Theoretical Computer Science 410 (2009) 2732–2745

Contents lists available at ScienceDirect

Theoretical Computer Science

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

A PTAS for parallel batch scheduling with rejection and dynamic job arrivals☆

Zhigang Cao *, Xiaoguang Yang

Key Laboratory of Management, Decision & Information Systems, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, 100190, PR China

ARTICLE INFO

Article history:
Received 20 April 2008
Received in revised form 5 April 2009
Accepted 9 April 2009
Communicated by X. Deng

Keywords:
Scheduling
Parallel batch
Dynamic job arrivals
Rejection
PTAS

ABSTRACT

In the parallel batch scheduling model, a group of jobs can be scheduled together as a batch while the processing time of this batch is the greatest processing time among its members; in the model of scheduling with rejection, any job can be rejected with a corresponding penalty cost added to the objective value. In this paper, we present a PTAS for the combined model of the above two scheduling models where jobs arrive dynamically. The objective is to minimize the sum of the makespan of the accepted jobs and the total penalty of the rejected ones. Our basic approaches are dynamic programming and roundings.

1. Introduction

Both the models of parallel batch scheduling and scheduling with rejection have their deep roots in the real world. In the industry of semiconductor manufacturing, the last stage is the final testing (called the burn-in operation). In this stage, chips are loaded onto boards, which are then placed in an oven and exposed to high temperature. Each chip has a pre-specified minimum burn-in time, and the burn-in oven has a limited capacity b. Up to b chips (which is called a batch) can be baked in an oven simultaneously, and the baking process is not allowed to be interrupted. To ensure that no defective chips will be passed to the customer, the processing time of a batch is that of the longest one among these chips. As the baking process in burn-in operations is much longer than other testing operations, an efficient algorithm for batching and scheduling is highly meaningful.

The basic motivation of the model of scheduling with rejection is simple. For the traditional research, it is always assumed that we have to process all the jobs. In the real world, however, when the processing time of some job is too large, we can reject processing it. To reject a job, of course, we should pay a corresponding penalty. The problem is how to make a tradeoff between some regular objective, say makespan, of the accepted jobs, and the total penalty of the rejected ones. When all the penalties are large enough, we have to accept all the jobs, thus the model of scheduling with rejection is a generalization of the classic scheduling model. Whether rejection is allowed or not really weights. For instance, the trivial problem $1|r_j|C_{\max}$ becomes NP-hard, when rejection is allowed, even if there are only two release times [6].

Next, we shall give literature reviews for the two models, respectively. The parallel batch scheduling model has been studied extensively since the late 1990s, we only list the primary results related with the makespan objective function. Interested readers can refer to the excellent surveys [3,18]. In contrast, relatively few researchers have concentrated on the

☆ The research is supported by the National Natural Science Foundation of China under Grants 70425004, 90818026.

* Corresponding author. Tel.: +86 010 82680259.
E-mail addresses: cullencao@gmail.com (Z. Cao), xgyang@iss.ac.cn (X. Yang).

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.04.006

rejection model, so we give a relatively complete review. In the 3-tuple notation of Graham et. al. [11], we put a *p-batch* in the middle field to specify the parallel batch scheduling model and we use TP in the rejection model as the abbreviation of total penalty.

The first paper studying the parallel batch scheduling model was due to Ikura and Gimple [14], who considered a special case of $1|p\text{-}batch, r_j|C_{\max}$, where the jobs have identical processing times. They showed that this special case can be solved in $O(n)$ time using a simple strategy. For the special case where there are two release times, Lee and Uzsoy [17] proposed an algorithm running in $O(nb^2P_{\text{sum}}P_{\max})$ time, where $P_{\max}$ and P_{sum} are the maximum and total processing time, respectively. For the general problem $1|p\text{-}batch, r_j|C_{\max}$, Brucker et al. [3] showed that it is equivalent to $1|p\text{-}batch, r_j|L_{\max}$ and therefore is strongly NP-hard even if $b = 2$. Deng et al. [8] present the first PTAS for the general case, which is the best possible theoretically speaking. Li et al. [15], Poon and Zhang [19] designed more delicate PTASs, respectively. There were also similar results on parallel machines [16,23].

For the model of scheduling with rejection, as to the makespan criterion, where our objective function is the sum of the makespan of the accepted jobs and TP of the rejected ones, Bartal et al. [4] studied the off-line version as well as the on-line version on identical parallel machines. Later, the uniform machines variant was considered: He et al. [12] present the best possible on-line algorithms for the two-machine case and a special three-machine case; Hoogeven et al. [13] and Seiden [20] also considered the preemptive version on parallel machines. Cao and Zhang [6] were the first to study the dynamic job arrival problem: They proved its NP-hardness and designed a PTAS for the off-line version; for the on-line version, a best possible algorithm with competitive ratio $(\sqrt{5}+1)/2$ was presented.

As to the total weighted completion time criterion, Engels et al. [9] addressed the off-line version and Epstein et al. [10] paid attention to the on-line version for a unit-weight-unit-processing-time special case. Noticing that scheduling with rejection is in fact a bi-criteria optimization problem, Cao et al. [5] studied this model through treating TP as a constraint.

Sengupta [21] also discussed the maximum lateness criterion and gave a PTAS through dynamic programming.

In this paper, we shall concern a combined model where jobs can be either processed in batches or be rejected. Jobs arrive dynamically and the objective is to minimize $C_{\max}+\text{TP}$. Trivially, this problem is strongly NP-hard. Our main result is a PTAS, and the basic approaches are dynamic programming and roundings. Our main idea, which was first explored by Deng et al. [7], is to treat *large* and *small* jobs separately. We note that a special case of this problem where jobs arrive simultaneously is proved solvable in $O(n^2 \log n)$ [22].

2. Model description and preliminaries

We are given a set of jobs $\{J_1, J_2, \ldots, J_n\}$, where each job J_j is characterized by a processing time p_j, a release time (arrival time) r_j, before which J_j can't be processed, and a penalty cost e_j, which we pay if J_j is rejected, $1 \le j \le n$. We assume all the numbers are non-negative integers. Up to b jobs can be processed together as a batch, and the processing time of a batch is the largest one among its members. Let $p(B)$ denote the processing time of a batch B, then $p(B) = \max\{p_j : J_j \in B\}$.

We also use $r(B)$ to denote the release time of B, i.e. the largest release time in B, and we use $s(B)$ and $c(B)$ to denote the start time and completion time of B respectively in some schedule. Obviously, $s(B) \ge r(B)$. We call a series of batches in some schedule $B_1, B_2, \ldots, B_k$ a *block*, if $c(B_i) = s(B_{i+1})$, for all $1 \le i \le k-1$. For some block $\mathcal{B}$, $s(\mathcal{B})$ and $c(\mathcal{B})$ are the start time and completion time of this block, respectively. Time intervals between blocks are called *gaps* or *idle times*.

The problem is (a) to accept a subset of the jobs, (b) to batch the accepted jobs and (c) to schedule these batches, such that the objective function $C_{\max}+\text{TP}$ is minimized, where $C_{\max}$ is the makespan or the maximum completion time of the accepted jobs and TP is the total penalty of the rejected ones. Here in (c), *schedule* means that we give a sequence of the batches and specify the start times of all the batches. Since the problem $1||C_{\max}$ can be solved trivially by processing all the jobs in any sequence as long as there is no idle time for the machine, and the problem $1|r_j|C_{\max}$ can be simply solved by processing any available job whenever the machine is free, we can see that the key points in problem $1|p\text{-}batch, rej, r_j|C_{\max}+\text{TP}$ are procedures (a) and (b), because as long as we know how to accept the jobs and how to batch them, we can treat each batch as a single job safely. The noun *schedule* means a solution, not definitely an optimal one, for some scheduling problem, which is denoted by π in this paper.

Next, we shall give some preliminary results. The problem $1|p\text{-}batch|C_{\max}$ can be easily solved in $O(n \log n)$ time by the famous yet simple FBLPT rule (Full Batch Largest Processing Time first) [2]. In this paper, we need two versions of it.

Algorithm FBLPT(1)

STEP 1. Index all the jobs in non-decreasing order of their processing times, i.e. $p_1 \le p_2 \le \cdots \le p_n$.

STEP 2. Take $J_1, J_2, \ldots, J_{i_0}$ as a batch, and denote it by B_1, where $i_0 = n - b(\lceil n/b \rceil - 1)$. For the other jobs, from J_{i_0+1} to J_n, take b of them in turn as a batch, and denote the batches as $B_2, B_3, \ldots, B_{\lceil n/b \rceil}$, respectively.

STEP 3. Sequence the batches in increasing order of their indices.

FBLPT(2) is similar, except that the jobs are indexed in non-increasing order of their processing times, and the last batch may not be full. The correctness of the two algorithms is straightforward. The following properties for $1|p\text{-}batch, rej|C_{\max}+\text{TP}$ are useful for attacking more complicated problems.

Lemma 1 ([22]). *There is an optimal schedule for* $1|p\text{-}batch, rej|C_{\max}+\text{TP}$ *such that the accepted jobs are scheduled according to the FBLPT(1) rule.* □

Suppose that π is an optimal schedule for $1|p\text{-}batch, rej|C_{\max}+\text{TP}$ which satisfies the FBLPT(1) rule. Its accepted batches are $B_1, B_2, \ldots, B_k$ and the largest jobs in these batches are $J_{i_1}, J_{i_2} \ldots, J_{i_k}$, respectively. Suppose also that the jobs have been indexed in non-decreasing order of their processing times. It's not hard to see that the rejected jobs whose indices are between i_{l-1} and i_l must be the ones with smaller penalty costs. More precisely, denote by $g(i, j)$ as the sum of the $j-i+1-(b-1)$ smallest penalties among jobs $J_i, J_{i+1}, \ldots, J_j, j-i+1 \geq b-1$, we have:

Lemma 2 ([22]). $\sum_{\substack{J_j \notin B_l \\ i_{l-1}<j<i_l}} e_j = g(i_{l-1}+1, i_l-1)$, *for all* $2 \leq l \leq k$. □

The above property may not hold for B_1 as it may contain less than b jobs, and it is straightforward that if this exception does occur, it must hold that $|B_1| = i_1$, where $|B_1|$ denotes the cardinality of the set B_1, that is, all the jobs whose indices are equal to or less than i_1 are accepted.

Lemma 3 ([22]). *The total running time for computing all the* $g(i, j)$*'s* ($1 \leq i, j \leq n, j-i \geq b-2$) *can be bounded by* $O(n^2 \log b)$. □

In the next sections, i, j in the above lemma will be restricted to be indices of jobs with the same release times, but similar results still hold.

For any optimization problem, if it has a family of algorithms $\{\mathcal{A}_\epsilon\}$, such that for any given error parameter $\epsilon > 0$, algorithm $\mathcal{A}_\epsilon$ generates a solution whose objective value is at most $1+\epsilon$ times the optimal one, and the running time is a polynomial of the input size while ϵ is taken as a constant, we say that $\{\mathcal{A}_\epsilon\}$ is a PTAS (Polynomial Time Approximation Scheme) for this problem. Further, if the running time is also a polynomial of $1/\epsilon$, we call $\{\mathcal{A}_\epsilon\}$ an FPTAS (Fully PTAS). It is well known that strongly NP-hard problems do not admit an FPTAS.

Without loss of generality, we assume in this paper that $1/\epsilon$ is integral. For any instance I of some scheduling problem, if we can transform it into a new instance I' which satisfies some property $\mathcal{P}$, and for any optimal schedule π' of I', the corresponding schedule π of I (which may not be unique) has an objective value at most $1+O(\epsilon)$ times the optimal objective value, we say that *with* $1+O(\epsilon)$ *loss, we assume that this scheduling problem has property* $\mathcal{P}$. The most frequently used transformations are identical rounding and geometry rounding. This language was explored by Afrita et al. [1]. It simplifies greatly the presentation of our ideas. We also overload this language by saying that *with* $1+O(\epsilon)$ *loss we assume that the schedules meet some property*, if there exists at least one schedule meeting this property whose objective value is at most $1+O(\epsilon)$ times the optimal one.

Lemma 4 ([15]). *With* $1+\epsilon$ *loss, we assume that there are at most* $1/\epsilon$ *distinct release times for any instance of* $1|p\text{-}batch, rej, r_j|C_{\max}+\text{TP}$. □

The above result in [15] (Lemma 1) is actually for the problem $1|p\text{-}batch, r_j|C_{\max}$ and it carries over trivially to the problem we concern, so we omit the proof. Another remark is that we use $1/\epsilon$ instead of $1/\epsilon+1$ for simplicity. This modification is valid since we can choose a slightly smaller ϵ at the very beginning.

Throughout this paper, we denote by *Opt* the optimal value of problem $1|p\text{-}batch, rej, r_j|C_{\max}+\text{TP}$, we use $m(\pi)$ to denote the objective value of some schedule π. We also use d to denote the optimal value of an imaginative scheduling problem where all jobs have the same parameters as in the problem concerned in this paper, except that all of them arrive at time zero. From [22], we know that d can be calculated in $O(n^2 \log b)$ time. It is straightforward that d is a lower bound for *Opt*. It is still not hard to see that $r_{\max}+d$ is an upper bound for *opt* as we can get a feasible schedule by accepting and batching the jobs as we do in the schedule corresponding to d and scheduling these batches from time $r_{\max}$ on.

Lemma 5. *If* $\{\mathcal{A}_\epsilon\}$ *is a PTAS for the restricted problem of* $1|p\text{-}batch, rej, r_j|C_{\max}+\text{TP}$ *in which* $Opt \geq r_{\max}$*, then the general problem of* $1|p\text{-}batch, rej, r_j|C_{\max}+\text{TP}$ *admits a PTAS, whose running time is at most n times that of* $\{\mathcal{A}_\epsilon\}$.

Proof. For any instance of the general problem, suppose that there are m distinct release times $r^1 < r^2 < \cdots < r^m$, and the corresponding sets of jobs are $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_m$. Algorithm $\mathcal{B}_{i\epsilon}$ ($1 \leq i \leq m$) is constructed from $\mathcal{A}_\epsilon$ easily as follows. There are two steps. In the first step, $\mathcal{B}_{i\epsilon}$ rejects all the jobs in $\mathcal{R}_i, \mathcal{R}_{i+1}, \ldots, \mathcal{R}_m$; In the second step, $\mathcal{A}_\epsilon$ is called to schedule all the remaining jobs. $\mathcal{B}_\epsilon$ simply runs $\mathcal{B}_{1\epsilon}, \mathcal{B}_{2\epsilon}, \ldots, \mathcal{B}_{m\epsilon}$ and picks a schedule with the smallest objective value.

We claim that $\{\mathcal{B}_\epsilon\}$ is a PTAS for the general problem. In fact, suppose that π^* is an optimal schedule for the general problem, π^* rejects all the jobs in $\mathcal{R}_i, \mathcal{R}_{i+1}, \ldots, \mathcal{R}_m$ and accepts at least one job in $\mathcal{R}_{i-1}$ ($1 \leq i \leq m$. $i=1$ means that π^* rejects all the jobs). Then, $\mathcal{B}_{i\epsilon}$ outputs a $1+\epsilon$ approximate solution. So does $\mathcal{B}_\epsilon$. It is trivial that the running time of $\mathcal{B}_\epsilon$ is at most n time of $\mathcal{A}_\epsilon$ since $m \leq n$. □

Due to Lemma 5, we assume later that

Assumption 1. $Opt \geq r_{\max}$.

Therefore we have

$$\max\{r_{\max}, d\} \leq Opt \leq r_{\max}+d. \tag{1}$$

Notice that a 2-approximate algorithm can be simply designed just as we do in getting the upper bound $r_{\max}+d$. Let $M = \epsilon \max\{r_{\max}, d\}$, then:

$$M/\epsilon \leq Opt \leq 2M/\epsilon. \tag{2}$$

Lemma 6. *Jobs with processing time larger than $r_{\max} + d$ should be rejected in any optimal schedule.* □

Due to the above trivial lemma, we assume later that

Assumption 2. $p_{\max} \leq r_{\max} + d$.

We will call jobs with processing times equal to or less than ϵM *small* jobs. Otherwise, they will be called *large* jobs. In the next section, we shall consider a special case where all the jobs are small. We present an FPTAS for this case, which tells us that this special case is at most weakly NP-hard and the strongly NP-hardness of the general problem comes from the large jobs. The analysis of this case also makes it easier for that of the general case.

3. A special case with purely small jobs

We assume in this section that the jobs are indexed in the non-decreasing order of their release times, that is,

$$r_1 \leq r_2 \leq \cdots \leq r_n. \tag{3}$$

For the jobs with the same release times, they are indexed in the non-decreasing order of their processing times. We still assume that there are m distinct release times in total: $r^1 < r^2 < \cdots < r^m$. By Lemma 4,

$$m \leq 1/\epsilon. \tag{4}$$

Job J_j has a release time r^{u_j}, that is $r_j = r^{u_j}$. The jobs arriving at time r^i are $J_{n_{i-1}+1}, J_{n_{i-1}+2}, \ldots, J_{n_i}$, $1 \leq i \leq m$, $n_0 = 0$.

Theorem 1. *With $1 + \epsilon$ loss, we assume that:*

Property 1 (*No Combined Batch*). *Accepted jobs in any batch have the same release time;*

Property 2 (*First Come First Serve*). *Batches with earlier release times are sequenced before the later ones;*

Property 3 (*FBLPT Rule*). *Accepted jobs with the same release time are batched and sequenced according to the FBLPT(1) rule;*

Property 4 (*No Gap*). *All the batches form a block. That is, there is no idle time between any two adjacent batches.*

Proof. Properties 2 and 3 are trivial, and they does not incur any loss in the accuracy. Property 4 can be easily satisfied without any loss in accuracy as we can fasten the last batch and push all the others forward until there are no idle times between adjacent batches. The restriction of Property 1 causes an error of at most ϵ. Since the proof is the same as that in [15] (Theorem 4), we omit it here. □

Next, we shall find through dynamic programming a schedule satisfying the properties in Theorem 1 and such that the objective value is minimized. Let $f_1(j, p)$ be the smallest total penalty among all the sub-schedules of jobs $J_1, J_2, \ldots, J_j$ that (a) satisfy the four properties in Theorem 1, (b) have a completion time of p and (c) accept J_j. $f_0(j, p)$ is similarly defined except that J_j is rejected. It is easy to see that

$$f_0(j, p) = \min\{f_0(j-1, p), f_1(j-1, p)\} + e_j. \tag{5}$$

Next, we shall discuss in three cases the recursive relationships of $f_1(j, p)$.

Case 1. $p - p_j < r_j$. In this case, it is impossible to accept J_j. As a matter of fact, if J_j is accepted, by Properties 1–3 in Theorem 1 and the way that jobs are indexed, J_j has a completion time p and its start time is $p - p_j$, which is less than its release time. A contradiction.

Case 2. $p - p_j \geq r_j$ and $j - n_{u_j-1} \leq b$. $p - p_j \geq r_j$ means J_j is available at time $p - p_j$. Recall that r^{u_j} is the release time of J_j and n_{u_j-1} is the index of the last as well as the largest job that arrives immediately before r^{u_j}. Therefore, $j - n_{u_j-1}$ equals the number of the jobs among $J_1, J_2, \ldots, J_j$ that arrive at r^{u_j}. Since $j - n_{u_j-1} \leq b$ and J_j is accepted, all the $j - n_{u_j-1}$ jobs should be accepted. By Property 1, no more jobs are batched together with them, that is, the $j - n_{u_j-1}$ jobs alone form a batch. Therefore, by Property 4, the completion time of the rest jobs should be $p - p_j$. Hence,

$$f_1(j, p) = \min\{f_0(n_{u_j-1}, p - p_j), f_1(n_{u_j-1}, p - p_j)\}. \tag{6}$$

Case 3. $p - p_j \geq r_j$ and $j - n_{u_j-1} > b$. The only difference between this case and Case 2 is that it is not necessarily true that all the jobs arriving at r^{u_j} are accepted, and even if they are all accepted, they may form more than one batches. Yet, similar to the observation in Lemma 2, we have the following two arguments:

• If the accepted jobs arriving at r^{u_j} form exactly one batch, the contribution that these jobs make to $f_1(j, p)$ should be $g(n_{u_j-1} + 1, j - 1)$, or equivalently $g(j - (j - n_{u_j-1} - 1), j - 1)$. For any $b - 1 \leq k < j - n_{u_j-1}$, denote

$$\Delta_0(p, j, k) = g(j - k, j - 1) + f_0(j - k - 1, p - p_j) \tag{7}$$

and

$$\Delta_1(p, j, k) = g(j - k, j - 1) + f_1(j - k - 1, p - p_j), \tag{8}$$

we have in this subcase that

$$f_1(j,p) = \min\{\Delta_0(p,j,j-n_{u_j-1}-1),\ \Delta_1(p,j,j-n_{u_j-1}-1)\}. \tag{9}$$

• If the accepted jobs arriving at r^{u_j} form more than one batches, we can consider all the possibilities by enumerating which is the largest job in the penultimate batch. More precisely, suppose that $J_{j-(k+1)}$ is a candidate, then the contribution that jobs $J_{j-k}, J_{j-k+1}, \ldots, J_j$ make to $f_1(j,p)$ should be $g(j-k,j-1)$. Therefore, we have in this situation $f_1(j,p) = \Delta_1(p,j,k)$. And hence we have in this subcase that

$$f_1(j,p) = \min\{\Delta_1(p,j,k) : b-1 \le k < j-n_{u_j-1}-1\}. \tag{10}$$

Thereby, the recursive relationship in Case 3 is:

$$f_1(j,p) = \min\left\{\Delta_0(p,j,j-n_{u_j-1}-1),\ \Delta_1(p,j,j-n_{u_j-1}-1),\ \min\{\Delta_1(p,j,k) : b-1 \le k < j-n_{u_j-1}-1\}\right\}. \tag{11}$$

In sum, the overall recursive equation is as follows:

Algorithm DP1

$$f_0(j,p) = \min\{f_0(j-1,p), f_1(j-1,p)\} + e_j;$$

$$f_1(j,p) = \begin{cases} \infty & \text{if } p - p_j < r_j \\ \min\{f_0(n_{u_j-1}, p-p_j), f_1(n_{u_j-1}, p-p_j)\} & \text{if } p-p_j \ge r_j \,\&\, j - n_{u_j-1} \le b \\ \min\begin{Bmatrix} \Delta_0(p,j,j-n_{u_j-1}-1),\ \Delta_1(p,j,j-n_{u_j-1}-1), \\ \min\{\Delta_1(p,j,k) : b-1 \le k < j-n_{u_j-1}-1\} \end{Bmatrix} & \text{if } p-p_j \ge r_j \,\&\, j-n_{u_j-1} > b. \end{cases}$$

The computations are done over all $2 \le j \le n$ and $1 \le p \le (1+2\epsilon)(r_{\max}+d)$.

Remark 1. Doing the computations over $1 \le p \le r_{\max+d}$ is not enough, as our objective is not to find an optimal schedule but an approximate one, whose objective value may be larger than $r_{\max}+d$, but will never be larger than $(1+2\epsilon)(r_{\max}+d)$, where the coefficient $1+2\epsilon$ is from Lemma 4 and Theorem 1. It is also valuable to notice that $1+2\epsilon$ is large enough and $(1+\epsilon)^2$ is unnecessary, because each of the restrictions in Lemma 4 and Theorem 1 causes an absolute error of ϵOpt in the objective value and absolute errors are additive. Similar remarks will still occur in the rest of this section as well as in the next section, and the same argument will still apply.

The boundary conditions are:

$$f_0(1,p) = e_1, \quad \text{for all } 1 \le p \le (1+2\epsilon)(r_{\max}+d); \tag{12}$$

$$f_1(1,p) = \begin{cases} 0 & \text{if } p \ge r_1 + p_1 \\ \infty & \text{if } p < r_1 + p_1 \end{cases}, \quad \text{for all } 1 \le p \le (1+2\epsilon)(r_{\max}+d). \tag{13}$$

After computing all the $f_0(n,p)$'s and all the $f_1(n,p)$'s, calculate

$$\min\{f_0(n,p)+p, f_1(n,p)+p : 1 \le p \le (1+2\epsilon)(r_{\max}+d)\}, \tag{14}$$

and output the corresponding schedule. By Lemma 4 and Theorem 1, this generates a schedule whose objective value is at most $1+2\epsilon$ times the optimal one.

It's not hard to see that the running time of DP1 is pseudo-polynomial. In fact, it can be bounded by $O(n^2(r_{\max}+d+\log b))$, where $O(n^2 \log b)$ is from the computing of $g(i,j)$'s (Lemma 3).

For any positive number x, let

$$x' = \left\lfloor \frac{x}{M/n} \right\rfloor M/n. \tag{15}$$

$f_1'(j,q)$ and $f_0'(j,q)$ are similarly defined as $f_1(j,q)$ and $f_0(j,q)$, respectively, except that the completion time of each batch is an integral multiple of M/n. Similar to DP1, we have the following dynamic programming.

Algorithm DP1′

$$f_0'(j,q) = \min\left\{f_0'(j-1,q), f_1'(j-1,q)\right\} + e_j;$$

$$f_1'(j,q) = \begin{cases} \infty & \text{if } q - p_j < r_j \\ \min\{f_0'(n_{u_j-1}, (q-p_j)'), f_1'(n_{u_j-1}, (q-p_j)')\} & \text{if } q-p_j \ge r_j \,\&\, j - n_{u_j-1} \le b \\ \min\begin{Bmatrix} \Delta_0'(q,j,j-n_{u_j-1}-1),\ \Delta_1'(q,j,j-n_{u_j-1}-1), \\ \min\{\Delta_1'(q,j,k) : b-1 \le k < j-n_{u_j-1}-1\} \end{Bmatrix} & \text{if } q-p_j \ge r_j \,\&\, \text{and } j-n_{u_j-1} > b \end{cases}$$

where

$$\Delta_0'(q,j,k) = g(j-k,j-1) + f_0'(j-k-1, (q-p_j)'), \tag{16}$$

$$\Delta_1'(q,j,k) = g(j-k,j-1) + f_1'(j-k-1, (q-p_j)'), \tag{17}$$

and the computations are done over all $2 \le j \le n$, and and over all $q \in Q_0 = \{M/n, 2M/n, \ldots, t_0 M/n\}$,

$$t_0 = \left\lceil \frac{(1+3\epsilon)(r_{\max}+d)}{M/n} \right\rceil. \tag{18}$$

The coefficient $1+3\epsilon$ is from proof of the next theorem, for the same reason as argued in Remark 1. The boundary conditions are:

$$f_0'(1,q) = e_j, \quad \text{for all } q \in Q_0; \tag{19}$$

$$f_1'(1,q) = \begin{cases} 0 & \text{if } q \ge r_1 + p_1 \\ \infty & \text{if } q < r_1 + p_1. \end{cases} \tag{20}$$

After computing all the $f_0'(n,q)$'s and all the $f_1'(n,q)$'s, calculate

$$\min\{f_0'(n,q)+q, f_1'(n,q)+q : q \in Q_0\}, \tag{21}$$

and output the corresponding schedule.

Theorem 2. *Algorithm DP1′ is an FPTAS for the small job special case of* $1|p\text{-batch}, rej, r_j|C_{\max} + \mathrm{TP}$, *with a running time of* $O((1/\epsilon)n^3)$.

Proof. For the accuracy, it suffices to show that there is a schedule whose objective value is at most $1+2\epsilon$ times the optimal one, such that all the properties in Theorem 1 are satisfied and the completion time of each accepted batch is integral times of M/n. Suppose that π is a schedule generated by DP1′, π' is the schedule constructed by rounding the completions of all the batches in π one by one such that the completion time of each batch is integral times of M/n. Since each batch in π is rounded by at most M/n and there are at most n batches in total, we have

$$m(\pi') - m(\pi) \le (M/n)n = M \le \epsilon Opt. \tag{22}$$

By Lemma 4 and Theorem 1, $m(\pi) \le (1+2\epsilon)Opt$. Thus we have $m(\pi') \le (1+3\epsilon)Opt$. Since

$$t_0 \le \frac{(1+3\epsilon)(r_{\max}+d)}{M/n} + 1 \tag{23}$$

$$= \frac{(1+3\epsilon)(r_{\max}+d)}{\epsilon \max\{r_{\max}, d\}/n} + 1 \tag{24}$$

$$\le 2(1+3\epsilon)(1/\epsilon)n + 1, \tag{25}$$

the running time can be bounded by $O(n^2 \log b + n^2 t_0) = O((1/\epsilon)n^3)$. □

4. The general case

In this section, we shall tackle the general case. Our basic idea to a great extent is to treat large jobs and small jobs separately.

4.1. Several properties

For any accepted batch of jobs, if all its members are small, we call it a small batch. Similarly, if all its members are large, we call it a large batch. Otherwise, if the batch contains both small jobs and large jobs, we call it a mixed batch. With a reasonable cost, we shall not consider mixed batches. The idea of the next result is from Lemma 6 of [15]. The proof is omitted here, because it is the same as in [15].

Theorem 3 (*[15]*). *With* $1+\epsilon$ *loss, we assume that:*

Property 5 (*No Mixed Batch*). *There is no batch that contains both small jobs and large jobs.* □

The next theorem tells us that, with some reasonable cost in precision, we can schedule the accepted large batches first and then insert the small ones into the gaps between large batches. This is the fundamental idea of our final algorithm.

Theorem 4. *With* $1+\epsilon$ *loss, we assume that:*

Property 6 (*Priority of Large Batches*). *For any small batch B, let* $s(B,\pi)$ *and* $c(B,\pi)$ *be its start time and completion time in* π, *respectively, then there are no available large batches during the period* $[s(B,\pi), c(B,\pi))$.

Proof. If some large batch arrives during $[s(B,\pi), c(B,\pi))$, we can let the machine idle from $s(B,\pi)$ on until this large batch arrives. We first process the large batch and then the batch B. This adds at most ϵM to the objective value since B is a small batch. As this may only happen when $[s(B,\pi), c(B,\pi))$ contains some release time, the total cost of this operation can be bounded by

$$m\epsilon M \le (1/\epsilon)\epsilon M = M \le \epsilon Opt, \tag{26}$$

which completes the proof. □

We still need to restrict the way of batching and sequencing. The next result is similar to Theorem 1.

Theorem 5. *With $1+\epsilon$ loss, we assume that:*

Property 7 (*Simple Rules for Small Jobs*). *Accepted small jobs are batched and sequenced according to Properties 1–3 in Theorem 1.* □

For any schedule π that meets Properties 5 and 6, let $\mathcal{L}_i$ denote the set of accepted large jobs that start processing during $[r^i, r^{i+1})$, $1 \le i \le m$, $r^{m+1} = \infty$. Note that their completion times may not be in this interval. Similar to Lemma 2.1 of [8], we have the following property.

Theorem 6 (*[8]*). *There exists a schedule whose objective value is not greater than π such that:*

Property 8 (*FBLPT Rule for Large Jobs*). *All the accepted jobs in $\mathcal{L}_i$ are batched and sequenced according to the FBLPT(2) rule.* □

Remark 2. The above property is quite different from Property 3. Neither can we require that large jobs meet Property 1. Therefore, the structure of large jobs is not so beautiful as that in the purely small jobs case.

Theorem 7. *For any schedule that meets Properties 5 and 6, we can require that:*

Property 9 (*Little Gap*). *$\mathcal{B}_1$ and $\mathcal{B}_2$ are any two adjacent blocks, $c(\mathcal{B}_1) < s(\mathcal{B}_2)$. Suppose that B is the last small batch processed before $\mathcal{B}_1$ (if it does exist), then $s(\mathcal{B}_2) - c(\mathcal{B}_1) < p(B)$.* □

The inequality in the above theorem means that the gap between $\mathcal{B}_1$ and $\mathcal{B}_2$ is smaller than the processing time of B. Since B is a small batch, we know that this gap is smaller than ϵM. The proof is trivial, because if this gap is greater than or equal to $p(B)$, we can process B right there. We remark that if the gap is too narrow to process B but wide enough to process some other small batch that is before B, we shall still let it idle in our algorithm.

4.2. Scheduling large jobs

Suppose that there are l large jobs in total: $\mathcal{L} = \{J_1, J_2, \ldots, J_l\}$, and they are indexed in the non-increasing order of processing times, that is

$$p_1 \ge p_2 \ge \cdots \ge p_l. \tag{27}$$

Remark 3. It is valuable to notice that the way of indexing in this subsection is different from that in the last section. This is why we need two versions of FBLPT.

The basic idea of this subsection is from [8]. Before describing our algorithm, we still need some notations. For any schedule π of large jobs that meets Property 8, we give three notations c_i, n_i and b_i for every set $\mathcal{L}_i$, $1 \le i \le m$. Remember that $\mathcal{L}_i$ is the set of accepted large jobs that start processing during $[r^i, r^{i+1})$. c_i is the total processing time of the batches in $\mathcal{L}_i$, or the length of the time interval that $\mathcal{L}_i$ occupies; n_i is the number of jobs of the last batch in $\mathcal{L}_i$. If the last batch is full or the set $\mathcal{L}_i$ is empty, we let $n_i = 0$; b_i is the delay time of $\mathcal{L}_i$, that is, the start time of $\mathcal{L}_i$ minus r^i. Therefore, the start time of $\mathcal{L}_i$ is $r^i + b_i$. Let $\mathbf{B}$, $\mathbf{C}$, $\mathbf{N}$ be three m-dimensional vectors whose i's components are b_i, c_i and n_i, respectively. $(\mathbf{B}, \mathbf{C}, \mathbf{N})$ is called the *state* of π, and is *feasible* if $r^i + b_i + c_i \le r^{i+1} + b_{i+1}$ holds for all $1 \le i \le m-1$. Our algorithm for large jobs considers all the feasible states. It's not hard to see that we can require $b_i \le p_{\max}$ and $c_i \le \min\{p_{\text{sum}}, r_{\max} + d\}$ for all $1 \le i \le m$.

For any schedule π for all the large jobs, we define $f(\pi)$ as the total penalty of rejected large jobs. Similarly, for any subschedule π_j for jobs $J_1, J_2, \ldots, J_j$, $1 \le j \le l$, which accepts a subset of these jobs, batches the accepted jobs, and determines the start time for each batch, $f(\pi_j)$ is the total penalty of jobs that are rejected in π_j. Notice that the penalties of $J_{j+1}, \ldots, J_l$ don't contribute to $f(\pi_j)$. The state concept in the above paragraph also applies to subschedules in the natural way.

For any $1 \le j \le l$, let $f_j(\mathbf{B}, \mathbf{C}, \mathbf{N}) = \min\{f(\pi_j) : \pi_j \text{ is a subschedule for } J_1, J_2, \ldots, J_j \text{ , whose state is } (\mathbf{B}, \mathbf{C}, \mathbf{N})\}$. We also define $f_j(k, \mathbf{B}, \mathbf{C}, \mathbf{N})$ similarly ($1 \le k \le m, r^k \ge r_j$), except that job J_j should either be rejected or have a start time in $[r^k, r^{k+1})$.

It is trivial that

$$f_j(\mathbf{B}, \mathbf{C}, \mathbf{N}) = \min\{f_j(k, \mathbf{B}, \mathbf{C}, \mathbf{N}) : r^k \ge r_j\}. \tag{28}$$

We discuss in three cases the recursive relationship of $f_j(k, \mathbf{B}, \mathbf{C}, \mathbf{N})$.

Case 1. $c_k < p_j$. In this case, J_j can only be rejected, and therefore

$$f_j(k, \mathbf{B}, \mathbf{C}, \mathbf{N}) = f_{j-1}(\mathbf{B}, \mathbf{C}, \mathbf{N}) + e_j. \tag{29}$$

Case 2. $c_k \ge p_j$ and $n_k \ne 1$. In this case, J_j can be either accepted or rejected.

• If J_j is rejected, it is trivial that $f_j(k, \mathbf{B}, \mathbf{C}, \mathbf{N}) = f_{j-1}(\mathbf{B}, \mathbf{C}, \mathbf{N}) + e_j$.

• If J_j is accepted, then it should be scheduled in the interval $[r^k, r^{k+1})$. $n_k \ne 1$ means that there are still other jobs that are scheduled in the same batch with J_j. By the indexing of large jobs, we know that J_j has the smallest processing

time among the accepted jobs in any subschedule for $J_1, J_2, \ldots, J_j$. Therefore, the state immediately before J_j's joining is $(\mathbf{B}, \mathbf{C}, \mathbf{N}_k)$, where $\mathbf{N}_k$ equals $\mathbf{N}$ except its k-th component becomes $n_k - 1 \ mod\ b$, i.e. $n_k - 1$ if $n_k > 1$, and $b - 1$ if $n_k = 0$. Remember that $n_k = 0$ in this case means that the last batch scheduled in $[r^k, r^{k+1})$ is full. Hence in this situation we have $f_j(k, \mathbf{B}, \mathbf{C}, \mathbf{N}) = f_{j-1}(\mathbf{B}, \mathbf{C}, \mathbf{N}_k)$.

Therefore, in this case we have

$$f_j(k, \mathbf{B}, \mathbf{C}, \mathbf{N}) = \min\{f_{j-1}(\mathbf{B}, \mathbf{C}, \mathbf{N}) + e_j, f_{j-1}(\mathbf{B}, \mathbf{C}, \mathbf{N}_k)\}. \tag{30}$$

Case 3. $c_k \geq p_j$ and $n_k = 1$. The only difference from Case 2 is that J_j alone forms a batch in any subschedule for $J_1, J_2, \ldots, J_j$ whose state is $(\mathbf{B}, \mathbf{C}, \mathbf{N})$. Therefore, compared with $(\mathbf{B}, \mathbf{C}, \mathbf{N})$, the second matrix of the previous state has a k-th component smaller by p_j. Let $\mathbf{C}_{kj}$ be the same as $\mathbf{C}$ except that its kth component becomes $c_k - p_j$, we have in this case

$$f_j(k, \mathbf{B}, \mathbf{C}, \mathbf{N}) = \min\{f_{j-1}(\mathbf{B}, \mathbf{C}, \mathbf{N}) + e_j, f_{j-1}(\mathbf{B}, \mathbf{C}_{kj}, \mathbf{N}_k)\}, \tag{31}$$

where $\mathbf{N}_k$ is the same as defined in Case 2.

To summarize, the overall recursive equations are

Algorithm DP2

$$f_j(k, \mathbf{B}, \mathbf{C}, \mathbf{N}) = \begin{cases} f_{j-1}(\mathbf{B}, \mathbf{C}, \mathbf{N}) + e_j & \text{if } c_k < p_j \\ \min\{f_{j-1}(\mathbf{B}, \mathbf{C}, \mathbf{N}) + e_j, f_{j-1}(\mathbf{B}, \mathbf{C}, \mathbf{N}_k)\} & \text{if } c_k \geq p_j \ \&\ n_k \neq 1 \\ \min\{f_{j-1}(\mathbf{B}, \mathbf{C}, \mathbf{N}) + e_j, f_{j-1}(\mathbf{B}, \mathbf{C}_{kj}, \mathbf{N}_k)\} & \text{if } c_k \geq p_j \ \&\ n_k = 1 \end{cases}$$

$$f_j(\mathbf{B}, \mathbf{C}, \mathbf{N}) = \min\{f_j(k, \mathbf{B}, \mathbf{C}, \mathbf{N}) : r^k \geq r_j\}.$$

The computations are done over all feasible states and over all $2 \leq j \leq l$. The boundary conditions are presented in three cases as follows.

- If there exists some $1 \leq i \leq m$ such that $r^i + b_i \geq r_1$, $c_i = p_1$, $n_i = 1$, and the other components of $\mathbf{C}$ and $\mathbf{N}$ are all 0, then $f_1(\mathbf{B}, \mathbf{C}, \mathbf{N}) = 0$.
- If for all $1 \leq i \leq m$, $n_i = c_i = 0$, then $f_1(\mathbf{B}, \mathbf{C}, \mathbf{N}) = e_1$.
- Otherwise, $f_1(\mathbf{B}, \mathbf{C}, \mathbf{N}) = \infty$.

For any feasible state $(\mathbf{B}, \mathbf{C}, \mathbf{N})$, let

$$D = \bigcup_{1 \leq i \leq m} (r^i + b_i, r^i + b_i + c_i], \tag{32}$$

that is, D is the union of time intervals during which some large batch is processed. D will also be referred to as a *concise state*. Obviously, each feasible state determines a unique concise state. We also let $h(D) = r^m + b_m + c_m$ and

$$f(D) = \min\{f_l(\mathbf{B}, \mathbf{C}, \mathbf{N}) : (\mathbf{B}, \mathbf{C}, \mathbf{N}) \text{ is feasible and determines } D\}. \tag{33}$$

We call D *feasible* if $f(D) \neq \infty$. For the set of large jobs $\mathcal{L}$, we denote by $\mathcal{D}(\mathcal{L})$ the set of all its feasible concise states.

For any schedule π, by D^π we denote the concise state that is determined by π. Obviously,

$$f(D^\pi) \leq f(\pi). \tag{34}$$

For any concise state D, we denote by π^D a schedule whose concise state is D, and whose total penalty of rejected jobs is $f(D)$, then

$$f(\pi^D) = f(D), \tag{35}$$

$$D = D^{\pi^D}. \tag{36}$$

Due to Theorem 4, we require that no small jobs are processed in D. Notice that different states may lead to the same concise state D, therefore we might expect the cardinality of the set $\mathcal{D}(\mathcal{L})$ to be small. However, it is still huge. And the running time of DP2 can be as large as $O(ml(bp_{\max} \min\{p_{\text{sum}}, r_{\max} + d\})^m)$, because there are $O((bp_{\max} \min\{p_{\text{sum}}, r_{\max} + d\})^m)$ feasible states and the running time for each state is $O(ml)$. To reduce both the cardinality of $\mathcal{D}(\mathcal{L})$ and the running time, we need identical rounding again.

Algorithm DP2′

STEP 0. Input the set of large jobs $\mathcal{L}$;

STEP 1. Construct a rounded-down set $\mathcal{L}'$ by letting $p'_j := \lfloor \frac{p_j}{M/(n+1)} \rfloor M/(n+1)$, $r'_j := \lfloor \frac{r_j}{M/(n+1)} \rfloor M/(n+1)$, and $e'_j := e_j$, for al $1 \leq j \leq l$;

STEP 2. Run Algorithm DP2 on the new instance $\mathcal{L}'$, where $M/(n+1)$ is taken as a unit;

STEP 3. Compute $\mathcal{D}(\mathcal{L}')$, the set of feasible concise states of $\mathcal{L}'$;

For all $D' \in \mathcal{D}(\mathcal{L}')$, compute $f(D')$ and $\pi^{D'}$;

STEP 4. For any $D' \in \mathcal{D}(\mathcal{L}')$, let π, a schedule of $\mathcal{L}$, be the counterpart of $\pi^{D'}$. Compute the set of D^{π}'s, which is denoted as $\mathcal{D}^*(\mathcal{L})$.

In the last step, it is easy to see that

$$f(D^{\pi}) \le f(\pi) = f(\pi^{D'}) = f(D'). \tag{37}$$

Remark 4. Since there is a natural bijection between $\mathcal{L}$ and $\mathcal{L}'$, i.e. the one that corresponds a member of $\mathcal{L}$ to its rounded-down counterpart in $\mathcal{L}'$, and a schedule is completely determined by the set of jobs it accepts, the batching and sequencing of these accepted jobs, the word *counterpart* in Step 4 of the above algorithm has a definite meaning. We will use this word again later.

Remark 5. It is valuable to notice that we do not calculate $f(D)$, for $D \in \mathcal{D}^*(\mathcal{L})$ in Step 4, since it is still not easy. In contrast, $f(D')$ can be calculated in polynomial time.

Since

$$\left\lfloor \frac{p_j}{M/(n+1)} \right\rfloor \le \frac{p_{\max}}{\epsilon \max\{r_{\max}, d\}/(n+1)} \le 2(n+1)/\epsilon, \tag{38}$$

where the second inequality is from Assumption 2, the running time of Step 2 in Algorithm DP2′ can be bounded by $O(ml(4bln^2/\epsilon^2)^m)$, which is a polynomial of n, since $m \le 1/\epsilon$. Accordingly, the cardinality of $\mathcal{D}(\mathcal{L}')$ as well as $\mathcal{D}^*(\mathcal{L})$ can be bounded by $O((4bln^2/\epsilon^2)^m)$. In Step 3, for each $(\mathbf{B}', \mathbf{C}', \mathbf{N}')$, a feasible state of $\mathcal{L}'$, the corresponding concise state D' can be calculated in $O(l)$ time; computing all the $f(D')$'s as well as all the $\pi^{D'}$'s requires a time of $O((4bln^2/\epsilon^2)^m)$, the number of $f_l(\mathbf{B}', \mathbf{C}', \mathbf{N}')$'s. So the running time of Step 3 is bounded by $O(l(4bln^2/\epsilon^2)^m)$. Since in Step 4, for each $D' \in \mathcal{D}(\mathcal{L}')$, the corresponding π and D^{π} can be easily computed in $O(l)$, and Step 1 can be trivially done in $O(l)$ time, the time complexity of Algorithm DP2′ is $O(ml(4bln^2/\epsilon^2)^m)$, which is determined by Step 2.

Next, we shall show that $\mathcal{D}^*(\mathcal{L})$ approximates $\mathcal{D}(\mathcal{L})$ very well.

Theorem 8. *For any $D = \bigcup_{1\le i\le m}(r^i + b_i, r^i + b_i + c_i^0] \in \mathcal{D}(\mathcal{L})$, there exists a $D^0 = \bigcup_{1\le i\le m}(r^i + b_i^0, r^i + b_i^0 + c_i^0] \in \mathcal{D}^*(\mathcal{L})$, such that $f(D^0) \le f(D)$, $h(D^0) - h(D) \le M$ and $\sum_{i=1}^{m-1} \max\{c_i^0 - c_i, 0\} \le M$.*

Proof. For any $D \in \mathcal{D}(\mathcal{L})$, let π', a schedule of $\mathcal{L}'$, be the counterpart of π^D. Then

$$f(D^{\pi'}) \le f(\pi') = f(\pi^D) = f(D), \tag{39}$$

and $D^{\pi'} \in \mathcal{D}(\mathcal{L}')$. Let $D^0 \in \mathcal{D}^*(\mathcal{L})$ be the concise state generated by $D^{\pi'}$ in Step 4 of DP2′. Therefore,

$$f(D^0) \le f(D^{\pi'}) \le f(D), \tag{40}$$

where the former part is from (37) and the latter is from (39).

Let π^0 be the corresponding schedule of D^0 in Step 4 of Algorithm DP2′, i.e. $D^0 = D^{\pi^0}$. Compared with its counterpart in π', each accepted batch in π^0 may have a larger processing time and a larger release time. However, the differences can be bounded by $M/(n+1)$. For all $1 \le i \le m$, let α_i be the number of batches in π^0 processed in the time interval $(r^i + b_i^0, r^i + b_i^0 + c_i^0]$, then

$$0 \le c_i^0 - c_i' \le \alpha_i M/(n+1). \tag{41}$$

Since $c_i \ge c_i'$, we have

$$\max\{c_i^0 - c_i, 0\} \le \alpha_i M/(n+1), \tag{42}$$

and therefore

$$\sum_{i=1}^{m-1} \max\{c_i^0 - c_i, 0\} \le \left(\sum_{i=1}^{m-1} \alpha_i\right) M/(n+1) \le M. \tag{43}$$

In addition to processing times, the differences of release times may also affect the difference between $h(D^0)$ and $h(D^{\pi'})$. However, unlike the processing times, the effect of the augmenting of release times is not cumulative. Therefore,

$$h(D^0) - h(D^{\pi'}) \le (n+1)M/(n+1) = M. \tag{44}$$

Together with $h(D^{\pi'}) \le h(D)$, we get the whole theorem. □

4.3. Inserting small jobs

Given any feasible concise state $D \in \mathcal{D}^*(\mathcal{L})$, we shall show in this subsection how to insert the small jobs. From Theorem 5 we know that small jobs have a nice structure. The only problem is to decide the set of accepted small jobs. To tackle this, we design a dynamic programming that is similar to DP1.

Suppose that the set of small jobs is $\mathcal{S}$, and there are s small jobs in total. Obviously, $s + l = n$. Without causing any confusion, we call these s jobs $J_1, J_2, \ldots, J_s$, and we assume that they are indexed in the nondecreasing order of their release times, that is

$$r_1 \le r_2 \le \cdots \le r_s. \tag{45}$$

For the jobs with the same release times, they are indexed in the nondecreasing order of their processing times. The notations r^i, n^i, $1 \le i \le m$, and u_j, $1 \le j \le s$, as defined in Section 3, are also used.

Remark 6. Notice that this way of indexing is the same as we do in Section 3 when we deal with the purely small jobs special case, but different from what we do in the last subsection on large jobs.

For any $D \in \mathcal{D}^*(\mathcal{L})$ and a positive number p, $p \notin D$, let $c(p, D)$ be the right most point in D that is smaller than p. That is, $c(p, D)$ is the right point of the interval in D which is nearest on the left of p. Let $s(p, D)$ be the left point of the corresponding interval. If there is no interval in D on the left of p, we simply let $c(p, D) = s(p, D) = 0$.

Given $D \in \mathcal{D}^*(\mathcal{L})$, let $f_1^D(j, p)$ denote the smallest total penalty among all the subschedules of small jobs $J_1, J_2, \ldots, J_j$ with completion time at most p, while J_j is accepted, $1 \le j \le s$. Notice that subschedules here $f_0^D(j, p)$ means the similar thing except that J_j is rejected. It is obvious that

$$f_0^D(j, p) = \min\{f_0^D(j-1, p), f_1^D(j-1, p)\}. \tag{46}$$

Next, we shall discuss in four cases the recursive relationship of $f_1^D(j, p)$.

Case 1. $p \in D$ or $p - p_j < r_j$. Obviously in this case $f_1^D(j, p) = \infty$.

Case 2. $p \notin D$ and $r_j \le p - p_j < c(p, D)$. $p - p_j < c(p, D)$ means that the gap between $c(p, D)$ and p is too narrow to process J_j. Since J_j is accepted, we need to consider process it in the immediate preceding gap. Therefore,

$$f_1^D(j, p) = f_1^D(j, s(p, D)). \tag{47}$$

Case 3. $p \notin D$, $p - p_j \ge \max\{r_j, c(p, D)\}$ and $j - n_{u_j-1} \le b$. $p \notin D$ and $p - p_j \ge \max\{r_j, c(p, D)\}$ mean that the gap between $c(p, D)$ and p is wide enough to process J_j, by Theorems 5 and 7, we process it right here. $j - n_{u_j-1} \le b$ means that there are less than b jobs arriving at r^{u_j}. Since J_j has the largest processing time among them, we have to accept all of them. Therefore,

$$f_1^D(j, p) = \min\{f_0^D(n_{u_j-1}, p - p_j), f_1^D(n_{u_j-1}, p - p_j)\}. \tag{48}$$

Case 4. $p \notin D$, $p - p_j \ge \max\{r_j, c(p, D)\}$ and $j - n_{u_j-1} > b$. The only difference between this case and Case 3 is that we can't definitely accept all the jobs arriving at r^{u_j}. For any $b - 1 \le k < j - n_{u_j-1}$, denote

$$\Delta_0^D(p, j, k) = g(j-k, j-1) + f_0^D(j-k-1, p - p_j), \tag{49}$$

and

$$\Delta_1^D(p, j, k) = g(j-k, j-1) + f_1^D(j-k-1, p - p_j), \tag{50}$$

similar to Case 3 in the analysis of DP1, we have the following relationship:

$$f_1^D(j, p) = \min\left\{\Delta_0^D(p, j, j - n_{u_j-1} - 1),\ \Delta_1^D(p, j, j - n_{u_j-1} - 1),\ \min\{\Delta_1^D(p, j, k) : b - 1 \le k < j - n_{u_j-1} - 1\}\right\}. \tag{51}$$

In sum, the overall recursive equations are presented as follows, where the concise state D is given as a parameter.

Algorithm DP3(D)

$$f_0^D(j, p) = \min\left\{f_0^D(j-1, p), f_1^D(j-1, p)\right\};$$

$$f_1^D(j, p) = \begin{cases} \infty & \text{if } p \in D \text{ or } p - p_j < r_j \\ f_1^D(j, s(p, D)) & \text{if } p \notin D \,\&\, r_j \le p - p_j < c(p, D) \\ \min\{f_0^D(n_{u_j-1}, p - p_j), f_1^D(n_{u_j-1}, p - p_j)\} & \text{if } p \notin D \,\&\, p - p_j \ge \max\{r_j, c(p, D)\} \\ & \&\, j - n_{u_j-1} \le b \\ \min\left\{\begin{array}{l}\Delta_0^D(p, j, j - n_{u_j-1} - 1),\ \Delta_1^D(p, j, j - n_{u_j-1} - 1), \\ \min\{\Delta_1^D(p, j, k) : b - 1 \le k < j - n_{u_j-1} - 1\}\end{array}\right\} & \text{if } p \notin D \,\&\, p - p_j \ge \max\{r_j, c(p, D)\} \\ & \&\, j - n_{u_j-1} > b \end{cases}$$

The computations are done over all $2 \le j \le s$, $h(D) \le p \le (1+4\epsilon)(r_{\max}+d)$, where the coefficient $1+4\epsilon$ is from Lemma 4 and Theorems 3–5, for the same reason as we argued in Remark 1. The boundary conditions are:

$$f_0^D(1,p) = e_1, \quad \text{for all } h(D) \le p \le (1+4\epsilon)(r_{\max}+d) \,\&\, p \notin D; \tag{52}$$

$$f_1^D(1,p) = \begin{cases} \infty & \text{if } p \in D \text{ or } p - p_1 < r_1 \\ f_1^D(1, s(p,D)) & \text{if } p \notin D \,\&\, r_1 \le p - p_1 < c(p,D) \\ 0 & \text{if } p \notin D \,\&\, p - p_1 \ge \max\{r_1, c(p,D)\}. \end{cases} \tag{53}$$

In the end of the algorithm, compute

$$g(D) = \min\{f_0^D(s,p) + p, f_1^D(s,p) + p : h(D) \le p \le (1+4\epsilon)(r_{\max}+d)\}. \tag{54}$$

Remark 7. For any schedule π, its objective value is divided into three parts: the total penalty of rejected large jobs, the total penalty of rejected small jobs, and makespan, where the first part corresponds to $f(D)$, the second part plus the third part correspond to $g(D)$.

It is easy to calculate that the running time for each $D \in \mathcal{D}^*(\mathcal{L})$ is $O(s^2(r_{\max}+d))$ (the time for computing all the $g(i,j)$'s is not accounted, as $g(i,j)$'s will be computed initially in the final algorithm). To reduce the complexity to a polynomial, we need to modify this algorithm similarly as we do in DP1′. For any real number x, by x' we still denote $\left\lfloor \frac{x}{M/n} \right\rfloor (M/n)$.

The next dynamic programming still takes $D \in \mathcal{D}^*(\mathcal{L})$ as a parameter.

Algorithm DP3′(D)

$$f_0'^D(j,q) = \min\left\{f_0'^D(j-1,q), f_1'^D(j-1,q)\right\};$$

$$f_1'^D(j,q) = \begin{cases} \infty & \text{if } q \in D \text{ or } q - p_j < r_j \\ f_1'(j, (s(q,D))') & \text{if } q \notin D \,\&\, r_j \le q - p_j < c(q,D) \\ \min\{f_0'^D(n_{u_j-1}, (q-p_j)'), f_1'^D(n_{u_j-1}, (q-p_j)')\} & \text{if } q \notin D \,\&\, q - p_j \ge \max\{r_j, c(q,D)\} \\ & \,\&\, j - n_{u_j-1} \le b \\ \min\left\{\begin{matrix} \Delta_0'^D(q,j,j-n_{u_j-1}-1), \Delta_1'^D(q,j,j-n_{u_j-1}-1), \\ \min\{\Delta_1'^D(q,j,k) : b-1 \le k < j - n_{u_j-1} - 1\} \end{matrix}\right\} & \text{if } q \notin D \,\&\, p - p_j \ge \max\{r_j, c(q,D)\} \\ & \,\&\, j - n_{u_j-1} > b \end{cases}$$

where

$$\Delta_0'^D(q,j,k) = g(j-k, j-1) + f_0'^D(j-k-1, (q-p_j)'), \tag{55}$$

$$\Delta_1'^D(q,j,k) = g(j-k, j-1) + f_1'^D(j-k-1, (q-p_j)'), \tag{56}$$

and the computations are done over all $2 \le j \le s$, and over all $q \in Q_1 = \{M/n, 2M/n, \ldots, t_1 M/n\}$,

$$t_1 = \left\lceil \frac{(1+10\epsilon)(r_{\max}+d)}{M/n} \right\rceil, \tag{57}$$

where the coefficient $1+10\epsilon$ is from the proof of Theorem 10 in the next subsection. The boundary conditions are:

$$f_0'^D(1,q) = e_1, \quad \text{for all } q \in Q_1 \text{ and } q \notin D; \tag{58}$$

$$f_1'^D(1,q) = \begin{cases} \infty & \text{if } q \in D \text{ or } q - p_1 < r_1 \\ f_1'^D(1, (s(q,D))') & \text{if } q \notin D \,\&\, r_1 \le q - p_1 < c(q,D) \\ 0 & \text{if } q \notin D \,\&\, q - p_1 \ge \max\{r_1, c(q,D)\}. \end{cases} \tag{59}$$

In the end of DP3′(D), compute

$$g'(D) = \min\{f_0'^D(s,q) + q, f_1'^D(s,q) + q : q \in Q_1 \text{ and } q \ge h(D)\}. \tag{60}$$

Since

$$t_1 \le \frac{(1+10\epsilon)(r_{\max}+d)}{M/n} + 1 \tag{61}$$

$$= \frac{(1+10\epsilon)(r_{\max}+d)}{\epsilon \max\{r_{\max}, d\}/n} + 1 \tag{62}$$

$$\le 2(1+10\epsilon)(1/\epsilon)n + 1, \tag{63}$$

the running time of DP3′(D) for each $D \in \mathcal{D}^*(\mathcal{L})$ can be bounded by $O((1/\epsilon)s^2 n)$, which is a polynomial.

Theorem 9. *For any $D \in \mathcal{D}^*(\mathcal{L})$, $g'(D) \le g(D) + 3M$.*

Proof. For any $D \in \mathcal{D}^*(\mathcal{L})$, suppose that π is the schedule we get in DP3(D) which attains the value $g(D)$, i.e. the total penalty of rejected small jobs in π plus its makespan is $g(D)$. By pushing the small batches in π forward one by one from the beginning such that all their completion times are integral times of M/n, we construct a schedule π'.

Notice in this construction that we have to evade all the blocks of large jobs defined by D: Whenever the moving forward of a small batch is stopped by a block, we jump the block and process the batch in the beginning of the nearest gap; if the completion time is not integral times of M/n, we move the batch forward again, and so on until we find a suitable position for the batch.

Suppose there are i_0 small batches in total: $B_1, B_2, \ldots, B_{i_0}$, and B_i makes v_i jumps, $1 \le i \le i_0$, then it makes at most $v_i + 1$ moves. Notice that we distinguished *jumps* from *moves*. Since each block can be jumped at most once, and the first block will never be jumped, we have $\sum_{1\le i\le i_0} v_i \le m - 1$. Since each jump causes a postponement in the makespan of at most ϵM, and each move causes M/n, we have

$$C_{\max}(\pi') - C_{\max}(\pi) \le \sum_{1\le i\le i_0} (v_i\epsilon M + (v_i + 1)M/n) \tag{64}$$

$$= \left(\sum_{1\le i\le i_0} v_i\right)\epsilon M + (i_0 + \sum_{1\le i\le i_0} v_i)M/n \tag{65}$$

$$\le (m-1)\epsilon M + (n + m - 1)M/n \tag{66}$$

$$\le 3M. \tag{67}$$

Therefore,

$$g'(D) - g(D) \le C_{\max}(\pi') - C_{\max}(\pi) \le 3M, \tag{68}$$

which completes the proof. □

4.4. The final PTAS

For ease of the presentation, we need still another notation. For any $D \in \mathcal{D}^*(\mathcal{L})$, let D' be the concise state of $\mathcal{L}'$ that generates D in Step 4 of DP2′, we define

$$f'(D) = f(D'). \tag{69}$$

Algorithm SLIS

STEP 1. Calculate d and M, specify the set of large jobs $\mathcal{L}$ and the set of small jobs $\mathcal{S}$, re-index the large jobs and small jobs separately as we do in Section 4.2 and Section 4.3;

STEP 2. Calculate all the $g(i, j)$'s for $\mathcal{S}$;

STEP 3. Call Algorithm DP2′ to compute $\mathcal{D}^*(\mathcal{L})$;

STEP 4. For each $D \in \mathcal{D}^*(\mathcal{L})$, call Algorithm DP3′(D) to calculate $g'(D)$;

STEP 5. Find a $D^* \in \mathcal{D}^*(\mathcal{L})$ such that $f'(D^*) + g'(D^*)$ is minimized; Output the corresponding schedule π^* such that $m(\pi^*) = f'(D^*) + g'(D^*)$.

Theorem 10. *Algorithm SLIS is a PTAS for* $1|p\text{-}batch, rej, r_j|C_{\max} + \mathrm{TP}$.

Proof. By the properties in Section 4.1, if we replace DP2′ in Step 3 with DP2, and DP3′ in Step 4 with DP3, we shall get a $1 + 4\epsilon$-approximate solution. Suppose that π^* is the solution we get in SLIS, and π^1 is the schedule we get if we replace DP2′ with DP2 and DP3′ with DP3. For the accuracy, it suffices to show that $m(\pi^*) - m(\pi^1) \le 6\epsilon Opt$, which immediately gives $m(\pi^*) \le (1 + 10\epsilon)Opt$.

By Theorem 8, there is a D^0 generated by DP2′ such that $f(D^0) \le f(D^{\pi^1})$, $h(D^0) - h(D^{\pi^1}) \le M$ and $\sum_{i=1}^{m-1} \max\{c_i^0 - c_i^1, 0\} \le M$, where we suppose

$$D^{\pi^1} = \bigcup_{1\le i\le m} (r^i + b_i^1, r^i + b_i^1 + c_i^1], \tag{70}$$

and

$$D^0 = \bigcup_{1\le i\le m} (r^i + b_i^0, r^i + b_i^0 + c_i^0]. \tag{71}$$

For the ease of notations, we assume later without loss of generality, which will be seen later, that there are $m - 1$ gaps in both D^{π^1} and D^0. That is, $b_i^1 = b_i^0 = 0$ for all $1 \le i \le m$, and $r^i + c_i^1 < r^{i+1}$, $r^i + c_i^0 < r^{i+1}$, for all $1 \le i \le m - 1$. We also let $G_i^1 = (r^i + c_i^1, r^{i+1}]$, $G_i^0 = (r^i + c_i^0, r^{i+1}]$, $1 \le i \le m - 1$, $G_m^1 = (r^m + c_m^1, \infty)$, and $G_m^0 = (r^m + c_m^0, \infty)$.

For $1 \le i \le m$, let $\mathcal{I}_i$ be the set of small batches that are processed by π^1 in the gap G_i^1. We construct a schedule π^0, based on D^0, by accepting the same set of small jobs as in π^1, batching these small jobs also as in π^1, and scheduling these small

batches as follows: For the small batches in $\mathcal{I}_i$, π^0 processes them in G_0^i in any order. If G_0^i is too narrow to hold all of them, process the surplus ones after $h(D^0)$. Intuitively, π^0 is a good approximation to π^1. We will show rigidly in the following that $m(\pi^0) - m(\pi^1) \leq 3M$.

For all $1 \leq i \leq m$, let x_i^1 be the total processing time of small batches of π^1 that are processed in G_i^1. Analogously, x_i^0 is the total processing time of small batches of π^0 that are processed in G_i^0.

By the construction of π^0, we know that for all $1 \leq i \leq m-1$, the total processing time of the surplus small batches in $\mathcal{I}_i$, which are scheduled after $h(D^0)$, is

$$x_i^1 - x_i^0 \leq \max\{c_i^0 - c_i^1, 0\} + \epsilon M. \tag{72}$$

From

$$\sum_{i=1}^{m} x_i^0 = \sum_{i=1}^{m} x_i^1, \tag{73}$$

we have

$$\begin{aligned}
x_m^0 - x_m^1 &= \sum_{i=1}^{m-1} (x_i^1 - x_i^0) && (74)\\
&\leq \sum_{i=1}^{m-1} (\max\{c_i^0 - c_i^1, 0\} + \epsilon M) && (75)\\
&\leq \sum_{i=1}^{m-1} \max\{c_i^0 - c_i^1, 0\} + (m-1)\epsilon M && (76)\\
&\leq M + M && (77)\\
&= 2M. && (78)
\end{aligned}$$

Since π^0 and π^1 accept the same set of small jobs,

$$\begin{aligned}
m(\pi^0) - m(\pi^1) &= (f(D^0) - f(D^{\pi^1}) + (C_{\max}(\pi^0) - C_{\max}(\pi^1)) && (79)\\
&\leq C_{\max}(\pi^0) - C_{\max}(\pi^1) && (80)\\
&= (h(D^0) + x_m^0) - (h(D^{\pi^1}) + x_m^1) && (81)\\
&= (h(D^0) - h(D^{\pi^1})) + (x_m^0 - x_m^1) && (82)\\
&\leq M + 2M && (83)\\
&= 3M. && (84)
\end{aligned}$$

Let $\pi^{0'}$ be the counterpart of π^0 in the sense of Algorithm DP2′, then $D^{\pi^{0'}} = D^{0'}$ (remember that $D^{0'}$ is defined as the concise state that generates D^0 in Step 4 of Algorithm DP2′), and therefore

$$f(D^{0'}) \leq f(\pi^{0'}) \quad \text{(due to (34))}. \tag{85}$$

Finally we have

$$\begin{aligned}
& m(\pi^*) - m(\pi^1) && (86)\\
&= f'(D^*) + g'(D^*) - m(\pi^1) && (87)\\
&\leq f'(D^0) + g'(D^0) - m(\pi^1) \quad \text{(definition of } \pi^*) && (88)\\
&= f(D^{0'}) + g'(D^0) - m(\pi^1) \quad \text{(due to (69))} && (89)\\
&\leq f(\pi^{0'}) + g'(D^0) - m(\pi^1) \quad \text{(due to (85))} && (90)\\
&= f(\pi^0) + g'(D^0) - m(\pi^1) && (91)\\
&\leq f(\pi^0) + (g(D^0) + 3M) - m(\pi^1) \quad \text{(due to Theorem 9)} && (92)\\
&\leq (m(\pi^0) - m(\pi^1)) + 3M \quad (m(\pi^0) = f(\pi^0) + g(D^0)) && (93)\\
&\leq 6M \quad \text{(due to (84))} && (94)\\
&\leq 6\epsilon Opt. && (95)
\end{aligned}$$

In addition, the main running time, which is occupied mainly in Step 4, can be bounded by

$$O(ml(4bln^2/\epsilon^2)^m) \times O((1/\epsilon)s^2 n) = O((4b)^{1/\epsilon} n^{3/\epsilon+4} (1/\epsilon)^{2/\epsilon+2}), \tag{96}$$

which is a polynomial in n. This completes the whole proof. □

5. Concluding remarks

In this paper, a combined model of parallel batch scheduling and scheduling with rejection, $1|p\text{-}batch, rej, r_j|C_{\max} + \mathrm{TP}$, is attacked by presenting a PTAS. The main procedure of our algorithm is to schedule the large jobs first and then insert the small jobs in gaps between large batches. The main running time comes from the large jobs, as we can schedule the small ones according to a beautiful structure, while the price of an error in precision which can be as small as possible is paid. It is not hard to notice that our algorithm on large jobs is very inefficient, just a little better than merely enumerating. Therefore, more efficient algorithms for large jobs are meaningful. Another meaningful direction for further research is to develop nice properties for penalties, since all our assumptions and properties are aimed at processing times. A straightforward observation is that if $e_j \geq p_j$, then J_j must be accepted.

Acknowledgments

The authors would like to thank the anonymous referees for their many helpful suggestions on presentation of this paper.

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] J.J. Bartholdi, unpublished manuscript, 1988.
[3] 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 (1) (1998) 31–54.
[4] Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall, L. Stougie, Multiprocessor scheduling with rejection, SIAM Journal of Discrete Maths 13 (1) (2000) 64–78.
[5] Z. Cao, Z. Wang, Y. Zhang, S. Liu, On several scheduling problems with rejection or discretely compressible processing times, Lecture Notes in Computer Science 3959 (2006) 90–98.
[6] Z. Cao, Y. Zhang, Scheduling with rejection and non-identical job arrivals, Journal of Systems Science and Complexity 20 (4) (2007) 529–535.
[7] X. Deng, H. Feng, G. Li, B. Shi, A PTAS for semiconductor burn-in scheduling, Journal of Combinatorial Optimization 9 (2005) 5–17.
[8] X. Deng, C.K. Poon, Y. Zhang, Approximation algorithms in batch processing, Journal of Combinational Optimization 7 (3) (2003) 247–257.
[9] D.W. Engels, D.R. Karger, S.G. Kolliopoulos, S. Sengupta, R.N. Uma, J. Wein, Techniques for scheduling with rejection, in: Lecture Notes in Computer Science, vol. 1461, 1998, pp. 490–501.
[10] L. Epstein, J. Noga, G.J. Woeginger, On-line scheduling of unit time jobs with rejection: Minimizing the total completion time, Operations Research Letters 30 (6) (2002) 415–420.
[11] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling, Annals of Discrete Mathematics 5 (1979) 287–326.
[12] Y. He, X. Min, On-line uniform machine scheduling with rejection, Computing 65 (1) (2000) 1–12.
[13] H. Hoogeveen, M. Skutella, G.J. Woeginger, Preemptive scheduling with rejection, Mathematical Programming Serial B94 (2003) 361–374.
[14] Y. Ikura, M. Gimple, Scheduling algorithms for a single batch processing machine, Operations Research Letters 5 (1986) 61–65.
[15] S. Li, G. Li, X. Wang, Q. Liu, Minimizing makespan on a single machine with release times and non-identical job sizes, Operations Research Letter 33 (2) (2005) 157–164.
[16] S. Li, G. Li, S. Zhang, Minimizing makespan with release times on identical parallel batching machines, Discrete Applied Mathematics 148 (1) (2005) 127–134.
[17] C.Y. Lee, R. Uzsoy, Minimizing makespan on a single batch processing machine with dynamic job arrivals, International Journal of Production Research 37 (1999) 219–236.
[18] C.N. Potts, M.Y. Lovalyov, Scheduling with batching: A review, European Journal of Operational Research 120 (2000) 228–249.
[19] C.K. Poon, P. Zhang, Minimizing makespan in batch machine scheduling, Algorithmica 39 (2) (2004) 155–174.
[20] S.S. Seiden, Preemptive multiprocessor scheduling with rejection, Theoretical Computer Science 262 (1) (2001) 437–458.
[21] S. Sengupta, Algorithms and approximation schemes for minimum lateness/tardiness scheduling with rejection, in: Lecture Notes in Computer Science, vol. 2748, 2003, pp. 79–90.
[22] Z. Wang, Z. Cao, Y. Zhang, Single machine batch scheduling with rejection to minimize makespan, Journal of Qufu Normal University 33 (2) (2007) 35–38. in Chinese.
[23] Y. Zhang, Z. Cao, Q. Bai, A PTAS for batch scheduling on agreeable unrelated parallel machines with dynamic job arrivals, in: Lecture Notes in Computer Science, vol. 3521, 2005, pp. 162–171.