SIAM J. OPTIM.
Vol. 7, No. 2, pp. 547–559, May 1997

016

SINGLE MACHINE SCHEDULING TO MINIMIZE BATCH DELIVERY AND JOB EARLINESS PENALTIES*

T. C. EDWIN CHENG†, MIKHAIL Y. KOVALYOV‡, AND BERTRAND M.-T. LIN§

Abstract. We study a problem in which a set of n jobs has to be batched as well as scheduled for processing on a single machine. A constant machine set-up time is required before the first job of each batch is processed. A schedule specifies the sequence of batches, where each batch comprises a sequence of jobs. The batch delivery time is defined as the completion time of the last job in a batch. The earliness of a job is defined as the difference between the delivery time of the batch to which it belongs and the job completion time. The objective is to find a number B of batches and a schedule so as to minimize the sum of the total weighted job earliness and mean batch delivery time. The problem is shown to be strongly NP-hard. It remains strongly NP-hard if the set-up time is zero and $B \leq U$ for any variable $U \geq 2$ or if $B \geq U$ for any constant $U \geq 2$. The problem is proved to be ordinary NP-hard even if the set-up time is zero and $B \leq 2$. For the case $B \leq U$, a dynamic programming algorithm is presented, which is pseudopolynomial for any constant $U \geq 2$. Algorithms with $O(n^2)$ running times are derived for the cases when all weights are equal or all processing times are equal. For the general problem, a family of heuristics is suggested. Computational experiments on the proposed heuristic algorithm are conducted. The results suggest that the heuristics are effective in generating near-optimal solutions quickly.

Key words. single machine scheduling, batch scheduling, NP-hardness, dynamic programming, polynomial algorithms

AMS subject classifications. 68Q25, 90C39

PII. S1052623494269540

1. Introduction.

Processing jobs in batches is a common practice in flexible manufacturing. Scheduling models which combine partitioning jobs into batches and sequencing jobs in each batch have been extensively studied lately. Most of the results in the batch scheduling area are obtained for the problem of scheduling jobs in batches on a single machine to minimize the total weighted job completion time. In this problem, there is a common set-up time between consecutively scheduled batches, and the completion time of a job is equal to the completion time of its batch, so all jobs in the same batch are completed at the same time. 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 when all job weights are equal (Coffman, Yannakakis, Magazine, and Santos [8]) all processing times are equal (Albers and Brucker [1]), and both weights and processing times are equal (Nadeff and Santos [17]; Coffman, Nozari, and Yannakakis [9]; Shallcross [19]).

* Received by the editors June 13, 1994; accepted for publication (in revised form) November 2, 1995.

http://www.siam.org/journals/siopt/7-2/26954.html

† Office of the Vice President (Research & Postgraduate Studies), The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong (mscheng@polyu.edu.hk). This author was supported in part by the Hong Kong Research Grants Council under grant HKP54/94E.

‡ Institute of Engineering Cybernetics, Belarus Academy of Sciences, Surganova 6, 220012 Minsk, Belarus (koval@newman.basnet.minsk.by). This author was supported in part by the International Association for the Promotion of Cooperation with Scientists from the Independent States of the former Soviet Union under grant 93-0257.

§ Department of Information Management, Ming-Chuan College, Taipei 11120, Taiwan, Republic of China (mtlin@mcu.edu.tw). This author was supported in part by the Republic of China National Science Foundation under grant 84-2213-E13-5.

In this paper, we introduce a scheduling problem with batch delivery and job earliness penalties, which may be stated as follows. There are n jobs to be scheduled on a single machine. Each job j has an integer processing requirement $p_j > 0$ and a weight $w_j \geq 0$, which may be a noninteger. Jobs may be combined to form batches containing contiguously scheduled jobs. For each batch, a constant machine set-up time $s \geq 0$ is required before the first job of the batch is processed. The machine can handle only one job at a time and cannot process any jobs while a set-up is performed. All jobs in the same batch are delivered to the customer together upon the completion of the last job in the batch.

Given a number B of batches, a schedule specifies the sequence $1, \ldots, B$ of these batches, where each batch b is a sequence of jobs it contains. Given a number of batches and a schedule, the completion time C_j of each job j is easily determined. It is measured from the beginning of the scheduling horizon, i.e., from time zero. We define the batch b delivery time D_b as the completion time of the last job in the batch and the earliness E_j of job j in batch b as the difference between the delivery time of batch b and the completion time of job j: $E_j = D_b - C_j$ if $j \in b$.

The objective is to find an optimal number B of batches and an optimal schedule so as to minimize the sum of the total weighted job earliness and mean batch delivery time:

$$\sum_{j=1}^{n} w_j E_j + \sum_{b=1}^{B} D_b/B.$$

This problem is closely related to the single machine scheduling problem to minimize the total weighted job earliness plus a batch delivery penalty depending only on the number of batches: $\sum_{j=1}^{n} w_j E_j + \gamma(B)$, where $\gamma(B)$ is a certain nonnegative function. Cheng and Kahlbacher [7] first showed that the general version of this problem is ordinary NP-hard, while Cheng, Gordon, and Kovalyov [6] later proved that it is strongly NP-hard. Polynomial algorithms for special cases when all weights are equal or all processing times are equal are presented by Cheng and Gordon [5] and Cheng, Gordon, and Kovalyov [6].

Motivation of our problem comes from the very large-scale integrated circuit manufacturing, which can be divided into four main stages: wafer fabrication, wafer probe, assembly, and final testing. Scheduling problems arising at the wafer fabrication stage have been considered by Dayhoff and Atherton [10], Bitran and Tirupati [3], Chen et al. [4], Glassey and Resende [13], and Wein [20]. Scheduling models which are typical for the assembly stage have been addressed in Dobson, Karmarkar, and Rummel [11], Baker [2], and in the papers indicated at the beginning of this section. The problem of scheduling semiconductor burn-in operations at the final testing stage has been studied by Lee, Uzsoy and Martin–Vega [16]. The scheduling problem studied in this paper arises at the assembly stage. In this stage, chips of various types are attached and placed on a circuit board by a pick-and-place machine. Each circuit board represents a job; upon completion, it is loaded onto a pallet. Intermittently, pallets are moved to the soldering machine and then to the test area. A set of circuit boards loaded on a pallet corresponds to a batch. The time to move a previous pallet and to install a new one corresponds to a set-up time.

For the assembly stage, an important performance criterion is to minimize the finished product inventories which are related to the total weighted earliness $\sum w_j E_j$. For succeeding operations, safety stocks of the product which justify the consideration of the mean product flow time criterion are important. Since the product flows on

pallets after assembly, the latter criterion is the mean batch delivery time $\sum D_b/B$. Our objective $\sum w_j E_j + \sum D_b/B$ is a linear combination of the above two criteria. By changing values for w_j, we can increase or decrease the impact of one of these criteria on the optimal schedule.

An analysis of our problem shows that the creation of small batches increases the total batch delivery time while the creation of large batches increases the job earliness within the batches. If only one of these strategies is applied to solve the problem, it is unlikely to find a solution with a reasonable objective value. This observation suggests that the batching decision is essential for our problem. As for the sequencing decision, we now show that we may restrict our search to schedules in which jobs in each batch are sequenced in *LWPT (longest weighted processing time)* order so that $p_{i_1}/w_{i_1} \geq p_{i_2}/w_{i_2} \geq \cdots \geq p_{i_k}/w_{i_k}$ if jobs $i_1, i_2, \ldots, i_k$ are sequenced in the batch in that order.

LEMMA 1.1. *In any optimal solution, jobs within each batch are sequenced in LWPT order.*

Proof. Consider an optimal solution and assume, without loss of generality, that jobs $i, i+1, \ldots, k$ are sequenced in a certain batch in that order. Assume that the statement of the lemma is not satisfied: $p_j/w_j < p_{j+1}/w_{j+1}$ for a certain $i \leq j \leq k-1$. It is easily checked that swapping j and $j+1$ decreases the total weighted job earliness by $w_j p_{j+1} - w_{j+1} p_j > 0$ and does not affect the batch delivery times. This contradicts the optimality of the original solution. □

Since jobs within each batch must be processed in LWPT order, the problem reduces to one of finding a number B of batches and a partition of the jobs into these batches.

The remainder of the paper is organized as follows. In the next section, we prove that the general problem is strongly NP-hard and that it remains strongly NP-hard when $s = 0$ and $B \leq U$ for a variable $U \geq 2$ or $B \geq U$ for any constant $U \geq 2$. We show that the problem is ordinary NP-hard even if $s = 0$ and $B \leq 2$. A dynamic programming algorithm is presented for the case when $B \leq U$. This algorithm runs in $O(nU^2(\sum_{j=1}^{n} p_j)^{U-1})$ time. In the following section, we derive $O(n^2)$ algorithms for the cases when all weights are equal or all processing times are equal. A heuristic approach for the general problem is then suggested. Computational results for the heuristics are also included. The paper concludes with some remarks and suggestions for further research.

2. NP-hardness proofs and dynamic programming. It is convenient to adopt the three-field notation of Graham et al. [14] to denote our family of problems. In the notation $1/\beta/\gamma$, the first field denotes the single machine environment. The second field, $\beta \subset \{\emptyset, B \leq U, B \geq U, B = U, s = 0, p_j = p\}$, indicates the batch constraint and job characteristics. Here, $B \leq U$ and $B \geq U$ indicate that the number of batches is bounded from above or from below, respectively, by a number U; $B = U$ denotes that the number of batches is equal to U; $s = 0$ denotes a zero set-up time; $p_j = p$ denotes that all processing times are equal to p. The third field, $\gamma \in \{\sum w_j E_j + \sum D_b/B, w\sum E_j + \sum D_b/B, \sum E_j + \sum D_b/B\}$, refers to the optimality criterion. Here, $w\sum E_j$ and $\sum E_j$ arise when $w_j = w$ and $w_j = 1$, respectively, for $j = 1, \ldots, n$. Our original problem is represented by $1//(\sum w_j E_j + \sum D_b/B)$.

In this section, we prove that the general problem, $1//(\sum w_j E_j + \sum D_b/B)$, is strongly NP-hard and the problem $1/B \leq U/(\sum w_j E_j + \sum D_b/B)$ is ordinary NP-hard for any constant $U \geq 2$. The complexities of the problems $1/B \geq U/(\sum w_j E_j + \sum D_b/B)$ and $1/s = 0, B \leq U/(\sum w_j E_j + \sum D_b/B)$ are easily established using the

same argument. Then, we present a dynamic programming algorithm for the problem $1/B \le U/(\sum w_j E_j + \sum D_b/B)$. We begin with the strong NP-hardness proof.

THEOREM 2.1. *The problem* $1//(\sum w_j E_j + \sum D_b/B)$ *is strongly* NP*-hard.*

Proof. We show that the decision version of our problem is strongly NP-complete by a transformation from the strongly NP-complete problem 3-PARTITION (Garey and Johnson, [12]): given positive integers $a_1, \ldots, a_{3U}$ and A such that $A/4 < a_j < A/2$ for $j = 1, \ldots, 3U$ and $\sum_{j=1}^{3U} a_j = AU$, is there a partition of the set $X = \{1, \ldots, 3U\}$ into U disjoint sets $X_1, \ldots, X_U$ such that for $1 \le b \le U$, $\sum_{j \in X_b} a_j = A$?

Define $c_j = 3Ua_j$ for $j = 1, \ldots, 3U$ and $C = \sum_{j=1}^{3U} c_j/U = 3UA$. Given any instance of 3-PARTITION, we construct an instance of our problem in which the set-up time

$$s = 2 \sum_{1 \le i < j \le 3U} c_i c_j - C^2U^2 + C^2U + 2CU + (C+1)(U+1)$$

and there are $4U$ jobs with $w_j = p_j = c_j$ for the *partition* jobs $j = 1, \ldots, 3U$ and $p_j = 1, w_j = y = (U+2)s/2$ for the *enforcer* jobs $j = 3U+1, \ldots, 4U$. We show that there exists a solution to 3-PARTITION if and only if there exists a solution to our problem with a value not exceeding y.

If X can be divided into U disjoint sets $X_1, \ldots, X_U$ such that $\sum_{j \in X_b} a_j = A$ for $b = 1, \ldots, U$, then we construct a schedule with U batches, where batch b consists of the partition jobs of the set X_b and one enforcer job scheduled last. Since $w_j = p_j$ for $j = 1, \ldots, 3U$, the order of the partition jobs in each batch does not affect the objective value F, which can be calculated as follows:

$$F = \sum_{j=1}^{4U} w_j E_j + \sum_{b=1}^{U} D_b/U,$$

where

$$\begin{aligned} \sum_{j=1}^{4U} w_j E_j &= \sum_{b=1}^{U} \left(\sum_{i<j,\ i,j \in X_b} c_i c_j + \sum_{j \in X_b} c_j \right) \\ &= \sum_{1 \le i < j \le 3U} c_i c_j - \sum_{1 \le b < e \le U} \left(\sum_{j \in X_b} c_j \right) \left(\sum_{j \in X_e} c_j \right) + \sum_{j=1}^{3U} c_j \end{aligned}$$

and

$$\sum_{b=1}^{U} D_b/U = (U+1)s/2 + \sum_{b=1}^{U} (U+1-b) \left(\sum_{j \in X_b} c_j + 1 \right) /U.$$

Since

$$(CU)^2 = \left(\sum_{j=1}^{3U} c_j \right)^2 = \sum_{b=1}^{U} \left(\sum_{j \in X_b} c_j \right)^2 + 2 \sum_{1 \le b < e \le U} \left(\sum_{j \in X_b} c_j \right) \left(\sum_{j \in X_e} c_j \right),$$

we have

$$F = (U+1)s/2 + \sum_{1 \le i < j \le 3U} c_i c_j - C^2U^2/2 \tag{1}$$

$$+\sum_{b=1}^{U}\left(\sum_{j\in X_b} c_j\right)^2/2 + CU + \sum_{b=1}^{U}(U+1-b)\left(\sum_{j\in X_b} c_j + 1\right)/U.$$

Setting $\sum_{j\in X_b} c_j = 3U\sum_{j\in X_b} a_j = 3UA = C$ for $b = 1,\dots,U$, we get $F = y$.

Assume that there is a solution to the problem $1//(\sum w_jE_j + \sum D_b/B)$ with a value $F \le y$. It is apparent that there cannot be more than U batches, since then there are at least $U+1$ set-ups and we have $F > (U+2)s/2 = y$. If there are less than U batches, then at least one batch includes at least two enforcer jobs. In this case, at least one enforcer job is not scheduled last in one of the batches. Since the weight of each enforcer job is equal to y, we again get $F > y$. Thus, there are exactly U batches and each batch includes exactly one enforcer job which is scheduled last. Denote the set of the partition jobs in batch b by X_b. Then the value F of our solution can be calculated as shown in (1). By simplifying $F \le y$, we obtain

$$\sum_{b=1}^{U}\left(\sum_{j\in X_b} c_j\right)^2/2 + \sum_{b=1}^{U}(U+1-b)\sum_{j\in X_b} c_j/U \le C^2U/2 + C(U+1)/2.$$

The latter inequality can be represented as follows:

$$\sum_{b=1}^{U}\left(\sum_{j\in X_b} c_j - C\right)\left(\sum_{j\in X_b} c_j + C\right) + 2(U+1-b)\left(\sum_{j\in X_b} c_j - C\right)/U \le 0.$$

Define $\delta_b = \sum_{j\in X_b} c_j - C$ for $b = 1,\dots,U$. Clearly, $\sum_{b=1}^{U}\delta_b = 0$. We have $\sum_{b=1}^{U}(\delta_b^2 + 2(U+1-b)\delta_b/U) \le 0$ or, equivalently,

$$\sum_{b=1}^{U}(\delta_b + (U+1-b)/U)^2 \le \sum_{b=1}^{U}(U+1-b)^2/U^2 \le U.$$

Thus, $\max_{1\le b\le U}|\delta_b| \le U^{1/2} + 1 \le 2U$. The latter relations provide

$$C - 2U \le \sum_{j\in X_b} c_j \le C + 2U \text{ for } b = 1,\dots,U.$$

Substituting $3Ua_j$ for c_j and $3UA$ for C, we deduce that

$$A - 2/3 \le \sum_{j\in X_b} a_j \le A + 2/3 \text{ for } b = 1,\dots,U.$$

These inequalities and the integrality of a_j yield $\sum_{j\in X_b} a_j = A$ for $b = 1,\dots,U$, as required. □

Similar reductions show that the problem $1/B \ge U/(\sum w_jE_j + \sum D_b/B)$ is strongly NP-hard if U is a constant and the problems $1/B \le U/(\sum w_jE_j + \sum D_b/B)$ and $1/s = 0, B \le U/(\sum w_jE_j + \sum D_b/B)$ are strongly NP-hard if U is a given variable. For the former problem, the only modification of the above proof is that the (variable) number U of sets in 3-PARTITION is substituted by B, since B is a variable number of batches now and U is a constant. For the second problem, the proof is completely the same. For the third problem with zero set-up time, we should set

$$y = \sum_{1\le i<j\le 3U} c_ic_j - C^2U^2/2 + C^2U/2 + CU + (C+1)(U+1)/2$$

in order to show that $B \geq U$. In Theorem 2.1, a nonzero set-up time has been used to show only that $B \leq U$. Therefore, if $B \leq U$ is given a priori, we can set $s = 0$ in our proof.

THEOREM 2.2. *The problem* $1/B \leq U/(\sum w_j E_j + \sum D_b/B)$ *is ordinary NP-hard for any constant* $U \geq 2$.

Proof. Our proof is similar to the one in the previous theorem. A transformation from the NP-complete problem PARTITION (Garey and Johnson [12]) is used. □

Besides the above results, we have also proved that the problem with a zero set-up time, $1/s = 0, B \leq U/(\sum w_j E_j + \sum D_b/B)$, is ordinary NP-hard for any constant $U \geq 2$.

It should be noted that all of the above complexity results remain valid if the total set-up time is included in the objective function instead of the mean batch delivery time.

We now present a dynamic programming algorithm DP for the problem $1/B \leq U/(\sum w_j E_j + \sum D_b/B)$. This algorithm is based on Lemma 1.1. Assume that jobs are numbered in *SWPT (shortest weighted processing time)* order so that $p_1/w_1 \leq \cdots \leq p_n/w_n$. In Algorithm DP, jobs are considered in natural order $1, \ldots, n$. Job j is either assigned to the beginning of one of the current batches or it starts a new batch. Thus, jobs within each batch are sequenced in LWPT order. We recursively compute the value of $F_j(P_1, \ldots, P_B)$, which represents the minimal objective value subject to j jobs being scheduled in B batches, and the total processing time of the jobs in batch b is equal to P_b for $b = 1, \ldots, B$. Note that the set-up time is not included in P_b.

Set $T_j = \sum_{i=1}^{j} p_i$ for $j = 1, \ldots, n$. A formal description of Algorithm DP is as follows.

ALGORITHM DP.

Step 1 (Initialization) Number jobs in SWPT order so that $p_1/w_1 \leq \cdots \leq p_n/w_n$. Set $F_j(P_1, \ldots, P_B) = \infty$ for $j = 0, 1, \ldots, n, 0 \leq P_b \leq T_n, b = 1, \ldots, B$ and $B = 1, \ldots, U$. Set $F_0(0) = 0$. Set $j = 1$.

Step 2 (Recursion) Compute the following for all tuples $(P_1, \ldots, P_B)$ such that $p_j \leq P_b \leq T_j, b = 1, \ldots, B, B = 1, \ldots, \min\{j, U\}$.

$$F_j(P_1, \ldots, P_B) = \min_{1 \leq b \leq B} \min \tag{2}$$

$$\begin{cases} F_{j-1}(P_1, \ldots, P_{b-1}, P_b - p_j, P_{b+1}, \ldots, P_B) & \\ +w_j(P_b - p_j) + p_j(B - b + 1)/B & \text{if } P_b > p_j, \\ F_{j-1}(P_1, \ldots, P_{b-1}, P_{b+1}, \ldots, P_B) + s/2 + p_j(B - b + 1)/B & \\ +(\sum_{k=1, k \neq b}^{B} (k-1)P_k - \sum_{k=b+1}^{B} (B - k + 1)P_k)/(B^2 - B) & \text{if } P_b = p_j, \\ \infty & \text{if } P_b < p_j. \end{cases}$$

The three quantities in the right-hand side of equation (2) represent the three possible scheduling choices for job j with respect to batch b:

1. Add job j to the beginning of the existing batch b.
2. Form a new batch b consisting of the sole job j.
3. Do not assign job j to batch b.

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

Step 3 (Optimal solution) Define optimal solution value

$$F^* = \min\{F_n(P_1, \ldots, P_B) | 0 \leq P_b \leq T_n, B = 1, \ldots, U\}$$

and use backtracking to find the corresponding optimal solution.

THEOREM 2.3. *Algorithm DP solves the problem* $1/B \le U/(\sum w_j E_j + \sum D_b/B)$ *in* $O(nU^2(\sum_{j=1}^{n} p_j)^{U-1})$ *time.*

Proof. Due to Lemma 1.1, there is always an optimal schedule with jobs arranged in LWPT order within each batch. Therefore, at each stage of the algorithm we need only decide whether to include job j in batch b and, if so, whether batch b is new or not. It is now easy to apply the general dynamic programming justification for scheduling problems (Rothkopf, [18]; Lawler and Moore, [15]) to show that DP solves the problem $1/B \le U/(\sum w_j E_j + \sum D_b/B)$. The time complexity of this algorithm can be established as follows.

In each iteration of Step 2, only $B-1$ of the values $P_1, \ldots, P_B$ are independent, since $P_1 + \cdots + P_B = T_j$. Hence, in iteration j of Step 2, the number of different tuples $(P_1, \ldots, P_B)$ for $B = 1, \ldots, U$ is at most UT_j^{U-1}. For each tuple $(P_1, \ldots, P_B)$, the right-hand side of equation (2) can be calculated in $O(B)$ time. Thus, Step 2 requires $O(nU^2T_n^{U-1})$ time, which is the overall time complexity of Algorithm DP as well. □

Theorem 3 shows that the problem $1/B \le U/(\sum w_j E_j + \sum D_b/B)$ is not strongly NP-hard for any constant $U \ge 2$.

3. Polynomially solvable cases. In this section, we present polynomial time algorithms for two special cases of our problem; namely, all weights are equal and all processing times are equal. We first show that the problem with equal weights, $1//(w\sum E_j + \sum D_b/B)$, can be solved in $O(n^2)$ time.

Consider a certain solution to the problem $1//(w\sum E_j + \sum D_b/B)$. To facilitate discussion, we represent it as a pair (U, x), where U is the number of batches, x is a sequence of the batches $1, 2, \ldots, U$, and each batch b includes jobs $i_1^b, i_2^b, \ldots, i_{j(b)}^b$ in that order. The total earliness of the jobs in batch b can be calculated as follows:

$$\sum_{k\in b} E_k = (j(b)-1)p_{i_{j(b)}^b} + (j(b)-2)p_{i_{j(b)-1}^b} + \cdots + p_{i_2^b} = \sum_{k=1}^{j(b)} (k-1)p_{i_k^b}.$$

For all n jobs, we have $\sum_{j=1}^{n} E_j = \sum_{b=1}^{U}\sum_{k=1}^{j(b)} (k-1)p_{i_k^b}$.

Recall the definition of the total processing time of jobs in batch b: $P_b = \sum_{k=1}^{j(b)} p_{i_k^b}$. For the mean batch delivery time, we have

$$\sum_{b=1}^{U} D_b/U = s(U+1)/2 + \sum_{b=1}^{U}(U+1-b)P_b/U = s(U+1)/2 + \sum_{b=1}^{U}\sum_{k=1}^{j(b)} p_{i_k^b}(U+1-b)/U.$$

Thus, the problem $1//(w\sum E_j + \sum D_b/B)$ reduces to one of minimizing $s(U+1)/2 + F(U,x)$, where

$$F(U,x) = \sum_{b=1}^{U}\sum_{k=1}^{j(b)}((U+1-b)/U + w(k-1))p_{i_k^b}.$$

Let (U^*, x^*) be an optimal solution to this problem and let $x^{(U)}$ be an optimal solution to the problem of minimizing $F(U,x)$. We have

$$s(U^*+1)/2 + F(U^*, x^*) = \min\{s(U+1)/2 + F(U, x^{(U)}) | U = 1, \ldots, n\}.$$

TABLE 1
(U).

$b\backslash k$	1	2	3	...	n
1	1	$1+w$	$1+2w$	...	$1+(n-1)w$
2	$(U-1)/U$	$(U-1)/U+w$	$(U-1)/U+2w$	...	$(U-1)/U+(n-1)w$
.	.	.	.	...	.
.	.	.	.	...	.
.	.	.	.	...	.
$U-2$	$3/U$	$3/U+w$	$3/U+2w$	...	$3/U+(n-1)w$
$U-1$	$2/U$	$2/U+w$	$2/U+2w$	...	$2/U+(n-1)w$
U	$1/U$	$1/U+w$	$1/U+2w$	...	$1/U+(n-1)w$

Consider the problem of minimizing $F(U,x)$. In this problem, $F(U,x)$ is a weighted sum of n number of p_j values where the weights are presented in Table $1(U)$.

In $F(U,x)$, each element in Table $1(U)$ may be used at most once in order to satisfy the restriction that each job should be assigned to exactly one batch, and at least one element from each row of this table should be used in order to satisfy the restriction that there are exactly U batches. To find an optimal solution $x^{(U)}$, it is obvious that we have to choose the smallest elements satisfying the above conditions, i.e., all U elements from the first column and the $n-U$ smallest elements from the remaining part of the table, and then match the smallest chosen elements with the largest processing requirements p_j. The procedure of choosing the r smallest elements $t_{bk}^U = (U+1-b)/U+w(k-1)$ can be implemented in $O(r)$ time. If p_j is matched with an element t_{bk}^U, then job j is sequenced kth in batch b. Thus, $x^{(U)}$ can be found in $O(n)$ time and (U^*,x^*) can be found in $O(n^2)$ time. Therefore, we have the following theorem.

THEOREM 3.1. *The problem* $1//(w\sum E_j+\sum D_b/B)$ *is solved in* $O(n^2)$ *time.*

We now study the problem with equal processing times, $1/p_j = p/(\sum w_jE_j + \sum D_b/B)$. For this problem, we first rearrange the jobs such that $w_1 \geq w_2 \geq \cdots \geq w_n$. Assume that there are exactly U batches, $B_1, B_2, \ldots,$ and $B_U, 1 \leq U \leq n$. Because the jobs have the same processing time, there is an optimal solution in which $|B_i| \leq |B_j|$ if batch B_i precedes batch B_j. With this observation, we devise the following algorithm for a fixed U:

Step 1: Assign jobs $1, 2, \ldots,$ and U to batch $B_U, B_{U-1}, \ldots,$ and B_1 as a partial schedule.

Step 2: Loop for job j over $U+1, U+2, \ldots,$ and n: For each partial schedule, find the last batch B_r satisfying $|B_r| < |B_U|$, and then assign job j in accordance with the following cases.

Case 1. There is no such a batch, i.e., all batches have the same number of jobs: Assign job j as the first job of batch B_U.

Case 2. $|B_r| = |B_U| - 1$:

- Enhance the partial schedule by assigning job j as the first job of batch B_U. If $j = n$, output the schedule as a candidate solution.
- Enhance the partial schedule by assigning job j as the first job of batch B_r. If $j = n$, output the schedule as a candidate solution.

Case 3. $|B_r| = |B_U| - 2$: Assign jobs $j, j+1, \ldots, n$ to batch B_U in the order of non-decreasing weights. Output this schedule as a candidate solution.

Step 3: Amongst the candidate solutions, output one of those with the mini-

mum cost.

To establish the correctness of the proposed algorithm, we first consider an important property. Let s_i denote the number of successors of job i in the batch containing job i. By a simple interchange argument, we readily see that there is an optimal solution where, for any two jobs i and j, if $w_i \geq w_j$, then $s_i \leq s_j$. Assume that schedule S is an optimal solution satisfying this property. We show that S can be transformed into a candidate solution delivered by the algorithm without increasing the costs. Suppose that the partial schedule for jobs 1, 2, $\ldots, j-1$ in S is the same as some partial schedule proposed by the algorithm. Now, consider the assignment of job j. In Case 1, it is evident that job j can be assigned to batch B_U to minimize the delivery penalty. In analyzing Case 2, we know, by the property just stated, that job j should be in some batch B_p with either $|B_p| = |B_r|$ or $|B_p| = |B_U|$. Therefore, if job j is not in one of the two specified positions, i.e., one in B_r and the other in B_U, we can swap the job positions without increasing the costs. As for Case 3, we note that the first job in batch B_U must be job $j-1$. Suppose that job k, $j \leq k \leq n$, is assigned to batch B_p, $p \neq U$. We can assume, without loss of generality, that $p = U - 1$. By swapping the positions of jobs $j-1$ and k, the cost will not increase. Furthermore, the derived solution has a partial schedule for jobs 1, 2, ..., and $j-1$ that is the same as a partial schedule proposed by the algorithm. Continuing the above interchange arguments, we finally obtain a schedule that is the same as a candidate solution proposed by the algorithm.

Now, we turn to the issue of the time complexity of the algorithm. Because the branching from a partial schedule terminates when the condition in Case 2 is satisfied, the total number of candidate solutions is bounded by $O(n)$. By performing a simple preprocessing step to compute the cumulative job weights, the objective values of all candidate solutions can be calculated in $O(n)$ time. Noting that there are n possible values for the variable U, we conclude with the following theorem.

THEOREM 3.2. *The problem* $1/p_j = p/(\sum w_j E_j + \sum D_b/B)$ *is solved in* $O(n^2)$ *time.*

Theorems 3.1 and 3.2 resolve the computational complexities of all special cases of our problem in which either all weights or all processing times are equal.

4. Heuristics. In this section, we present a heuristic approach to solving the general problem $1//(\sum w_j E_j + \sum D_b/B)$.

We first describe a list-scheduling algorithm for the problem with a fixed number of batches, $1/B = U/(\sum w_j E_j + \sum D_b/B)$.

Let $LIST$ be a sequence of jobs and let $RULE$ be a rule of assigning a job from $LIST$ to a batch. In a list-scheduling algorithm, jobs are considered in an order determined by $LIST$. Each successive job is scheduled according to $RULE$. We consider $LIST \in \{LWPT, SWPT, SPT, LPT, SW, LW\}$, where the jobs are numbered so that

$$\begin{aligned} p_1/w_1 \geq p_2/w_2 \geq \cdots \geq p_n/w_n &\text{ in } LWPT,\\ p_1/w_1 \leq p_2/w_2 \leq \cdots \leq p_n/w_n &\text{ in } SWPT,\\ p_1 \geq p_2 \geq \cdots \geq p_n &\text{ in } LPT,\\ p_1 \leq p_2 \leq \cdots \leq p_n &\text{ in } SPT,\\ w_1 \geq w_2 \geq \cdots \geq w_n &\text{ in } LW,\\ w_1 \leq w_2 \leq \cdots \leq w_n &\text{ in } SW. \end{aligned}$$

We use two types of $RULE$: $RULE1$ and $RULE2$. According to $RULE1$, a job is assigned to the end of the earliest batch b with the minimal total processing time

TABLE 2
Computational results for $s = 500$ and $n = 100$.

$RULE$, $LIST$	$p' = 100$ $w' = 10$	$p' = 100$ $w' = 1$	$p' = 10$ $w' = 10$	$p' = 10$ $w' = 1$
$RULE1$, $LWPT$	30399.40	22561.75	17931.31*	7210.09
$RULE1$, $SWPT$	30194.13	23130.77	18563.15	7536.56
$RULE1$, LPT	30436.84	22486.16*	18000.67	7251.58
$RULE1$, SPT	30157.10*	23021.43	18502.26	7496.78
$RULE1$, LW	30281.10	23105.58	18487.39	7495.54
$RULE1$, SW	30311.46	22664.68	18011.39	7251.39
$RULE2$, $LWPT$	30399.40	22648.63	17934.77	7204.55*
$RULE2$, $SWPT$	30194.13	23116.94	18557.74	7530.62
$RULE2$, LPT	30436.84	22641.48	18009.04	7252.37
$RULE2$, SPT	30157.10*	23021.43	18502.26	7489.33
$RULE2$, LW	30281.10	23089.33	18480.68	7487.63
$RULE2$, SW	30311.46	22669.87	18011.33	7258.79
$EQUAL\_W\_AVG$	30157.10	22423.06	18009.20	7245.77
$EQUAL\_W\_MIN$	30157.10	21824.43	17611.56	6958.70

$P_b = \sum_{j \in b} p_j$. According to $RULE2$, a job is assigned to the end of the earliest batch b with the minimal total weighted earliness $F_b = \sum_{j \in b} w_j E_j$. Values P_b or F_b, $b = 1, \ldots, U$, are stored in a heap. The heap can be initiated in $O(U \log U)$ time and updated in $O(\log U)$ time. Therefore, our list-scheduling algorithm can be implemented in $O(U \log U)$ time.

We apply the list-scheduling algorithm for all possible combinations of $LIST$ and $RULE$. Let $LIST(U)$ be an algorithm which performs all twelve combinations and chooses the best constructed schedule $S^{(U)}$ with the value $F(S^{(U)})$ with respect to the problem $1/B = U/(\sum w_j E_j + \sum D_b/B)$. Our final algorithm H is to apply $LIST(U)$ for $U = 1, \ldots, n$, and select the best schedule S^H with the value

$$F(S^H) = \min\{F(S^{(U)}) | U = 1, \ldots, n\}.$$

The complexity of the algorithm H is $O(n^2 \log n)$.

In the following, we conduct computational experiments to test the proposed heuristic algorithm. Because of the intractability of the general problem, it is hard to derive exact solutions. Therefore, we make use of the polynomial algorithm designed for the equal-weight case in the previous section.

In the experiments, four parameters (namely, set-up time (s), number of jobs (n), job length (p'), and job weight (w')), are taken into consideration, and two possible values for each parameter will be set. There are a total of 16 combinations from

$$\{s = 50 \text{ or } 500\} \times \{n = 20 \text{ or } 100\} \times \{p' = 10 \text{ or } 100\} \times \{w' = 1 \text{ or } 10\}.$$

Note the actual implication of p' and w'. For a given p' (w'), all processing times (weights), p_i (w_i), are randomly drawn from the uniform distribution $[p' - 0.1p', p' + 0.1p']$ ($[w' - 0.1w', w' + 0.1w']$). For example, $p' = 100$ means that all the job lengths, p_i, are randomly drawn from the uniform distribution $[100 - 10, 100 + 10]$. This indicates a ten percent variation in processing times. Such an assumption is reasonable in real-world applications because the processing times of jobs on a production line often exhibit some degree of variation.

The platform of our experiments is a personal computer that contains an Intel Pentium 75 processor and runs MS-DOS 6.2. All the programs are coded in Turbo Pascal 6.0. Tables 2–5 display the numerical results. For each parameter combination, 12 objective values are listed for all possible $RULE \times LIST$ pairs. Besides, we

TABLE 3
Numerical results for $s = 500$ and $n = 20$.

RULE, LIST	$p' = 100$ $w' = 10$	$p' = 100$ $w' = 1$	$p' = 10$ $w' = 10$	$p' = 10$ $w' = 1$
*RULE*1, *LWPT*	6304.15	4807.89*	3824.19*	1671.08
*RULE*1, *SWPT*	6259.10	4922.49	3920.09	1715.72
*RULE*1, *LPT*	6308.85	4821.18	3853.32	1679.34
*RULE*1, *SPT*	6254.40*	4907.74	3871.92	1710.37
*RULE*1, *LW*	6286.15	4916.87	3906.86	1704.23
*RULE*1, *SW*	6278.30	4818.16	3836.68	1681.63
*RULE*2, *LWPT*	6304.15	4813.10	3826.13	1668.98*
*RULE*2, *SWPT*	6259.10	4918.23	3920.24	1715.55
*RULE*2, *LPT*	6308.85	4834.14	3857.70	1679.16
*RULE*2, *SPT*	6254.40*	4907.74	3871.92	1700.69
*RULE*2, *LW*	6286.15	4910.27	3907.97	1709.12
*RULE*2, *SW*	6278.30	4819.40	3836.59	1682.32
EQUAL_W_AVG	6254.40	4782.91	3837.49	1672.71
EQUAL_W_MIN	6254.40	4659.33	3749.47	1634.29

TABLE 4
Numerical results for $s = 50$ and $n = 100$.

RULE, LIST	$p' = 100$ $w' = 10$	$p' = 100$ $w' = 1$	$p' = 10$ $w' = 10$	$p' = 10$ $w' = 1$
*RULE*1, *LWPT*	7706.68	7609.11	3040.94	2244.61
*RULE*1, *SWPT*	7462.55	7374.40	3023.33	2310.07
*RULE*1, *LPT*	7742.37	7646.22	3046.20	2240.97*
*RULE*1, *SPT*	7426.82*	7337.13*	3018.05*	2298.52
*RULE*1, *LW*	7574.22	7484.05	3033.05	2302.99
*RULE*1, *SW*	7587.10	7500.85	3031.20	2251.14
*RULE*2, *LWPT*	7706.68	7609.11	3040.94	2250.04
*RULE*2, *SWPT*	7462.55	7374.40	3023.33	2307.61
*RULE*2, *LPT*	7742.37	7646.22	3046.20	2253.24
*RULE*2, *SPT*	7426.82*	7337.13*	3018.05*	2298.52
*RULE*2, *LW*	7574.22	7484.05	3033.05	2302.79
*RULE*2, *SW*	7587.10	7500.85	3031.20	2253.51
EQUAL_W_AVG	7426.82	7337.13	3018.05	2230.09
EQUAL_W_MIN	7426.82	7337.13	3018.05	2169.22

TABLE 5
Numerical results for $s = 50$ and $n = 20$.

RULE, LIST	$p' = 100$ $w' = 10$	$p' = 100$ $w' = 1$	$p' = 10$ $w' = 10$	$p' = 10$ $w' = 1$
*RULE*1, *LWPT*	1610.45	1569.60	630.01	473.78*
*RULE*1, *SWPT*	1571.05	1544.70	626.09	487.67
*RULE*1, *LPT*	1619.95	1583.55	630.77	477.60
*RULE*1, *SPT*	1561.55*	1530.75*	625.34*	486.05
*RULE*1, *LW*	1594.45	1562.35	628.27	486.21
*RULE*1, *SW*	1589.95	1550.20	627.94	478.25
*RULE*2, *LWPT*	1610.45	1569.60	630.01	476.33
*RULE*2, *SWPT*	1571.05	1544.70	626.09	487.09
*RULE*2, *LPT*	1619.95	1583.55	630.77	475.46
*RULE*2, *SPT*	1561.55*	1530.75*	625.34*	486.05
*RULE*2, *LW*	1594.45	1562.35	628.27	487.23
*RULE*2, *SW*	1589.95	1550.20	627.94	478.58
EQUAL_W_AVG	1561.55	1530.75	625.34	472.97
EQUAL_W_MIN	1561.55	1530.75	625.34	463.17

have two values that are categorized as *EQUAL_W_AVG* and *EQUAL_W_MIN*. The *EQUAL_W_AVG* and *EQUAL_W_MIN* values are obtained by applying the $O(n^2)$ algorithm for $w = \sum_{i=1}^{n} w_i$ and $w = \min_{i=1}^{n}\{w_i\}$, respectively. For each column of the table, the entries with an asterisk denote the objective value output by the heuristic algorithm, H.

It is not hard to see that the *EQUAL_W_MIN* values serve as lower bounds for H values. An analysis of Tables 2–5 shows that the results delivered by algorithm H are quite close to that delivered by the polynomial algorithm. The largest relative percentage deviation between H and *EQUAL_W_MIN* occurs when $n = 100, s = 500, p' = 10,$ and $w' = 10$, and the value is around 3.61 percent, which is much smaller than the assumed 10 percent deviation among the data instances. In other words, the effectiveness of algorithm H in producing near-optimal solutions is convincingly evident. Detailed observations further show some interesting properties:

1. When the set-up time, s, is relatively small, the impact of the weight, w_i, is alleviated. In Tables 4 and 5, the cases in which *EQUAL_W_AVG* = *EQUAL_W_MIN* indicate that all batches contain exactly one job and that the effect of the weights is null.

2. For most of the data instances (or columns), the minimal objective values for H occur when using $RULE1$.

3. For a specific list-scheduling policy, the difference between the objective values for $RULE1$ and $RULE2$ is small.

Finally, the above numerical results show no preference to any specific list-scheduling policy. Therefore, we do not expect to obtain satisfactory solutions by simply applying a specific combination of $RULE \times LIST$. Another supporting argument for employing algorithm H is that the running sessions take less than three seconds.

5. Conclusions. The problems $1//(\sum w_j E_j + \sum D_b/B)$, $1/B \geq U/(\sum w_j E_j + \sum D_b/B)$, $1/B \leq U/(\sum w_j E_j + \sum D_b/B)$, and $1/s = 0, B \leq U/(\sum w_j E_j + \sum D_b/B)$ have been shown to be strongly NP-hard. The problems $1/B \leq 2/(\sum w_j E_j + \sum D_b/B)$ and $1/s = 0, B \leq 2/(\sum w_j E_j + \sum D_b/B)$ have been proved to be ordinary NP-hard. Algorithms with $O(n^2)$ running times have been derived for the cases when all weights are equal or all processing times are equal. Thus, the computational complexities of all special cases of the problem in which all weights or all processing times are equal have been resolved. A dynamic programming algorithm has been presented for the case with a limited number of batches. A heuristic approach has been suggested for the general problem. The numerical results reveal the practical significance of this algorithm in producing near-optimal solutions quickly.

An interesting problem for further research is one for which there is a natural restriction that each batch can include no more than a given number of jobs. The complexity aspects of this problem are yet to be studied. However, our dynamic programming algorithm DP and heuristic algorithm H can easily be modified to solve this problem. These algorithms can also be adopted for the problem in which, besides the job weights, the batch weights are given and the total weighted batch delivery time is included in the objective function.

Acknowledgments. The authors wish to thank the referees for their constructive comments and helpful discussions of the presented model.

REFERENCES

[1] S. Albers and P. Brucker (1993), *The complexity of one-machine batching problems*, Discrete Appl. Math., 47, pp. 87–107.

[2] K. R. Baker (1988), *Scheduling the production of components at a common facility*, IIE Trans., 20, pp. 32–35.

[3] G. R. Bitran and D. Tirupati (1988), *Planning and scheduling for epitaxial wafer production*, Oper. Res., 36, pp. 34–49.

[4] H. Chen, J.M. Harrison, A. Mandelbaum, A. Van Ackere, and L. M. Wein (1988), *Empirical evaluation of a queueing network model for semiconductor wafer fabrication*, Oper. Res., 36, pp. 202–215.

[5] T. C. E. Cheng and V. S. Gordon (1994), *Batch delivery scheduling on a single machine*, J. Oper. Res. Soc., 45, pp. 1211–1215.

[6] T. C. E. Cheng, V. S. Gordon, and M. Y. Kovalyov (1996), *Single machine scheduling with batch deliveries*, European J. Oper. Res., 94, pp. 277–283.

[7] T. C. E. Cheng and H. G. Kahlbacher (1993), *Scheduling with delivery and earliness penalties*, Asia-Pacific J. Oper. Res., 10, pp. 145–152.

[8] E. G. Coffman, Jr., M. Yannakakis, M. J. Magazine, and C. Santos (1990), *Batch sizing and job sequencing on a single machine*, Ann. Oper. Res., 26, pp. 135–147.

[9] E. G. Coffman, A. Nozari, and M. Yannakakis (1989), *Optimal scheduling of products with two subassemblies on a single machine*, Oper. Res., 37, pp. 426–436.

[10] J. E. Dayhoff and R. W. Atherton (1987), *A model for wafer fabrication dynamics in integrated circuit manufacturing*, IEEE Trans. Systems Man Cybernet., 17, pp. 91–100.

[11] G. Dobson, U. S. Karmarkar, and J. L. Rummel (1987), *Batching to minimize flow times on one machine*, Management Sci., 33, pp. 784–799.

[12] M. R. Garey and D. S. Johnson (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, CA.

[13] C. R. Glassey and M. G. C. Resende (1988), *Closed-loop job release control for VLSI circuit manufacturing*, IEEE Trans. Semiconductor Manufacturing, 1, pp. 36–46.

[14] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan (1979), *Optimization and approximation in deterministic sequencing and scheduling*, Ann. Discrete Math., 5, pp. 287–326.

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

[16] C.-Y. Lee, R. Uzsoy, and L. A. Martin–Vega (1992), *Efficient algorithms for scheduling semiconductor burn-in operations*, Oper. Res., 40, pp. 764–775.

[17] D. Nadeff and C. Santos (1988), *One-pass batching algorithms for the one machine problem*, Discrete Appl. Math., 21, pp. 133–145.

[18] M. H. Rothkopf (1966), *Scheduling independent tasks on parallel processors*, Management Sci., 12, pp. 437–447.

[19] D. Shallcross (1992), *A polynomial algorithm for a one machine batching problem*, Oper. Res. Lett., 11, pp. 213–218.

[20] L. M. Wein (1988), *Scheduling semiconductor wafer fabrication*, IEEE Trans. Semiconductor Manufacturing, 1, pp. 115–129.