ f_{j-1}(c - s_j \cdot \text{sign}(q_j - U_j)) - p_j (q_j - U_j) \} + w_j U_j \mid U_j = 0, 1, \ldots, q_j$$

If $j = n$, then go to Step 3; otherwise set $j = j + 1$ and repeat Step 2.

Step 3: (Optimal solution) For each $f_n(c) < \infty$, $c \in \{0, 1, \ldots, \sum_{j=1}^n (s_j + p_j q_j)\}$, use backtracking to find the corresponding values $U_1^{(c)}, \ldots, U_n^{(c)}$. An optimal solution is found as

$$F(d^*, U^*) = \min \left\{ F(c, U^{(c)}) \mid c \in \left\{ 0, 1, \ldots, \sum_{j=1}^n (s_j + p_j q_j) \right\} \right\}.$$

Theorem 7. Algorithm DP2 solves the problem (3)-(5) in $O(\sum_{j=1}^n q_j \sum_{i=1}^j (s_i + p_i q_i))$ time.

Proof. This theorem is easily proved using the same argumentation as used in the previous theorem. □

It should be noted that algorithms DP1 and DP2 do not take into consideration the specificity of the due date penalty function. Hence, they could be applied to solve the problem with an arbitrary nondecreasing penalty function $r(d)$.

4. A fully polynomial approximation scheme

In this section, we present a fully polynomial approximation scheme $\{A_c\}$ for our general problem $1/\sum w_j U_j + r(d)$.

An algorithm $A_c$ for this problem is a $(1 + \varepsilon)$-approximation algorithm if we have $F(1 + \varepsilon)F(d^*, U^*)$ for all problem instances, where $F(d^*, U^*)$ is the optimal solution value and $F$ is the value of a solution given by the algorithm. A family of algorithms $\{A_c\}$ defines a fully polynomial approximation scheme if, for any $\varepsilon > 0$, $A_c$ is a $(1 + \varepsilon)$-approximation algorithm which is polynomial in the problem instance length in binary encoding and in $1/\varepsilon$.

We first consider the problem of minimizing the total weighted number of tardy jobs with an externally given due date $d$ which we denote by $1/\sum w_j U_j$. This problem is, in fact, problem (3)-(5) where $d$ is fixed and $r(d)$ is removed from (3). Denote a fully polynomial approximation scheme for this problem by $\{B_c\}$. Algorithm $B_c$ is used in our fully polynomial approximation scheme $\{A_c\}$. Kovalyov, Potts and Van Wassenhove [11] claimed that $\{B_c\}$ can be derived from their fully polynomial approximation scheme for the problem with equal weights, $1/\sum U_j$. However, the
construction of algorithm $B_\varepsilon$ is not evident from their paper. We now present another idea to develop $B_\varepsilon$.

Let $W^*$ be the minimum solution value for the problem $1/\sum w_jU_j$. Assume that $L$ and $V$ are such numbers that $0 < L \leq W^* \leq V$. Set $\delta = \varepsilon L/n$ and formulate the following rounded problem.

Minimize $G(U) = \sum_{j=1}^{n} [w_jU_j/\delta]$, subject to (5) and

$$U_j \in \{x_j(0), x_j(1), \ldots, x_j(\lceil V/\delta \rceil)\}, j = 1, \ldots, n,$$

where $x_j(I)$ is the maximal value of $U_j \in \{0, 1, \ldots, q_j\}$ for which $[w_jU_j/\delta] = I$ is satisfied. If such a value does not exist, we set $x_j(I) = \emptyset$.

It is evident that the rounded problem can be formulated in $O(nV/\delta)$ time.

**Theorem 8.** Any exact algorithm for the rounded problem is a $(1 + \varepsilon)$-approximation algorithm for the problem $1/\sum w_jU_j$.

**Proof.** Let $U^{\text{opt}}$ be an optimal solution to the problem $1/\sum w_jU_j$ and let $U^{\text{rou}}$ be an optimal solution to the rounded problem. We first note that, by the definition of $x_j(I)$, $U^{\text{rou}}$ is a feasible solution to the problem $1/\sum w_jU_j$. Moreover, there exists a feasible solution $U'$ to the rounded problem such that

$$[w_jU'_j/\delta] = [w_jU^{\text{opt}}_j/\delta], \quad j = 1, \ldots, n.$$

It remains to show that $\sum_j w_jU^{\text{rou}}_j \leq (1 + \varepsilon)W^*$. We have

$$\sum_j w_jU^{\text{rou}}_j \leq \delta \sum_j [w_jU^{\text{rou}}_j/\delta] + n\delta \leq \delta \sum_j [w_jU'_j/\delta] + n\delta = \delta \sum_j [w_jU^{\text{opt}}_j/\delta] + n\delta \leq W^* + n\delta \leq (1 + \varepsilon)W^*. \quad \Box$$

We now present a dynamic programming algorithm for the rounded problem. In this algorithm $B_\varepsilon(L, V)$, we recursively compute the value of $c_j(g)$ which is the minimum value of $C_j(U)$, subject to $\sum_{j=1}^{n} [w_jU_j/\delta] = g$. Since $G(U^{\text{rou}}) \leq G(U^{\text{opt}}) \leq W^*/\delta \leq V/\delta$, we require $g \leq \lceil V/\delta \rceil$. A formal statement of algorithm $B_\varepsilon(L, V)$ is as follows.

**Algorithm $B_\varepsilon(L, V)$**

**Step 1:** (Initialization) Set $c_j(g) = 0$ if $j = 0$, $g = 0$ and $c_j(g) = \infty$, otherwise. Set $j = 1$. 
Step 2: (Recursion) Compute the following for \( g = 0, 1, \ldots, \lfloor V/\delta \rfloor \):
\[
c_j(g) = \min\{c_{j-1}(g - \lfloor w_j U_j/\delta \rfloor) + s_j \text{sign}(q_j - U_j) + p_j(q_j - U_j) \mid U_j \in \{x_j(0), \ldots, x_j(g)\}\}.
\]
If \( j < n \), repeat Step 2; otherwise, go to Step 3.

Step 3: (Optimal solution) Compute the optimal solution value
\[
g^0 = \min\{g \mid c_n(g) < d, \ g = 0, 1, \ldots, \lfloor V/\delta \rfloor\}
\]
and use backtracking to find the corresponding optimal solution \( U^{\text{opt}} \).

The general dynamic programming justification shows that the algorithm \( B_c(L, V) \) is correct. Its time complexity is \( O(n(V/\delta)^2) \), or equivalently, \( O((V/L)^2 n^3/\epsilon^2) \). The algorithm delivers an approximate solution \( U^{\text{opt}} \) to the problem \( 1/\sum w_j U_j \) with value \( \sum_j w_j U_j^{\text{opt}} \leq (1 + \epsilon) W^* \).

To find appropriate values for \( L \) and \( V \), we consider the problem of minimizing the maximum weighted number of tardy jobs, \( \max_j w_j U_j \), subject to given due-date \( d \). If \( W^0 \) is the optimal objective value of this problem, then \( W^0 \leq W^* \leq n W^0 \). To prove the latter inequality, it is sufficient to note that the value of an optimal solution for min-max problem calculated with respect to the min-sum problem is an upper bound for the min-sum problem. To find \( W^0 \), a bisection search procedure with \( O(n \log \max_j \{w_j q_j\}) \) running time can be derived (see [11] for details). Clearly, if \( W^0 = 0 \), then there is a schedule in which all jobs are early. In this case, \( W^* = 0 \). Assume \( W^0 > 0 \). Define \( L = W^0 \) and \( V = n W^0 \). Then algorithm \( B_c(W^0, n W^0) \) will run in \( O(n^5/\epsilon^2) \). We note that algorithm \( B_c(W^0, n W^0) \) has properties which allow us to apply the bound improvement procedure presented by Kovalyov [10] for finding such a value \( F^0 \) that \( F^0 \leq W^* \leq 3 F^0 \). This procedure will run in \( O(n^3 \log n) \) time. Algorithm \( B_c(F^0, 3 F^0) \) will run in \( O(n^3/\epsilon^2) \) time.

We define \( B_c \) as algorithm \( B_c(F^0, 3 F^0) \) with the binary search procedure and bound improvement procedure included. Algorithm \( B_c \) requires \( O(n \log \max_j \{w_j q_j\} + n^3 \log n + n^3/\epsilon^2) \) time.

We now continue the study of the problem \( 1/\sum w_j U_j + r(d) \). To construct \( A_c \), let us assume that a due-date value \( d^0 \) can be found which is related to the optimal due-date value as follows:
\[
d^0/(1 + \epsilon) \leq d^* \leq d^0.
\]
Consider the problem of minimizing the total weighted number of tardy jobs with a given due-date value \( d^0 \). Let \( U' \) be an optimal solution to this problem. Since \( d^0 \geq d^* \), we have \( \sum_{j=1}^n w_j U^*_j \leq \sum_{j=1}^n w_j U^*_j \).

Let \( B_c(d) \) denote algorithm \( B_c \) applied for the due-date value \( d \). Let \( U^0 \) be a solution given by algorithm \( B_c(d^0) \). By the definition of a \((1 + \epsilon)\)-approximation algorithm, we have
\[
\sum_{j=1}^n w_j U^0_j \leq (1 + \epsilon) \sum_{j=1}^n w_j U^*_j \leq (1 + \epsilon) \sum_{j=1}^n w_j U^*_j.
\]
From (7) and (8), we deduce that

\[
F(d^0, U^0) = \sum_{j=1}^{n} w_j U_j^0 + r(d^0) \leq (1 + \varepsilon) \left( \sum_{j=1}^{n} w_j U_j^* + r(d^*) \right). \tag{9}
\]

The inequality (9) shows that \( B_\varepsilon(d^0) \) is a \((1 + \varepsilon)\)-approximation algorithm for our general problem \( \frac{1}{n} \sum w_j U_j + r(d) \).

It remains to show how to find the value of \( d^0 \). Set \( P = \sum_{j=1}^{n} (s_j + p_j q_j) \). Clearly, \( d^* \leq P \) can be imposed. Moreover, we assume without loss of generality that \( 1 \leq d^* \). If \( d^* = 0 \), then we have \( U_j^* = q_j \) for \( j = 1, \ldots, n \). To find the value of \( d^0 \), we perform a \((1 + \varepsilon)\)-search in the range \( 1, \ldots, P \) as follows. Compute values \( d_l = \min\{(1 + \varepsilon)^l P\} \) and apply algorithm \( B_\varepsilon(d_l) \) for \( l = 1, \ldots, k \), where \( d_{k-1} < P \) and \( d_k = P \). Let \( U_l^{(l)} \) be a solution given by \( B_\varepsilon(d_l) \). It is apparent that for a certain \( 1 \leq l \leq k \) we have \( d_l/(1 + \varepsilon) \leq d^* \leq d_l \). Therefore, a solution \( (d^0, U^0) \) satisfying (9) can be found as follows.

\[
F(d^0, U^0) = \min\{F(d_l, U_l^{(l)})| l = 1, \ldots, k\}. \tag{10}
\]

Finally, we define algorithm \( A_\varepsilon \) as a sequence of algorithms \( B_\varepsilon(d_l) \) for \( l = 1, \ldots, k \), with formula (10) included. Since we have \( k \leq \log P/\log(1 + \varepsilon) \leq \max\{\log P, (\log P) / \varepsilon\} \), each algorithm \( A_\varepsilon \) runs in \( O(n \log \max\{w_j q_j\} + n^3 \log n + n^3 / \varepsilon) \) time. Thus, a family of algorithms \( \{A_\varepsilon\} \) forms a fully polynomial approximation scheme for our general problem \( \frac{1}{n} \sum w_j U_j + r(d) \).

5. Conclusion

The problem of scheduling \( n \) groups of jobs on a single machine in batches has been studied. In this problem, along with scheduling and batching decisions, a due-date common to all jobs has to be determined so as to minimize the sum of the due-date assignment penalty and the weighted number of tardy jobs: \( r(d) + \sum_{j=1}^{n} w_j U_j \).

<table>
<thead>
<tr>
<th>Problem</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/p_j = 1/ \sum U_j + r(d)</td>
<td>NP-hard</td>
</tr>
<tr>
<td>1/s_j = s/ \sum U_j + r(d)</td>
<td>NP-hard</td>
</tr>
<tr>
<td>1/q_j = q/ \sum U_j + r(d)</td>
<td>O(n log n)</td>
</tr>
<tr>
<td>1/s_j = s, p_j = p/ \sum U_j + r(d)</td>
<td>O(n log n)</td>
</tr>
<tr>
<td>1/q_j = 1, s_j = 0/ \sum w_j U_j + r(d)</td>
<td>NP-hard</td>
</tr>
<tr>
<td>1/q_j = 1, p_j = 0/ \sum w_j U_j + r(d)</td>
<td>NP-hard</td>
</tr>
<tr>
<td>1/s_j = s, p_j = p/ \sum w_j U_j + r(d)</td>
<td>Open</td>
</tr>
<tr>
<td>1/q_j = q, s_j = s, p_j = p/ \sum w_j U_j + r(d)</td>
<td>O(n log n)</td>
</tr>
</tbody>
</table>
Computational complexities of all special cases of this problem with equal job weights \( w_j \), equal job processing times \( p_j \), equal set-up times \( s_j \) or equal numbers of jobs in each group \( q_j \) are presented in Table 1.

The only open question is the complexity of the problem with equal set-up times and equal job processing times.

Two dynamic programming algorithms have been presented for the general problem as well as a fully polynomial approximation scheme.

Further research can be undertaken to resolve the remaining open question and to establish more effective approximation algorithms. Also, generalizations of the problem are of interest.

References