Batch Delivery Scheduling with Batch Delivery Cost on a Single Machine

Min Ji[1] [2]
Email: jimkeen@math.zju.edu.cn

Yong He[1]
Email: mathhey@zju.edu.cn

T. C. E. Cheng[2]
Corresponding author. Email: LGTcheng@polyu.edu.hk

[1]Department of Mathematics, State Key Lab of CAD & CG, Zhejiang University, Hangzhou 310027, P.R.China
[2]Department of Logistics, The Hong Kong Polytechnic University, Kowloon, Hong Kong

Abstract

We consider a scheduling problem in which n independent and simultaneously available jobs are to be processed on a single machine. The jobs are delivered in batches and the delivery date of a batch equals the completion time of the last job in the batch. The delivery cost depends on the number of deliveries. The objective is to minimize the sum of the total weighted flow time and delivery cost. We first show that the problem is strongly NP-hard. Then we show that, if the number of batches is B, the problem remains strongly NP-hard when $B \leq U$ for a variable $U \geq 2$ or $B \geq U$ for any constant $U \geq 2$. For the case of $B \leq U$, we present a dynamic programming algorithm that runs in pseudo-polynomial time for any constant $U \geq 2$. Furthermore, optimal algorithms are provided for two special cases: (i) jobs have a linear precedence constraint, and (ii) jobs satisfy the agreeable ratio assumption, which is valid, for example, when all the weights or all the processing times are equal.

Keywords. Single machine scheduling; Batch delivery cost; Optimal algorithm; Computational complexity

1 Introduction

Modern production technologies such as flexible manufacturing make it possible to process jobs in batches. Scheduling models in this area have given rise to the batch scheduling problem that combines partitioning jobs into batches and sequencing jobs in each batch. Most of the results in batch scheduling focus on the problem of scheduling jobs in batches on a single machine to minimize the total weighted flow time. In this problem, there is a common set-up time between consecutively scheduled batches, and the flow time of a job is equal to the completion time of its batch. Albers and Brucker [1] proved that this problem is NP-hard but polynomially solvable when the job sequence is predetermined. Polynomial time algorithms have also been presented for the cases where all job weights are equal (Coffman et al. [7]), all job processing times are equal (Albers and Brucker [1]), and both weights and processing times are equal (Nadeff and Santos [11], Coffman et al. [6], Shallcross [14]).

In this paper we study a problem that falls into a different category of the batch scheduling problem, namely the *batch delivery problem*, which was first introduced by Cheng and Kahlbacher [2]. This problem is significant and relevant to logistics and supply chain management as it addresses the issue of striking a proper balance between the rate of inventory turnover and the speed of delivery. Cheng and Kahlbacher [2] studied single machine batch delivery scheduling to minimize the sum of the total weighted earliness and delivery cost. Cheng and Gordon [3] provided a dynamic programming algorithm to solve the general problem. Cheng et al. [4] further showed that this problem can be formulated as a classical parallel machine scheduling problem, and thus the complexity results and algorithms for the corresponding parallel machine scheduling problem can be easily extended to the problem. Cheng et al. [5] studied the single machine batch delivery problem to minimize the sum of the total weighted earliness and mean batch delivery time.

While the objectives of all the prior batch delivery scheduling studies are related to job earliness, our objective is to minimize the total weighted flow time and delivery cost. To the best of our knowledge, only Wang and Cheng [16] have studied a similar problem where the objective is related to job flow time (i.e., equal weights). They studied parallel machine scheduling with batch delivery cost. They showed that the problem to minimize the sum of the total flow time and delivery cost is strongly NP-hard, and provided a dynamic programming algorithm. The algorithm is pseudo-polynomial when the number of machines is constant and the number of batches has a fixed upper bound. They also provided two polynomial time algorithms to solve the special cases where the job assignment is given or the job processing times are equal.

The problem in this paper can be formally stated as follows. We are given n independent non-preemptive jobs $\mathcal{J} = \{J_1, J_2, \cdots, J_n\}$, which must be sequenced for processing on a single machine, and partitioned into several batches for delivery. Each job J_j has an integer processing time $p_j > 0$ and a weight $w_j \geq 0$, which may be a noninteger. All the jobs are available for processing at time zero. Jobs in a batch are delivered to customers together. The batch delivery date is equal to the

completion time of the last job in the batch. Thus the flow time of a job is equal to the batch delivery date on which it is delivered. Given a partition of the jobs into B ($B \leq n$) batches and a job sequence for each batch, the job flow times $F_1, F_2, \cdots, F_n$ and the batch delivery dates $D_1, D_2, \cdots, D_B$ are easily determined. The nonnegative delivery cost $\alpha(B)$ is assumed to be a non-decreasing function of the number of batches B, with $\alpha(0) = 0$. Let S_k be a sequence of the jobs in batch k, $k = 1, 2, \cdots, B$. We denote the corresponding schedule as $\mid S_1 \mid S_2 \mid \cdots \mid S_k \mid$. The goal is to find simultaneously the number of batches B, i.e., a partition of the jobs into batches, and the job sequence in each batch such that the objective function

$$f(S_1, S_2, \cdots, S_B) = \alpha(B) + \sum_{i=1}^{n} w_i F_i$$

is minimized. Using the three-field notation introduced by Graham et al. [9], we denote our problem as $1/bd/(\alpha(B) + \sum w_i F_i)$.

Notice that when the delivery cost is negligible, the general problem simply reduces to the classical problem $1//\sum w_i F_i$. It is well-known that $1//\sum w_i F_i$ is solved by scheduling jobs in the *weighted shortest processing time* (WSPT) order, i.e., in nondecreasing order of the ratios p_i/w_i [15]. However, from the following Example 1, we can see that this order cannot guarantee yielding an optimal solution for the problem $1/bd/(\alpha(B) + \sum w_i F_i)$. Lemma 1 of Wang and Cheng [16] shows that if there exists an optimal schedule for their considered problem, then all jobs assigned to the same machine have to be scheduled in the *shortest processing time* (SPT) order. So we can conclude that all the work of Wang and Cheng [16] cannot be straightforwardly transformed to tackle our considered problem.

Example 1 : Let $p_1 = 1$, $p_2 = 3$, $p_3 = 2$; $w_1 = 2$, $w_2 = 5$, $w_3 = 3$; $\alpha(B) = 10B$. The WSPT order is $\frac{p_1}{w_1} < \frac{p_2}{w_2} < \frac{p_3}{w_3}$. Enumerate all the feasible schedules satisfying the WSPT order, we obtain that the minimal objective value is 66, which is achieved by the schedule $\mid J_1, J_2 \mid J_3 \mid$. On the other hand, the schedule $\mid J_1, J_3 \mid J_2 \mid$ yields an objective value of 65.

Batch delivery is characteristic of many practical systems in which jobs are transported and ultimately delivered to customers in containers such as boxes or carts. For such systems, an important performance objective is to minimize the work-in-process (WIP) inventories, which are related to the total weighted flow time. Furthermore, as there are always costs associated with each delivery, we face a situation that can be modelled as the above batch delivery problem.

The rest of the paper is organized as follows. In Section 2, we prove that the general problem is strongly NP-hard, and that it remains strongly NP-hard when $B \leq U$ for a variable $U \geq 2$ or $B \geq U$ for any constant $U \geq 2$. A pseudo-polynomial algorithm based on the dynamic programming is also presented for the case when $B \leq U$ for any constant $U \geq 2$. In Section 3, we identify some polynomially solvable cases. In the final section, we present some concluding remarks and suggest a few topics for future research.

2 NP-hardness and dynamic programming

Besides the general problem $1/bd/(\alpha(B)+\sum w_iF_i)$, we denote $1/bd,\theta/(\alpha(B)+\sum w_iF_i)$ as the family of problems under study, in which $\theta \subset \{\emptyset, B \le U, B \ge U, p_j = p, prec\}$ indicates the batch constraint and job characteristics. Here, $B \le U$ and $B \ge U$ means that the number of batches is bounded from above or from below, respectively, by a number U; $p_j = p$ denotes that all the processing times are equal to p; *prec* indicates that there are precedence relations $\prec$ between each pair of jobs. Moreover, for the case where all the weights are equal to w, we denote it as $1/bd/(\alpha(B) + w\sum F_i)$.

In this section we prove that the general problem $1/bd/(\alpha(B) + \sum w_iF_i)$, the problem $1/bd, B \le U/(\alpha(B) + \sum w_iF_i)$ for a variable $U \ge 2$, and the problem $1/bd, B \ge U/(\alpha(B) + \sum w_iF_i)$ for any constant $U \ge 2$, are strongly NP-hard. We also show the problem $1/bd, B \le U/(\alpha(B) + \sum w_iF_i)$ is ordinary NP-hard for any constant $U \ge 2$.

The following two lemmas can be easily established.

Lemma 1 *Let $y_k > 0$ for $1 \le k \le r$, then $\sum_{i=1}^{r} y_i^2 \ge \frac{1}{r}\left(\sum_{i=1}^{r} y_i\right)^2$, and the equality holds if and only if $y_k = \frac{1}{r}\sum_{i=1}^{r} y_i$ for $1 \le k \le r$.*

Lemma 2 *For any optimal schedule, the sequence of jobs within each batch is immaterial.*

Define the function $g_1(r) = \left(\frac{r}{2m(m-1)} + \frac{r+1}{2r}\right)(mA)^2 - r$ over $r \in (0, \infty)$, where $A \ge \sqrt{2(m-1)}$. We have the following lemma.

Lemma 3 *For any positive integer z, $g_1(z) \ge g_1(m)$, and the equality holds if and only if $z = m$.*

Proof. The derivative of the function $g(r)$ is $g_1'(r) = \left[\frac{1}{2m(m-1)} - \frac{1}{2r^2}\right](mA)^2 - 1$. Let $r_0 = \sqrt{\frac{(m-1)(mA)^2}{mA^2-2(m-1)}}$, then we get $g_1'(r_0) = 0$, $g_1'(r) > 0$ for $r > r_0$, and $g_1'(r) < 0$ for $0 < r < r_0$. Thus we conclude that $g_1(r)$ is strictly decreasing from 0 to r_0 and strictly increasing from r_0. We prove below that $m - 1 < r_0 \le m$ and $g_1(m-1) > g_1(m)$, which complete the proof.

It is clear that $r_0 = \sqrt{\frac{(m-1)(mA)^2}{mA^2-2(m-1)}} > \sqrt{\frac{(m-1)(mA)^2}{mA^2}} = \sqrt{m(m-1)} > m-1$. Since $A \ge \sqrt{2(m-1)}$, we can easily verify that $r_0 \le m$ similarly. Finally, by direct calculation, we have $g_1(m-1) - g_1(m) = 1 > 0$. □

Theorem 1 *The problem $1/bd/(\alpha(B) + \sum w_iF_i)$ is strongly NP-hard even when $\alpha(B)$ is a simple linear function of B.*

Proof. We show that the decision version of our problem is strongly NP-hard by a reduction from the 3-Partition problem, which is strongly NP-complete (Garey and Johnson [8]). An instance I of the 3-Partition problem is formulated as follows:

Given positive integers $a_1, a_2, \cdots, a_{3m}$ and A such that $\frac{A}{4} < a_j < \frac{A}{2}$ for $j = 1, 2, \cdots, 3m$ and $\sum_{j=1}^{3m} a_j = mA$, does there exist a partition of the set $X = \{1, 2, \cdots, 3m\}$ into m disjoint subsets $X_1, X_2, \cdots, X_m$ such that $\sum_{j \in X_k} a_j = A$ for $k = 1, 2, \cdots, m$?

In the above instance, we can assume that $A \geq \sqrt{2(m-1)}$ without loss of generality, since each a_j and A can be multiplied by a sufficiently large positive integer to ensure that the condition is met.

For any given instance I of the 3-Partition problem, we construct a corresponding instance II of our problem as follows:

– Number of jobs: $n = 3m$.

– Job processing times: $p_j = a_j$, $j = 1, 2, \cdots, n$.

– Job weights: $w_j = a_j$, $j = 1, 2, \cdots, n$.

– The batch delivery cost: $\alpha(B) = cB$, where $c = \frac{(mA)^2}{2m(m-1)} - 1$ is a constant.

– The threshold: $G = g_1(m) = \frac{m^2+m-1}{2m(m-1)}(mA)^2 - m$.

It is clear that the reduction can be done in polynomial time. We prove that instance I has a solution if and only if instance II has a solution with an objective value no greater than G.

If I has a solution, then we can construct a schedule with m batches, where the kth batch consists of the jobs in the set $\{J_j \mid j \in X_k\}$, $k = 1, 2, \cdots, m$. Thus, we obtain a feasible schedule with an objective value

$$\begin{aligned} cm + \sum_{j=1}^{n} w_j F_j &= cm + \sum_{k=1}^{m} \left(\sum_{j \in X_k} w_j \right) \left(\sum_{j \in X_1 \cup \cdots X_k} p_j \right) \\ &= cm + (A * A + A * 2A + \cdots + A * mA) = G. \end{aligned}$$

On the other hand, suppose II have a solution with an objective value no greater than G. We assume that the jobs are partitioned into r batches in the solution. Let Y_k be the set of jobs that are processed in the kth batch, and $y_k = \sum_{J_j \in Y_k} p_j$, $k = 1, 2, \cdots, r$. Then we also have $y_k = \sum_{J_j \in Y_k} w_j$, since $p_j = w_j$ for the constructed instance, and

$$\sum_{k=1}^{r} y_k = \sum_{j=1}^{n} p_j = mA.$$

Therefore, the objective value of the solution of instance II is

$$\begin{aligned} f(Y_1, Y_2, \cdots, Y_r) &= \alpha(r) + \sum_{j=1}^{n} w_j F_j \\ &= cr + \sum_{k=1}^{r} \left(\sum_{J_j \in Y_k} w_j \right) \left(\sum_{J_j \in Y_1 \cup \cdots \cup Y_k} p_j \right) \\ &= cr + \sum_{k=1}^{r} y_k \left(\sum_{j=1}^{k} y_j \right) \\ &= cr + \sum_{k=1}^{r} y_k^2 + \sum_{1 \leq i < j \leq r} y_i y_j \\ &= cr + \sum_{k=1}^{r} y_k^2 + \frac{1}{2} \left(\left(\sum_{k=1}^{r} y_k \right)^2 - \sum_{k=1}^{r} y_k^2 \right) \end{aligned}$$

$$= \quad cr + \frac{1}{2}\left(\sum_{k=1}^{r} y_k\right)^2 + \frac{1}{2}\sum_{k=1}^{r} y_k^2.$$

Combining the above result with Lemma 1, we obtain

$$\begin{aligned} f(Y_1, Y_2, \cdots, Y_r) &\geq cr + \frac{1}{2}\left(\sum_{k=1}^{r} y_k\right)^2 + \frac{1}{2}\left(\frac{1}{r}\left(\sum_{k=1}^{r} y_k\right)^2\right) \\ &= \left(\frac{(mA)^2}{2m(m-1)} - 1\right) r + \frac{r+1}{2r}\left(\sum_{k=1}^{r} y_k\right)^2 \\ &= \left(\frac{(mA)^2}{2m(m-1)} - 1\right) r + \frac{r+1}{2r}(mA)^2 \\ &= \left(\frac{r}{2m(m-1)} + \frac{r+1}{2r}\right)(mA)^2 - r, \end{aligned}$$

where the equality holds if and only if $y_k = \frac{1}{r}(\sum_{i=1}^{r} y_i) = \frac{1}{m}(mA) = A, k = 1, \cdots, r$.

By Lemma 3, we have $f(Y_1, Y_2, \cdots, Y_r) \geq g_1(m) = G$, and the equality holds if and only if $r = m$. Hence, by the assumption that instance II has a solution with an objective value no greater than G, we conclude that all the jobs are partitioned into m batches, and the total processing time of each batch satisfies $y_k = A, k = 1, \cdots, m$. This yields a solution for instance I, which completes the proof of Theorem 1. □

By applying a similar method of reduction, we conclude that the problem $1/bd, B \geq U/(\alpha(B) + \sum w_i F_i)$ is strongly NP-hard if U is a constant no smaller than 2, and the problem $1/bd, B \leq U/(\alpha(B) + \sum w_i F_i)$ is strongly NP-hard, too, if U is a variable no smaller than 2. For the former problem, the only modification of the above proof is that we set U as any integer satisfying $U \leq m$ in constructing instance II. For the latter problem, the proof is completely the same. The following theorem shows that the latter problem remains NP-hard in the ordinary sense even if $U = 2$.

Theorem 2 *The problem* $1/bd, B \leq U/(\alpha(B) + \sum w_i F_i)$ *is NP-hard for any constant* $U \geq 2$ *even when* $\alpha(B)$ *is a simple linear function of* B.

Proof. The proof is similar to that of Theorem 1. A reduction from the NP-hard Partition problem (Garey and Johnson [8]) is used. An instance I of the Partition problem is formulated as follows:

Given positive integers $a_1, a_2, \cdots, a_n$ and A such that $\sum_{j=1}^{n} a_j = 2A$, does there exist a partition of the set $X = \{1, 2, \cdots, n\}$ into 2 disjoint subsets X_1, X_2 such that $\sum_{j \in X_k} a_j = A$ for $k = 1, 2$?

In the above instance, we assume that $A \geq \sqrt{2}$ without loss of generality. For any given instance I of the Partition problem, we construct a corresponding instance II of our problem as follows:

– Number of jobs: n.

– Job processing times: $p_j = a_j$, for $j = 1, 2, \cdots, n$.

– Job weights: $w_j = a_j$, for $j = 1, 2, \cdots, n$.

– The batch delivery cost: $\alpha(B) = cB$, where $c = A^2 - 1$ is a constant.

– The upper bound of the batch number: U, which is an arbitrary constant no less than 2.

– The threshold: $G = 5A^2 - 2$.

It is clear that the reduction can be done in polynomial time. It can be shown similarly that instance I has a solution if and only if instance II has a solution with an objective value no greater than G. □

We next present a pseudo-polynomial time algorithm BDP based on dynamic programming, which shows that the problem $1/bd, B \leq U/(\alpha(B) + \sum w_i F_i)$ is NP-hard in the ordinary sense only.

Define $H_j(y_1, y_2, \cdots, y_B)$ as the minimum objective value, given (i) we have assigned jobs $J_1, J_2, \cdots, J_j$ to B batches, and (ii) the total processing time of the jobs in the kth batch is equal to y_k for $k = 1, 2, \cdots, B$. Set $T_j = \sum_{k=1}^{j} p_j$ for $j = 1, 2, \cdots, n$. We provide a formal description of algorithm BDP as follows.

Algorithm BDP:

Step 1 (Initialization) For $B = 1, 2, \cdots, U$, set $H_j(y_1, y_2, \cdots, y_B) = \infty$ for $j = 0, 1, \cdots, n$, and $0 \leq y_k \leq T_n$, $k = 1, 2, \cdots, B$. Set $H_0(0) = 0$ and $j = 1$.

Step 2 (Recursion) For $B = 1, 2, \cdots, \min\{j, U\}$, the recursion is

$$H_j(y_1, y_2, \cdots, y_B) = \min_{\{k|1 \leq k \leq B\}} \min$$

$$\begin{cases} H_{j-1}(y_1, \cdots, y_{k-1}, y_k - p_j, y_{k+1}, \cdots, y_B) + w_j(y_1 + y_2 + \cdots + y_k), & \text{if } p_j < y_k \leq T_j, \\ H_{j-1}(y_1, \cdots, y_{k-1}, y_{k+1}, \cdots, y_B) + w_j(y_1 + y_2 + \cdots + y_k) + \alpha(B) - \alpha(B-1), & \text{if } y_k = p_j, \\ \infty & \text{if } p_j > y_k. \end{cases} \quad (1)$$

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

Step 3 (Output) The optimal value is determined as

$$H^* = \min_{B=1,2,\cdots,U} \{H_n(y_1, y_2, \cdots, y_B) \mid 0 \leq y_k \leq T_n, k = 1, 2, \cdots, B\},$$

and backtracking can be used to find the corresponding optimal solution.

Some remarks should be made about algorithm BDP. From Lemma 2, we know that arranging the order of jobs in advance in the initialization step is unnecessary. The three quantities on the right-hand side of (1) represent three possible scheduling choices when we assign job J_j to batch k: (i) add job J_j to the existing batch k; (ii) form a new batch k consisting of the sole job J_j; (iii) do not assign job J_j to batch k. Therefore, at each stage of the algorithm we only need to decide whether to add job J_j to batch k, and if so, whether batch k is new or not. Based on the dynamic programming justification for scheduling problems (Rothkopf [12]; Lawler and Moore [10]), it is easy to see that algorithm BDP solves our problem $1/bd, B \leq U/(\alpha(B) + \sum w_i F_i)$ optimally.

We establish the time complexity of algorithm BDP as follows. In each iteration of Step 2, only $B - 1$ of the values $y_1, y_2, \cdots, y_B$ are independent, since $y_1 + y_2 + \cdots + y_B = T_j$ when we take into account job J_j. Hence, in the iteration of Step 2 for J_j, the number of different tuples $(y_1, y_2, \cdots, y_B)$

for $B = 1, \cdots, U$ is at most UT_j^{U-1}. For each tuple $(y_1, y_2, \cdots, y_B)$, the right-hand side of (1) can be calculated in $O(B)$ time. Thus, Step 2 takes $O(nU^2T_n^{U-1})$ time, which is the overall time complexity of algorithm DBP, too. Therefore we conclude that the problem $1/bd, B \leq U/(\alpha(B) + \sum w_iF_i)$ is indeed only ordinary NP-hard for any constant $U \geq 2$.

3 Polynomially solvable cases

In this section we consider some special cases of the general problem. We first show that the strongly NP-hard problem $1/bd/(\alpha(B) + \sum w_iF_i)$ becomes polynomial time solvable when all the jobs have a linear precedence constraint. In fact, by appropriately modifying the algorithm provided by Coffman et al. [7] and the algorithm presented by Albers and Brucker [1], the problem $1/bd, prec/(\alpha(B) + \sum w_iF_i)$ can be solved in $O(n)$ time. These two algorithms, which are based on the backward dynamic programming method and run in linear time, can solve the batch scheduling problem where the job sequence is predetermined. We show that, for the general problem with the *agreeable ratio* assumption, i.e., $p_i \leq p_j$ implies $w_i \geq w_j$, there is an optimal solution with all the jobs being scheduled in the $WSPT$ order. Therefore, this special case reduces to the case where all the jobs have a linear precedence constraint, and thus can be solved in $O(n \log n)$ time. Finally, it is valid to conclude that the case where all the job weights are equal or all the job processing times are equal can be solved in $O(n \log n)$ time, too.

Without loss of generality, we assume that the linear precedence constraint of the jobs is $J_1 \prec J_2 \prec \cdots \prec J_n$. Define $\beta_j = p_j + p_{j+1} + \cdots + p_n$ and $\gamma_j = w_j + w_{j+1} + \cdots + w_n$ for $j = 1, 2, \cdots, n$. For any $k < l \leq n+1$, let $H(k, l)$ and E_k be the objective value and the batch number in an optimal schedule for $J_k, J_{k+1}, \cdots, J_n$ with job J_l being the first job of the second batch, i.e., the optimal schedule is in the form of $\mid J_k \cdots J_{l-1} \mid J_l \cdots$. The second batch is empty if $l = n+1$ and $\mid J_k \cdots J_n \mid$ is the schedule corresponding to $H(k, n+1)$. We define $Z_k = \min_{l>k} H(k, l)$ for $k = 1, 2, \cdots, n$, and set $Z_{n+1} = 0$, $\beta_{n+1} = 0$, $E_{n+1} = 0$. Our goal is to compute Z_1. We obtain the following recursion formula

$$\begin{cases} H(k,l) = Z_l + \gamma_k(\beta_k - \beta_l) + \alpha(E_l + 1) - \alpha(E_l), \quad l > k, \\ Z_k = \min_{l>k} H(k,l) = \min_{l>k}\{\gamma_k(\beta_k - \beta_l) + Z_l + \alpha(E_l + 1) - \alpha(E_l)\}, \\ E_k = E_{l_0} + 1 \quad if \quad Z_k = H(k, l_0). \end{cases}$$
$$k = 1, 2, \cdots, n. \tag{2}$$

Noting that the increment in the objective value is $\gamma_k(\beta_k - \beta_l) + \alpha(E_l + 1) - \alpha(E_l)$ if we add a new batch consisting of $J_k, J_{k+1}, \cdots, J_{l-1}$, so the first equation in (2) is true. The validity of the remaining part of the equation is trivial.

Based on (2), we can construct a standard dynamic programming algorithm. However, such a solution solves our problem in time quadratic in n, since for any fixed k, $k = n, n-1, \cdots, 1$ we need to calculate $H(k, l)$, $l = k+1, k+2, \cdots, n+1$. In the remainder of this section, we give a linear time optimal algorithm based on the backward dynamic programming method.

Lemma 4 *Let $k < l < m$, with $m \leq n+1$. If $H(k,l) \leq H(k,m)$, then $H(j,l) \leq H(j,m)$ for any $1 \leq j \leq k$.*

Proof. From (2), we obtain

$$\begin{aligned} g_2(k) &\doteq H(k,m) - H(k,l) \\ &= \gamma_k(\beta_l - \beta_m) - (Z_l - Z_m) + (\alpha(E_m+1) - \alpha(E_m)) - (\alpha(E_l+1) - \alpha(E_l)). \end{aligned} \tag{3}$$

Because $\beta_l > \beta_m$ and $-(Z_l - Z_m) + (\alpha(E_m+1) - \alpha(E_m)) - (\alpha(E_l+1) - \alpha(E_l))$ is constant with regard to k, the function $g_2(k)$ increases as k decreases. Thus the conclusion follows. □

Setting (3) to zero, we define the "threshold" by solving γ_k as

$$\delta(l,m) = \left\lceil \frac{(Z_l - Z_m) + (\alpha(E_l+1) - \alpha(E_l)) - (\alpha(E_m+1) - \alpha(E_m))}{\beta_l - \beta_m} \right\rceil. \tag{4}$$

By Lemma 4, we conclude that $H(k,l) \leq H(k,m)$ for any k with $\gamma_k \geq \delta(l,m)$ (we say that m is not better than l with respect to k), and $H(k,l) > H(k,m)$ for all k with $\gamma_k < \delta(l,m)$ (we say that m is better than l with respect to k).

Lemma 5 *Let $k < l < m$, with $m \leq n+1$. If $\delta(k,l) \leq \delta(l,m)$, then for any j, $j \leq k$, either l is not better than k or m is better than l, with respect to j.*

Proof. If $\gamma_j \geq \delta(k,l)$, then l is not better than k with respect to j. On the other hand, if $\gamma_j < \delta(k,l)$, then $\gamma_j < \delta(l,m)$ and m is better than l with respect to j. □

The above two lemmas are crucial for our algorithm. We now explain the algorithm, which is described in the following. We calculate Z_k for all $k = 1, 2, \cdots, n$. Starting with job J_n, we scan all the jobs in decreasing order of their indices. When Z_k for some k needs to be computed, the values of Z_l, $l = j+1, j+2, \cdots, n$, have already been calculated and therefore it suffices to determine an immediate successor $l > k$ as the index of the first job of the second batch for jobs $J_k, \cdots, J_n$. Such a job index l is computed with the help of a queue q.

In this algorithm, we have two $(n+1)$-element vectors z_j and q_j, $1 \leq j \leq n+1$. z_j is the objective value yielded by the algorithm right after jobs $J_j, \cdots, J_n$ have been processed. It is obvious to define $z_{n+1} = 0$, which is the objective value of an empty sequence. The algorithm calculates z_j, $1 \leq j \leq n$, in decreasing order of their indices. Theorem 3 below verifies that $z_j = Z_j$, $1 \leq j \leq n$. We use q as a queue. Right before z_k is computed, it contains indices exceeding k, which are candidates (possible indices) for the first job of the second batch in an optimal solution for $J_k, J_{k+1}, \cdots, J_n$. At any time, the elements of q are stored as $q_t, q_{t+1}, \cdots, q_h$, in which we call q_h the head and q_t the tail of the queue q. In general, elements are removed from both ends of q, while new elements are only appended to the tail end. We denote $\mid q \mid = h - t + 1$ as the length of q. Initially, q contains the single element $n+1$. We also have two other n-element vectors η_j and δ_j , $1 \leq j \leq n$. η_j records the index of the first job of the second batch in an optimal solution right after jobs $J_j, J_{j+1}, \cdots, J_n$ have

been scheduled. Then from the algorithm, the optimal solution can be recovered from η: η_1 indexes the first job of the second batch, then η_j with $j = \eta_1$ indexes the first job of the third batch, and so on. δ_j is only a denotation, which is defined as $\delta_j = \delta(q_j, q_{j+1})$, $t \le j \le h-1$, if $\mid q \mid > 1$.

Algorithm *LBDP*.

Step 1 (Initialization) $\beta_{n+1} = 0$, $\gamma_{n+1} = 0$, $z_{n+1} = E_{n+1} = 0$, $q_{n+1} = n+1$, $h = t = n+1$. For $j = n, n-1, \cdots, 1$, do $\beta_j = \beta_{j+1} + p_j$ and $\gamma_j = \gamma_{j+1} + w_j$.

Step 2 (Recursion) $k = n$;

while $(k > 0)$, do

begin

(1) while ($t < h$ and $\gamma_k \ge \delta_{h-1}$), do $h = h - 1$. {delete the head of q}

(2) $z_k = H(k, q_h)$, $E_k = E_{q_h} + 1$, $\eta_k = q_h$. {compute the objective value and the batch numbers for $J_k, \cdots, J_n$, and set the index of the first job of the second batch}

(3) if $\delta(k, q_t) \le \gamma_1$, then

begin

(i) while ($t < h$ and $\delta(k, q_t) \le \delta_t$), do $t = t + 1$. {if k is a new candidate, then delete the elements from the tail of q, if any, that can no longer be candidates (by Lemma 5)}

(ii) $t = t - 1$, $q_t = k$, $\delta_t = \delta(q_t, q_{t+1})$. {append k to q and set δ_t}

end

$k = k - 1$.

end

Step 3 (Output) z_1 is the optimal value, and we can use backtracking to find the corresponding optimal solution.

Theorem 3 *In algorithm LBDP, $z_k = Z_k$, $k = 1, 2, \cdots, n$, and η_k, $k = 1, 2, \cdots, n$, indexes the first job of the second batch in an optimal schedule for $J_k, \cdots, J_n$. Thus the problem $1/bd, prec/(\alpha(B) + \sum w_i F_i)$ is solved by algorithm LBDP in $O(n)$ time.*

Proof. The indices are deleted from the head and the tail of the queue, and appended only to the tail, so right after step 2(3) we have

$$k < q_t < q_{t+1} < \cdots < q_h \le n+1,$$

and hence

$$\gamma_k > \gamma_{q_t} > \gamma_{q_{t+1}} > \cdots > \gamma_{q_h}. \tag{5}$$

In step 2(3)(i), if $t < h$ and $\delta(k, q_t) \le \delta_t$, we conclude by Lemma 5 that q_t can never be the best candidate. Hence, deleting it from the tail of q is appropriate. The algorithm continues to delete

the tail of q in step 2(3)(i) by the same reason until $\delta(k, q_r) > \delta_r$ for some $r > 1$ or q contains only q_h. So right after step 2(3)(ii), we ensure that

$$\gamma_1 \geq \delta_t > \delta_{t+1} > \cdots > \delta_{h-1} \geq \gamma_{n+1}. \tag{6}$$

In step 2(1), if $t < h$ and $\gamma_k \geq \delta_{h-1}$, then by Lemma 4, we have $H(i, q_{h-1}) \leq H(i, q_h)$ for all $i \leq k$. It follows that q_h can never be the best candidate, and hence deleting it from q is appropriate. The algorithm continues to delete the head of q in step 2(1) by the same reason until finally $\gamma_k < \delta_r$ for some $r \geq t$, or $r = t$ and q contains only q_t. Right after step 2(1), we have $h = r + 1$, $\eta_k = q_h = q_{r+1}$, and

$$\delta_t > \delta_{t+1} \cdots > \delta_{h-1} > \gamma_k, \tag{7}$$

because of (6). Combining (5) and (7), we obtain

$$\delta_t > \delta_{t+1} \cdots > \delta_{h-1} > \gamma_k > \gamma_{q_t} > \gamma_{q_{t+1}} > \cdots > \gamma_{q_h}, \tag{8}$$

right after step 2(1), which implies

$$H(k, q_v) > H(k, q_{v+1}) \text{ for } v = t, \cdots, h-1.$$

It follows that q is ordered by the better-than relation with respect to k from the head to the tail. Thus, among all the indices stored in q, q_h is the best one for indexing the first job of the second batch for $J_k, \cdots, J_n$ right after step 2(1). So we obtain

$$z_k = H(k, q_h) = \min_{l \in q} H(k, l). \tag{9}$$

From the initialization, we immediately have $z_n = Z_n$ and $\eta_n = n + 1$, as desired, right after step 2(2) is executed for the first time. In order to verify $z_k = Z_k$ for $1 \leq k < n$, as we have already obtained (9), it suffices to show that there is no $l \notin q$, $k < l \leq n + 1$, such that $H(k, l) < H(k, q_h)$ right before the $(n - k + 1)$th iteration of step 2(2). To see this, let $l_0 > k$ be an index that minimizes $H(k, l)$ and suppose $l_0 \notin q$ when computing z_k in step 2(2). We first claim that $l_0 \neq n+1$. If $l_0 = n + 1$, then l_0 is initially in q. Because $l_0 \notin q$ when z_k is calculated, l_0 must be deleted from the head of q in some iteration of step 2 and therefore l_0 can never be a candidate by Lemma 4. This contradicts the assumption that l_0 minimizes $H(k, l)$. If $l_0 < n + 1$, one reason why $l_0 \notin q$ is because it is never appended to q. By step 2(3), this means that in the $(n - l_0 + 1)$th iteration, $\delta(l_0, q_t) > \gamma_1$, which by Lemma 4, implies that l_0 can never be the best candidate with respect to k, for all $1 \leq k < l_0$. That is to say, $H(k, l_0) > H(k, m)$, where $m = q_t$ when z_k is calculated. This contradicts the assumption that l_0 minimizes $H(k, l)$, too. This completes the proof that $z_k = Z_k$, $k = 1, 2 \cdots, n$, and therefore η_k indexes the first job of the second batch in an optimal schedule for jobs $J_k, \cdots, J_n$.

We now investigate the complexity of the algorithm. Each index $1 \leq k \leq n$ is added and deleted at most once from q. Therefore, the operations in step 2 are performed at most n times, and we conclude that algorithm $LBDP$ runs in linear time. □

The following example illustrates the algorithm in detail:

Example 2 : $p_1 = p_2 = 2$, $p_3 = p_4 = 5$, $p_5 = p_6 = 10$; $w_1 = 1$, $w_2 = w_3 = 2$, $w_4 = w_5 = w_6 = 3$; $\alpha(B) = 20B$. Initially, we have $t = h = 7$, $q = (q_t, \cdots, q_h) = (7)$, and $z_7 = E_7 = 0$. The recursions are as follows:

$k = 6$ The first iteration begins with $q = (q_7) = (7)$.
$z_6 = H(6,7) = \gamma_6(\beta_6 - \beta_7) + z_7 + \alpha(1) - \alpha(0) = 50$, $E_6 = E_7 + 1 = 1$, $\eta_6 = 7$,
$\delta(6,7) = \left\lceil \frac{z_6 - z_7}{\beta_6 - \beta_7} \right\rceil = 5$, $t = t - 1 = 6$, $q_6 = 6$.
6 is appended to the queue since $\delta(k, q_t) \leq \gamma_1$ and hence $q_t = 6$ is a new candidate.
$\delta_6 = \delta(6,7) = 5$.

$k = 5$ $h = h - 1 = 6$, q_7 is deleted from the head of the queue since $\gamma_5 \geq \delta_6$ and hence q_6 is better than q_7.
$z_5 = H(5,6) = \gamma_5(\beta_5 - \beta_6) + z_6 + \alpha(2) - \alpha(1) = 130$, $E_5 = E_6 + 1 = 2$, $\eta_5 = 6$,
$\delta(5,6) = \left\lceil \frac{z_5 - z_6}{\beta_5 - \beta_6} \right\rceil = 8$, $t = t - 1 = 5$, $q_5 = 5$.
5 is appended to the queue with $\delta_5 = \delta(5,6) = 8$.

$k = 4$ $h = h - 1 = 5$, q_6 is deleted from the head of the queue similarly.
$z_4 = H(4,5) = \gamma_4(\beta_4 - \beta_5) + z_5 + \alpha(3) - \alpha(2) = 195$, $E_4 = E_5 + 1 = 3$, $\eta_4 = 5$,
$\delta(4,5) = \left\lceil \frac{z_4 - z_5}{\beta_4 - \beta_5} \right\rceil = 13$, $t = t - 1 = 4$, $q_4 = 4$.
4 is appended to the queue with $\delta_4 = \delta(4,5) = 13$.

$k = 3$ $z_3 = H(3,5) = \gamma_3(\beta_3 - \beta_5) + z_5 + \alpha(3) - \alpha(2) = 260$, $E_3 = E_5 + 1 = 3$, $\eta_3 = 5$,
$\delta(3,4) = \left\lceil \frac{z_3 - z_4}{\beta_3 - \beta_4} \right\rceil = 13 \leq \delta_4$, $t = t + 1 = 5$.
$q_4 = 4$ is deleted from the tail of the queue by the step 2(3)(i) of the algorithm.
$t = t - 1 = 4$; $q_4 = 3$.
3 is appended to the queue, and $\delta_4 = \delta(3,5) = \left\lceil \frac{z_3 - z_5}{\beta_3 - \beta_5} \right\rceil = 13$.

$k = 2$ $h = h - 1 = 4$, q_5 is deleted from the head of the queue similarly.
$z_2 = H(2,3) = \gamma_2(\beta_2 - \beta_3) + z_3 + \alpha(4) - \alpha(3) = 306$, $E_2 = E_3 + 1 = 4$, $\eta_2 = 3$,
$\delta(2,3) = \left\lceil \frac{z_2 - z_3}{\beta_2 - \beta_3} \right\rceil = 23 > \gamma_1$.
So 2 is not appended to the queue.

$k = 1$ $z_1 = H(1,3) = \gamma_1(\beta_1 - \beta_3) + z_3 + \alpha(4) - \alpha(3) = 336$, $E_1 = E_3 + 1 = 4$, $\eta_1 = 3$,
$\delta(1,3) = \left\lceil \frac{z_1 - z_3}{\beta_1 - \beta_3} \right\rceil = 19$.

Therefore, we get

k :	6	5	4	3	2	1
q at the end of the $(n-k+1)$th iteration :	$(6,7)$	$(5,6)$	$(4,5)$	$(3,5)$	(3)	(3)
η_k :	7	6	5	5	3	3
E_k :	1	2	3	3	4	4.

Using backtracking, we obtain an optimal schedule $\mid J_1, J_2 \mid J_3, J_4 \mid J_5 \mid J_6 \mid$, and the corresponding optimal value is 336.

We now consider the general problem under the *agreeable ratio* assumption, i.e., $p_i \leq p_j$ implies $w_i \geq w_j$.

Lemma 6 *For the general problem* $1/bd/(\alpha(B) + \sum w_i F_i)$ *under the agreeable ratio assumption, there is an optimal schedule in which all the jobs are sequenced in the* $WSPT$ *order.*

Proof. It is clear that in an optimal schedule, batch completion times occur only at job completion times. So it is routine to verify that, if job J_j is scheduled after J_i and $\frac{p_j}{w_j} < \frac{p_i}{w_i}$, then interchanging J_i and J_j does not increase the objective value. Iterating such interchanges leads to the conclusion. □

Combining Theorem 3 and Lemma 6, we obtain the following theorem.

Theorem 4 *The general problem* $1/bd/(\alpha(B) + \sum w_i F_i)$ *under the agreeable ratio assumption is solved in* $O(n \log n)$ *time.*

Proof. From Lemma 6, we reduce the special case to $1/bd, prec/(\alpha(B) + \sum w_i F_i)$ in $O(n \log n)$ time. Then applying algorithm $LBDP$ solves the problem in linear time. □

For the case where all the job weights are equal or all the job processing times are equal, Theorem 4 directly implies the following corollary.

Corollary 1 *The special cases* $1/bd/(\alpha(B) + w \sum F_i)$ *and* $1/bd, p_j = p/(\alpha(B) + \sum w_i F_i)$ *are solved in* $O(n \log n)$ *time.*

Proof. It is easy to verify that the problems $1/bd/(\alpha(B)+w \sum F_i)$ and $1/bd, p_j = p/(\alpha(B)+\sum w_i F_i)$ satisfy the agreeable ratio assumption. So by Theorem 4, we obtain the conclusion trivially. In fact, equation (2) becomes simpler here and the same for algorithm $LBDP$. □

4 Conclusions

We showed that the problems $1/bd/(\alpha(B) + \sum w_i F_i)$, $1/bd, B \leq U/(\alpha(B) + \sum w_i F_i)$ for a variable $U \geq 2$, and $1/bd, B \geq U/(\alpha(B) + \sum w_i F_i)$ for any constant $U \geq 2$ are strongly NP-hard. We also showed that the problem $1/bd, B \leq U/(\alpha(B) + \sum w_i F_i)$ for any constant $U \geq 2$ is NP-hard in the ordinary sense. We presented a pseudo-polynomial time algorithm based on dynamic programming for the case with a limited number of batches. For the case where the jobs have a linear precedence constraint, we designed a linear time algorithm to solve it. We presented an algorithm with $O(n \log n)$ running time for the case where the jobs satisfy the agreeable ratio assumption, which is valid, for example, when all the job weights or all the job processing times are equal.

For future research, it will be worth considering the design of efficient and effective heuristics for the general problem. It is also worth extending the existing model to models with multiple machines and develop solution methods for them.

Acknowledgement

This research was supported in part by the Hong Kong Polytechnic University under grant number G-T997, the Teaching and Research Award Program for Outstanding Young Teachers in Higher Education Institutions of the MOE, China, and the National Natural Science Foundation of China (10271110, 60021201).

References

[1] S. Albers, P. Brucker, The complexity of one-machine batching problems, *Discrete Applied Mathematics*, 47 (1993) 87-107.

[2] T.C.E. Cheng, H.G. Kahlbacher, Scheduling with delivery and earliness penalties, *Asia-Pacific Journal of Operational Research*, 10 (1993) 145-152.

[3] T.C.E. Cheng, V.S. Gordon, Batch delivery scheduling on a single machine, *Journal of the Operational Research Society*, 45 (1994) 1211-1215.

[4] T.C.E. Cheng, V.S. Gordon, M.Y. Kovalyov, Single machine scheduling with batch deliveries, *European Journal of Operational Research*, 94 (1996) 277-283.

[5] T.C.E. Cheng, M.Y. Kovalyov, B.M.T. Lin, Single machine scheduling to minimize batch delivery and job earliness penalties, *SIAM Journal on Optimization*, 7 (1997) 547-559.

[6] E.G.Coffman, A. Nozari, M. Yannakakis, Optimal scheduling of products with two subassemblies on a single machine, *Operations Research*, 37 (1989) 426-436.

[7] E.G. Coffman, M. Yannakakis, M.J. Magazine, C. Santos, Batch sizing and sequencing on a single machine, *Annals of Operations Research*, 26 (1990) 135-147.

[8] M.R. Garey, D.S. Johnson, A guide to the theory of NP-completeness, W.H. Freeman, San Francisco, CA, 1979.

[9] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, *Annals of Operations Research*, 5 (1979) 287-326.

[10] E.L. Lawler, J.M. Moore, A functional equation and its application to resource allocation and sequencing problems, *Management Science*, 16 (1969) 77-84.

[11] D. Naddef, C. Santos, One-pass batching algorithms for the one-machine problem, *Discrete Applied Mathematics*, 21 (1988) 133-146.

[12] M.H. Rothkopf, Scheduling independent tasks on parallel processors, *Management Science*, 12 (1966) 437-447.

[13] C. Santos, M.J. Magazine, Batching in single operation manufacturing systems, *Operations Research Letters*, 4 (1985) 99-104.

[14] D. Shallcross, A polynomial algorithm for a one machine batching problem, *Operations Research Letters*, 11 (1992) 213-218.

[15] W. E. Smith, Various optimizers for single-stage production, *Naval Research Logistics Quarterly*, 3 (1956), 59-66.

[16] G.Q. Wang, T.C.E. Cheng, Parallel machine scheduling with batch delivery costs, *International Journal of Production Economics*, 68 (2000) 177-183.