Approximation Schemes for Minimizing Total (Weighted) Completion Time with Release Dates on a Batch Machine

Zhaohui Liu[1, 2] and T.C. Edwin Cheng[2,*]

[1]Department of Mathematics, East China University of Science and Technology

Shanghai 200237, People's Republic of China

[2]Department of Logistics, The Hong Kong Polytechnic University

Kowloon, Hong Kong SAR, People's Republic of China

June 3, 2005

Abstract

A batch machine is a machine that can process a number of jobs simultaneously as a batch, and the processing time of a batch is equal to the longest processing time of the jobs assigned to it. In this paper we present a polynomial time approximation scheme (PTAS) for scheduling a batch machine to minimize the total completion time with job release dates. Also, we present a fully polynomial time approximation scheme (FPTAS) for scheduling an unbounded batch machine, which can process an arbitrary number of jobs simultaneously, to minimize the total weighted completion time with job release dates.

Keywords: Scheduling; Batch Processing; Approximation Scheme

*Corresponding author.

1 Introduction

This research is concerned with the so-called burn-in model for scheduling wafer production in semiconductor manufacturing [9]. Wafers (i.e., jobs) are produced by a batch machine or batch processing machine that can process a number of jobs simultaneously as a batch. Once the processing of a batch is initiated, it cannot be interrupted, nor can other jobs be introduced into the batch. The processing time of a batch is equal to the longest processing time of the jobs assigned to it. Then all the jobs processed in a batch have the same start time and the same completion time. In this paper we study the problem of scheduling a set of jobs $\mathcal{J} = \{1, 2, \ldots, n\}$ on a batch machine that can process up to c jobs simultaneously. Each job j ($j = 1, 2, \ldots, n$) is associated with a processing time p_j and a release date r_j, before which the job cannot be scheduled. The scheduling objective is to minimize the total completion time $\sum_{j=1}^{n} C_j$, where C_j is the completion time of job j.

The problem is strongly NP-hard even for the case of $c = 1$, but it can be solved in $O(n^{c(c-1)})$ time if $c \geq 2$ and all release dates are equal [2]. If $c = 1$ and all release dates are equal, it can be solved in $O(n \log n)$ time by the shortest processing time (SPT) rule. If c is variable and all release dates are equal, the complexity of the problem is still open, but Hochbaum and Landy [8] presented a 2-approximation algorithm, which was later improved by Cai et al. [3] to a polynomial time approximation scheme (PTAS). For arbitrary release dates, Chen et al. [4] gave a $(4 + \epsilon)$-approximation algorithm for any $\epsilon > 0$. Their algorithm is on-line and applicable even to the total weighted completion time objective.

In this paper we present a PTAS for the batch machine scheduling problem with arbitrary job release dates, which improves on the result of [3]. Unlike the work of [3] that depends heavily on the structural properties developed in [8], our method follows closely the seminal work of Afrati et al. [1]. We use the same basic tools as in [1], namely geometric rounding, time stretching, small and large jobs partitioning and dynamic programming, but the characteristics of the batch machine make the analysis tricky. Our result also improves on the recent work of Deng et al. [5], who consider the case where c is fixed.

A less restrictive version of the above problem is the unbounded version in which $c = +\infty$. For this case, Deng et al.[6] presented a PTAS. In this paper we give a fully polynomial time approximation scheme (FPTAS) for the unbounded batch machine

scheduling problem with a more general objective, i.e., the total weighted completion time $\sum_{j=1}^{n} w_j C_j$, where w_j is the weight of job j. Our FPTAS is based upon the pseudopolynomial dynamic programming algorithm developed in Liu et al. [10]. Also, we note that the unbounded problem with the total weighted completion time objective has been proved NP-hard in Deng and Zhang [7], but the complexity of the unbounded problem with the total completion time objective is open.

The remainder of this paper is divided into two sections. In Section 2, we present the PTAS for the total completion time problem on a bounded batch machine. In Section 3, we present the FPTAS for the total weighted completion time problem on an unbounded batch machine.

2 The total completion time problem on a bounded batch machine

In this section we design a PTAS for the problem of minimizing total completion time with release dates on a bounded batch machine.

2.1 The framework of our approach

Let $\frac{1}{100} \geq \epsilon > 0$ and $1/\epsilon$ be integral. We partition the time interval $(0, +\infty)$ into disjoint intervals of the form $I_x = [R_x, R_{x+1})$, where $R_x = (1+\epsilon)^x$ and $x \in \mathcal{Z} = \{0, \pm 1, \pm 2, \cdots\}$. I_x will also be used to refer to the length of the interval $[R_x, R_{x+1})$, thus $I_x = R_{x+1} - R_x = \epsilon R_x$.

As in Afrati et al. [1], we use a combination of several general techniques. The first is geometric rounding that rounds up all processing times and release dates to integer powers of $1+\epsilon$ to create a well-structured data set. The second is time stretching that stretches each interval I_x by a factor of $1+\epsilon$ to create ϵI_x units of extra space in it. Each application of these two techniques potentially increases the objective value by a factor of $1+\epsilon$, i.e., producing a $1+\epsilon$ loss. The third technique is to call each job small or large with respect to a given interval. We call a job small with respect to I_x if its processing time is less than $\epsilon^3 I_x = \epsilon^4 R_x$; large, otherwise.

Lemma 1 *With a $1+O(\epsilon)$ loss, we can assume that for each job j, both p_j and r_j are integer powers of $1+\epsilon$, and $r_j \geq \epsilon p_j$.*

Proof First, we round up all p_j to integer powers of $1+\epsilon$, which produces a $1+\epsilon$ loss. Second, since adding an idle time being ϵ times the processing time before each batch produces at most a $1+\epsilon$ loss, we can guarantee $r_j \geq \epsilon p_j$ by increasing some release dates. Third, we round up all r_j to integer powers of $1+\epsilon$, which produces a $1+\epsilon$ loss again. □

Lemma 2 *Each batch crosses at most* $s = \left\lceil \log_{1+\epsilon}\left(1+\frac{1}{\epsilon}\right)\right\rceil$ *intervals.*

Proof Let j be the longest job in a batch. Since both r_j and p_j are integer powers of $1+\epsilon$ but $1/\epsilon$ is not, $r_j \geq \epsilon p_j$ implies $r_j > \epsilon p_j$. Since $r_j > \epsilon p_j$, the number of intervals the batch containing job j crosses does not exceed the number of intervals j crosses when it starts at ϵp_j. Then

$$s = \log_{1+\epsilon}(1+\epsilon)p_j - \left\lfloor \log_{1+\epsilon} \epsilon p_j \right\rfloor = \left\lceil \log_{1+\epsilon}\left(1+\frac{1}{\epsilon}\right)\right\rceil .$$

□

Let $R = \min_{j=1}^{n} r_j$ and $D = \max_{j=1}^{n} r_j + \sum_{j=1}^{n} p_j$. Then an optimal schedule will span a time interval in $[R, D]$. We proceed to search for that schedule in $[R, D]$. Let u and v be the indices of the first and last intervals among $\{I_x\}$ that intersect $[R, D]$, namely $u = \log_{1+\epsilon} R$ and $v = \left\lfloor \log_{1+\epsilon} D \right\rfloor - 1$. We group the intervals $\{I_x \,|\, u \leq x \leq v\}$ into blocks in the following manner. Let $t = \left\lceil 5 \log_{1+\epsilon} \frac{1}{\epsilon} \right\rceil$ and $m = \lceil (v-u+1)/t \rceil$. Then $\{I_x \,|\, u \leq x \leq v\}$ is partitioned into m blocks, denoted by $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_m$, where each of the first $m-1$ blocks contains t intervals, while the last block contains the remaining intervals. We schedule all jobs by dynamic programming one block at a time. It is possible that a batch crosses several intervals, but since $t > s$, no batch can cross an entire block. Also, we note that t has been set a much greater value than s for further analysis. Let $F(i, a, U)$ be the minimum total completion time for a given set of jobs U, subject to the constraints: (i) all the jobs start before the end of block $\mathcal{B}_i$; (ii) all the jobs finish no later than a, where a is a time no earlier than the end of $\mathcal{B}_i$. Let $F(0, R, \emptyset) = 0$. Then

$$F(i+1, a', U) = \min_{a \in \mathcal{A},\, V \subseteq U} \{F(i, a, V) + W(i+1, a, a', U-V)\}, \tag{1}$$

where $\mathcal{A}$ is the set of possible values of a and $W(i+1, a, a', U-V)$ is the minimum total completion time for the job set $U-V$, subject to the constraints: (i) all the jobs start between a and the end of $\mathcal{B}_{i+1}$; (ii) all the jobs finish no later than a'. The optimal

objective value is given by $F(m, D, \mathcal{J})$. To implement the dynamic programming scheme in polynomial time, we must show that at each stage, $a\,, a'\,, U$ and V have a polynomial number of choices and $W(i+1, a, a', U-V)$ can be computed in polynomial time.

Theorem 1 *With a $(1+\epsilon)^2$ loss, we can assume that a and a' in (1) each have at most s choices.*

Proof Consider all blocks in order of increasing indices. If the last batch starting in $\mathcal{B}_1$ crosses out of $\mathcal{B}_1$ and finishes at time $C \in I_{x(1)}$, where $I_{x(1)}$ is one of the first $s-1$ intervals in $\mathcal{B}_2$, we round up $C = R_{x(1)+1}$, which increases the completion time of each batch completing after $R_{x(1)}$ by less than $I_{x(1)}$. If the last batch starting in $\mathcal{B}_i$ crosses out of $\mathcal{B}_i$ and finishes at time $C \in I_{x(i)}$ after the rounding is done for $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_{i-1}$, we will round up $C = R_{x(i)+1}$. Since the last batches in two adjacent blocks are separated by more than $t - 2s > 2\log_{1+\epsilon}\frac{1}{\epsilon}$ intervals, rounding up the completion times of the batches crossing out of $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_i$ increases the completion time of each batch finally completing after the end of $\mathcal{B}_i$ by less than

$$\begin{aligned} I_{x(1)} + \cdots + I_{x(i)} &< \frac{I_{x(i)}}{1 - \left(\frac{1}{1+\epsilon}\right)^{2\log_{1+\epsilon}\frac{1}{\epsilon}}} \\ &= \frac{I_{x(i)}}{1-\epsilon^2} < R_{x(i)+1} - R_{x(i)-1}\,. \end{aligned}$$

Thus, the objective value increases by less than a factor of $(1+\epsilon)^2$ after the rounding is done for all blocks. The analysis shows that a and a' can be restricted to taking the ends of $\mathcal{B}_i$ and $\mathcal{B}_{i+1}$ or the ends of their next $s-1$ intervals, respectively. □

2.2 The choices of U and V

In this subsection, we discuss how to reduce the choices of U and V in (1).

Lemma 3 *The large jobs released at R_x have at most $t = \left\lceil 5\log_{1+\epsilon}\frac{1}{\epsilon}\right\rceil$ distinct processing times, and we can assume that there are at most c/ϵ^3 large jobs with the same processing time released at R_x, which have a total processing time less than cI_x/ϵ^2.*

Proof Since the processing time p of a large job released at R_x satisfies $R_x/\epsilon \geq p \geq \epsilon^4 R_x$ and is an integer power of $1+\epsilon$, the number of distinct p is no more than

$$1 + \left\lfloor \log_{1+\epsilon}\frac{R_x}{\epsilon}\right\rfloor - \left\lceil \log_{1+\epsilon}\epsilon^4 R_x\right\rceil \leq 1 + \left\lfloor 5\log_{1+\epsilon}\frac{1}{\epsilon}\right\rfloor = t\,.$$

The number of large jobs with processing time p that start in I_x is no more than $\frac{cI_x}{\epsilon^3 I_x} = \frac{c}{\epsilon^3}$, and their total processing time is less than $c(I_x + p) < cI_x/\epsilon^2$ if $p < I_x$, and is no more than $cp < cI_x/\epsilon^2$ if $p \geq I_x$. Delaying the extra jobs to the next release date satisfies our assumption. □

Let $l(\mathcal{S})$ denote the total length of all batches when a set $\mathcal{S}$ of available jobs is first sorted according to the SPT rule and then divided into batches such that each batch except the last one contains c jobs. Let $\mathcal{S}_x$ be the set of small jobs starting in I_x. Obviously, $l(\mathcal{S}_x)$ exceeds the minimum total length of the batches consisting of the jobs in $\mathcal{S}_x$ by less than $\epsilon^3 I_x$.

Lemma 4 *With a $1 + O(\epsilon)$ loss, we can assume that (i) no batch contains both small and large jobs; (ii) in each interval, the batches of small jobs are scheduled before the batches of large jobs; (iii) the jobs in $\mathcal{S}_x$ are scheduled as in computing $l(\mathcal{S}_x)$.*

Proof For each I_x, we reschedule the jobs starting in I_x as follows: first sort the jobs according to the SPT rule, and then divide them into batches such that the last batch contains as many jobs as the original last batch, each middle batch contains c jobs, and the beginning batch contains the remaining jobs. The rescheduling does not increase the total length of the batches starting in each I_x and produces at most a $1+\epsilon$ loss. Now, there is at most one mixed batch that starts in I_x and contains both small and large jobs. We separate the small jobs from the mixed batch and reschedule all the small jobs starting in I_x as in computing $l(\mathcal{S}_x)$. The operations increase the total length of the batches starting in I_x by less than $2\epsilon^3 I_x$. Then stretching each I_x by a factor of $1 + \epsilon$ creates enough extra space. This completes the proof. □

Lemma 5 *Let σ_x be the SPT sequence of the unscheduled available jobs at R_x. With a $1 + O(\epsilon)$ loss, we can assume that $\mathcal{S}_x$ is a beginning segment of σ_x.*

Proof Consider all $\mathcal{S}_x$ in order of increasing indices. Let x be the currently smallest index such that $\mathcal{S}_x$ is not a beginning segment of σ_x. We will replace $\mathcal{S}_x$ by $\mathcal{S}'_x$ that is the longest beginning segment of σ_x such that $l(\mathcal{S}'_x) \leq l(\mathcal{S}_x)$. The jobs in $\mathcal{S}_x \setminus \mathcal{S}'_x$ will replace the jobs in $\mathcal{S}'_x \setminus \mathcal{S}_x$. Note that $|\mathcal{S}_x \setminus \mathcal{S}'_x| \leq |\mathcal{S}'_x \setminus \mathcal{S}_x|$. We first divide the jobs in $\mathcal{S}_x \setminus \mathcal{S}'_x$ into batches as in computing $l(\mathcal{S}_x \setminus \mathcal{S}'_x)$.

Let $\mathcal{S}$ be the subset of $\mathcal{S}'_x \setminus \mathcal{S}_x$ consisting of the jobs starting in I_y ($y > x$). We reschedule the small jobs starting in I_y such that the jobs in $\mathcal{S}$ are separated from the others. Since there are at most two extra batches for each distinct processing time of

the jobs in $\mathcal{S}$, the rescheduling increases the total length of the batches starting in I_y by no more than

$$2\sum_{i<0}\epsilon^3 I_x(1+\epsilon)^i = 2\epsilon^2 I_x\,.$$

Then we replace $\mathcal{S}$ by some batches of $\mathcal{S}_x \setminus \mathcal{S}'_x$ exceeding the batches of $\mathcal{S}$ in total length by at most $\epsilon^3 I_x$. Note that we use as many batches as possible to replace $\mathcal{S}$ in earlier I_y.

After all $\mathcal{S}_x$ with $x < y$ are adjusted, the total length of the batches starting in I_y increases by at most

$$\sum_{x<y}\left(2\epsilon^2 I_x + \epsilon^3 I_x\right) = 2\epsilon I_y + \epsilon^2 I_y\,.$$

However, adjusting $\mathcal{S}_x$ with $x \geq y$ does not increase the total length of the batches starting in I_y. Then stretching I_y by a factor of $(1+\epsilon)^2$ creates enough extra space. □

Lemma 6 *Let J_x^S be the set of small jobs released at R_x. With a $1+O(\epsilon)$ loss, we can assume that $l(J_x^S) < (1+\epsilon^3)I_x$.*

Proof By Lemma 5, we can pick the jobs in J_x^S according to the SPT rule and delay the remaining jobs to the next release date. Then the conclusion holds. □

Lemma 7 *With a $1+O(\epsilon)$ loss, we can assume that all the jobs released at R_x finish before R_{x+t+1}.*

Proof By Lemmas 3 and 6, the jobs released at R_x can be divided into batches with a total length less than $(1+\epsilon^3+t/\epsilon^2)I_x$. Since $5\log_{1+\epsilon}\frac{1}{\epsilon} < \frac{1}{\epsilon^2} - 1$ for $0 < \epsilon \leq \frac{1}{100}$, it holds that $t < \frac{1}{\epsilon^2} - \epsilon^2 - \epsilon^5$, and hence,

$$\left(1+\epsilon^3+\frac{t}{\epsilon^2}\right)I_x < \frac{I_x}{\epsilon^4} < \epsilon(1+\epsilon)^t I_x = \epsilon I_{x+t}\,.$$

Then stretching I_{x+t} by a factor of $1+\epsilon$ creates enough extra space in (R_x, R_{x+t+1}) such that all the jobs released at R_x can finish before R_{x+t+1}. □

According to Lemma 7, the jobs released at R_x will be scheduled in the block containing I_x or the next block. According to Lemma 3, there are at most $(1+c/\epsilon^3)^t$ ways to divide the large jobs released at R_x into two subsets. Since each block contains at most t release dates, there are at most $(1+c/\epsilon^3)^{t^2}$ ways to divide the large jobs released in a block into two subsets, which can be reduced to $(1+ct/\epsilon^3)^t$ by further analysis.

We now consider the number of ways to divide the small jobs released in a block into two subsets. For each J_x^S, by Lemma 5, it suffices to consider the number of ways to divide the SPT sequence of J_x^S into two subsequences. We divide the schedule constructed in computing $l(J_x^S)$ into at most $\lceil (1+\epsilon^3)/\epsilon \rceil = 1/\epsilon + 1$ segments, where the length of each segment is no more than ϵI_x. If a subset $\mathcal{S} \subseteq J_x^S$ containing the first $k-1$ segments and a portion of the kth segment is scheduled in the block containing I_x, then we can enlarge $\mathcal{S}$ to contain the first k segments after stretching I_x by a factor of $1+\epsilon$. This implies that $2+1/\epsilon$ ways are sufficient for dividing J_x^S into two subsets. Then, $(2+1/\epsilon)^t$ ways are sufficient for dividing the small jobs released in a block into two subsets.

Theorem 2 *With a $1+O(\epsilon)$ loss, we can assume that U and V in (1) each have at most $(1/\epsilon)^{O(1)}\,(ct/\epsilon^4)^t$ choices.*

Proof The number of ways to divide the jobs released in a block into two subsets are at most

$$\left(1+\frac{ct}{\epsilon^3}\right)^t \left(2+\frac{1}{\epsilon}\right)^t = \left(\frac{1}{\epsilon}\right)^{O(1)} \left(\frac{ct}{\epsilon^4}\right)^t .$$

Since U should contain all the jobs released in blocks $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_i$ and a portion of the jobs released in block $\mathcal{B}_{i+1}$, its choices are determined by the number of ways to divide the jobs released in block $\mathcal{B}_{i+1}$. Similarly, the choices of V are determined by the number of ways to divide the jobs released in block $\mathcal{B}_i$. □

2.3 Scheduling within a block

In this subsection we discuss how to compute $W(i+1, a, a', U-V)$, given a, a' and the job set $U-V$.

Lemma 8 *With a $(1+\epsilon)^2$ loss, we can assume that the last batch starting in I_x starts at one of the times $R_x + k\epsilon I_x$ ($0 \le k \le \frac{1}{\epsilon} - 1$).*

Proof If the last batch starting in I_x starts in $(R_x + k\epsilon I_x, R_x + (k+1)\epsilon I_x)$ $(0 \le k \le \frac{1}{\epsilon} - 1)$, we can delay its start time to $R_x + (k+1)\epsilon I_x$, which increases the completion time of each batch starting after R_x by less than ϵI_x. Delaying the batches starting in I_x with $x \le y$ to satisfy our assumption increases the completion times of the batches finally starting after R_y by less than

$$\sum_{x \le y} \epsilon I_x < (1+\epsilon) I_y = \epsilon(1+\epsilon) R_y .$$

Thus, the objective value increases by less than a factor of $(1+\epsilon)^2$ after delaying all batches. □

We first schedule the large jobs starting in block $\mathcal{B}_{i+1}$. Note that after the longest job in a batch is determined, the other jobs in the batch can be selected greedily among the currently available jobs. Then a batch is determined completely by its longest jobs. Since the large jobs available at R_x have at most t distinct processing times and at most $1/\epsilon^3$ batches of large jobs can start in I_x, we have at most $(1+t)^{1/\epsilon^3}$ ways to form the batches of large jobs starting in I_x. Since a block contains t intervals, we have at most $(1+t)^{t/\epsilon^3}$ ways to form the batches of large jobs starting in block $\mathcal{B}_{i+1}$. According to Lemma 4, the batches of large jobs will be scheduled after the batches of small jobs in each I_x. So, according to Lemma 8, $\frac{1}{\epsilon^t}(1+t)^{t/\epsilon^3}$ ways suffice for scheduling the large jobs starting in block $\mathcal{B}_{i+1}$. Each way requires no more than $O(ht)$ time, where $h = |U - V|$.

Note that Lemma 8 also implies that no batch of small jobs crosses out of an interval. After the batches of large jobs starting in I_x are scheduled, according to Lemma 5, the set of small jobs in I_x will be taken as the possibly longest beginning segment of σ_x that can be contained in the remaining space in I_x while being scheduled as in computing $l(\mathcal{S}_x)$. It requires no more than $O(h)$ time to schedule the small jobs in I_x given σ_x. So, the following theorem holds.

Theorem 3 *With a $1+O(\epsilon)$ loss, $W(i+1, a, a', U-V)$ in (1) can be computed in $O\left(\frac{t}{\epsilon^t}(1+t)^{t/\epsilon^3}h + h\log h\right)$ time.*

2.4 The main theorem

According to Lemma 7, we may omit the latter one of any two consecutive blocks in which no job is released. Thus, it actually needs no more than $2n$ stages to compute a $1+O(\epsilon)$-approximation of $F(m, D, \mathcal{J})$ by (1). Combining this fact with Theorems 1, 2 and 3, we obtain the following conclusion.

Theorem 4 *The problem of minimizing total completion time with release dates on a batch machine has a PTAS.*

3 The total weighted completion time problem on an unbounded batch machine

In this section, we present an FPTAS for the problem of minimizing total weighted completion time with release dates on an unbounded batch machine. Let $1/4 \geq \epsilon > 0$ and $1/\epsilon$ be integral. We partition the time interval $(0, +\infty)$ into disjoint intervals $\{I_x \,|\, x \in \mathcal{Z}\}$ as in Section 2. Like Lemma 1, the following lemma holds.

Lemma 9 *With a $1 + O(\epsilon)$ loss, we can assume that for each j, $r_j \geq \epsilon p_j$ and r_j are integer powers of $1 + \epsilon$.*

Lemma 10 *With a $1 + O(\epsilon)$ loss, we can assume that each batch completes at one of the times in*

$$\mathcal{A} = \{R_x + k\epsilon I_x \,|\, x \in \mathcal{Z}\,,\ 1 \leq k \leq 1/\epsilon\}\ .$$

Proof Consider all I_x in order of increasing indices. If a batch starts before R_x and completes in $(R_x + (k-1)\epsilon I_x, R_x + k\epsilon I_x)$ $(1 \leq k \leq 1/\epsilon)$, we delay its completion time to $R_x + k\epsilon I_x$. Afterwards, we combine all the batches contained within I_x into a new batch and let the new batch complete at the earliest time in $\{R_x + k\epsilon I_x \,|\, 1 \leq k \leq 1/\epsilon\}$. These two operations increase the completion time of each job finally completing in I_x by less than I_x and the completion time of each batch finally completing after R_{x+1} by less than $2\epsilon I_x$. After performing the two operations for all I_x with $x \leq y$, the completion time of each batch completing in I_y increases by less than

$$\sum_{x<y} 2\epsilon I_x + I_y < 3I_y = 3\epsilon R_y\,.$$

Thus, the objective value increases by less than a factor of $1/(1-3\epsilon)$ after performing the two operations for all I_x. This completes the proof. □

Lemma 11 *In a schedule with the property in Lemma 10, any job completes within $O(1/\epsilon^2)$ intervals of its release date.*

Proof Let job j be released at R_x. It holds that

$$p_j \leq \frac{R_x}{\epsilon} \leq \epsilon^2(1+\epsilon)^t R_x = \epsilon I_{x+t}\,,$$

where $t = \left\lceil 3\log_{1+\epsilon}(1/\epsilon) \right\rceil = O(1/\epsilon^2)$. So, if $(R_{x+t}, R_{x+t} + \epsilon I_{x+t})$ is idle, we can schedule job j into the interval and the conclusion holds. If the interval has been occupied

(wholly or partially) by a batch, we can add job j to the batch, which does not increase the completion time of any other job in a schedule with the property in Lemma 10. Since the batch has a length of no more than ϵI_{x+2t}, j will complete before R_{x+2t}. The conclusion holds too. □

Combining Lemmas 10, 11 and the pseudopolynomial algorithm in [10], we can construct an FPTAS for the total weighted completion time problem on an unbounded batch machine. Let α and γ be the job sequences such that $r_{\alpha(1)} \leq r_{\alpha(2)} \leq \cdots \leq r_{\alpha(n)}$ and $p_{\gamma(1)} \leq p_{\gamma(2)} \leq \cdots \leq p_{\gamma(n)}$, respectively. Let $\alpha(i,j) = \{\alpha(i), \alpha(i+1), \cdots, \alpha(j)\}$ and $\gamma(i,j) = \{\gamma(i), \gamma(i+1), \cdots, \gamma(j)\}$. Let $J(i_1, i_2; k) = \alpha(i_1, i_2) \cap \gamma(1, k)$. In addition, we introduce an auxiliary job $n+1$ with $r_{n+1} = r_{\alpha(n)}$ and $p_{n+1} = w_{n+1} = 0$. Let $\alpha(n+1) = \gamma(n+1) = n+1$. We will schedule job $n+1$ as the last job.

Let $F(i_1, i_2; k_1; k_2; a, a')$ ($k_1 < k_2$ and $a, a' \in \mathcal{A}$) denote the minimum total weighted completion time when scheduling the jobs among $J(i_1, i_2; k_1) \cup \{\gamma(k_2)\}$ into the interval $[a, a']$, subject to the constraint that each batch completes at one of the times in $\mathcal{A}$ and job $\gamma(k_2)$ completes at time a'. If $J(i_1, i_2; k_1) = \emptyset$, then

$$F(i_1, i_2; k_1; k_2; a, a') = \begin{cases} w_{\gamma(k_2)} a', & \text{if } \max\{r_{\gamma(k_2)}, a\} \leq a' - p_{\gamma(k_2)} \\ +\infty, & \text{otherwise.} \end{cases}$$

Generally, $F(i_1, i_2; k_1; k_2; a, a')$ can be computed recursively as follows.

(i) If $\gamma(k_1) \notin \alpha(i_1, i_2)$, then $J(i_1, i_2; k_1) = J(i_1, i_2; k_1 - 1)$ and we have

$$F(i_1, i_2; k_1; k_2; a, a') = F(i_1, i_2; k_1 - 1; k_2; a, a') .$$

(ii) If $\gamma(k_1) \in \alpha(i_1, i_2)$ and $r_{\gamma(k_1)} > a' - p_{\gamma(k_2)}$, then job $\gamma(k_1)$ cannot be scheduled in $[a, a']$, and hence $F(i_1, i_2; k_1; k_2; a, a') = +\infty$.

(iii) If $\gamma(k_1) \in \alpha(i_1, i_2)$ and $r_{\gamma(k_1)} \leq a' - p_{\gamma(k_2)}$, we have

$$F(i_1, i_2; k_1; k_2; a, a') = \min \begin{cases} F(i_1, i_2; k_1 - 1; k_2; a, a') + w_{\gamma(k_1)} a' \\ \min\{H(b) \,|\, b \in \mathcal{A}'\} \end{cases} ,$$

where the first term is taken if job $\gamma(k_1)$ is processed in the batch including job $\gamma(k_2)$, and in the second term,

$$\mathcal{A}' = \left\{ b \in \mathcal{A} \,\middle|\, \max\{r_{\gamma(k_1)}, a\} + p_{\gamma(k_1)} \leq b \leq a' - p_{\gamma(k_2)} \right\}$$

and $H(b) = H_1(b) + H_2(b)$ is taken if job $\gamma(k_1)$ completes at time b. We note that the first term will not be taken when $k_2 = n+1$, i.e., job $n+1$ will occupy the last batch alone.

$H_1(b)$ is the contribution to $H(b)$ of the jobs processed in $[a, b]$. It is reasonable to assume that none of the jobs with release dates no more than $b-p_{\gamma(k_1)}$ in $J(i_1, i_2; k_1-1)$ is scheduled after the batch including job $\gamma(k_1)$ since they have processing times no more than $p_{\gamma(k_1)}$. Let i'_2 $(i_1 \le i'_2 \le i_2)$ be the maximum index satisfying $r_{\alpha(i'_2)} \le b - p_{\gamma(k_1)}$. Then,

$$H_1(b) = F(i_1, i'_2; k_1 - 1; k_1; a, b)\,.$$

$H_2(b)$ is the contribution to $H(b)$ of the jobs processed in $[b, a']$. It obviously holds that

$$H_2(b) = F(i'_2 + 1, i_2; k_1 - 1; k_2; b, a')\,.$$

By computing $F(1, n; n; n + 1; r_{\alpha(1)}, L)$ recursively, where $L = r_{\alpha(n)}(1 + \epsilon)^{O(1/\epsilon^2)}$, we can obtain a $1 + O(\epsilon)$-approximation of the optimal objective value. A $1 + O(\epsilon)$-approximate schedule can be found by backtracking.

Now we analyse the complexity of the recursion in the above dynamic programming formulation. According to Lemma 11, we need only to consider $O(n/\epsilon^2)$ intervals immediately following n release dates. In each interval, a and a' each have $O(1/\epsilon)$ choices. Then, a and a' together have $O(n^2/\epsilon^6)$ choices, and the size of the domain of $F(i_1, i_2; k_1; k_2; a, a')$ is $O(n^6/\epsilon^6)$. To obtain the vaule of each $F(i_1, i_2; k_1; k_2; a, a')$, we need at most $O(n/\epsilon^3)$ time (see cases (i)-(iii)). Thus, the complexity of the recursion is $O(n^7/\epsilon^9)$, which leads to the following conclusion.

Theorem 5 *The problem of minimizing total weighted completion time with release dates on an unbounded batch machine has an FPTAS.*

Acknowledgments

This research was supported in part by The Hong Kong Polytechnic University under a grant from the *ASD in China Business Services*. The first author was also supported by the National Natural Science Foundation of China under grant number 10101007.

References

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, M. Sviridenko. Approximation schemes for

minimizing average weighted completion time with release dates. Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (1999) 32–43.

[2] P. Brucker, A. Gladky, H. Hoogeveen, M.Y. Kovalyov, C.N. Potts, T. Tautenhahn, S.L. van de Velde. Scheduling a batching machine. Journal of Scheduling 1 (1998) 31–54.

[3] M.-C. Cai, X. Deng, H. Feng, G. Li, G. Liu. A PTAS for minimizing total completion time of bounded batch scheduling. Lecture Notes in Computer Science 2337 (2002) 304–314.

[4] B. Chen, X. Deng, W. Zang. On-line scheduling a batch processing system to minimize total weighted job completion time. Lecture Notes in Computer Science 2223 (2001) 380–389.

[5] X. Deng, H. Feng, G. Li, B. Shi. A PTAS for semiconductor burn-in scheduling. Journal of Combinatorial Optimization 9 (2005) 5–17.

[6] X. Deng, H. Feng, P. Zhang, H. Zhu. A polynomial time approximation scheme for minimizing total completion time of unbounded batch scheduling. Lecture Notes in Computer Science 2223 (2001) 26–35.

[7] X. Deng, Y. Zhang. Minimizing mean response time in batch processing system, Lecture Notes in Computer Science 1627 (1999) 231–240.

[8] D.S. Hochbaum, D. Landy. Scheduling semiconductor burn-in operations to minimize total flowtime. Operations Research 45 (1997) 874–885.

[9] C.-Y. Lee, R. Uzsoy, L.A. Martin-Vega, Efficient algorithms for scheduling semiconductor burn-in operations, Operations Research 40 (1992) 764–775.

[10] Z. Liu, J. Yuan, T.C.E. Cheng. On scheduling an unbounded batch machine. Operations Research Letters 31 (2003) 42–48.