Makespan Minimization on Parallel Batch Processing Machines with Release Times and Job Sizes

Shuguang Li
College of Computer Science and Technology, Shandong Institute of Business and Technology, Yantai, China
Email: sgliytu@hotmail.com

***Abstract*—This paper investigates the scheduling problem of minimizing makespan on parallel batch processing machines encountered in different manufacturing environments, such as the burn-in operation in the manufacture of semiconductors and the aging test operation in the manufacture of thin film transistor-liquid crystal displays (TFT-LCDs). Each job is characterized by a processing time, a release time and a job size. Each machine can process multiple jobs simultaneously in a batch, as long as the total size of all jobs in the batch does not exceed machine capacity. The processing time of a batch is represented by the longest time among the jobs in the batch. An approximation algorithm with worst-case ratio $2+\varepsilon$ is presented, where $\varepsilon > 0$ can be made arbitrarily small.**

***Index Terms*—scheduling, parallel batch processing machines, makespan, release times, job sizes, worst-case analysis**

I. INTRODUCTION

Batch processing machines are encountered in many different environments, such as the diffusion and burn-in operations in semiconductor fabrication, heat treatment operations in metalworking, and aging test operations in the manufacture of thin film transistor-liquid crystal displays (TFT-LCDs). In these operations, the machines are usually treated as batch-processing machines that can accommodate several jobs as a batch for processing simultaneously, with the total size of the batch not exceeding machine capacity. Since different batching groups require different available times and processing times, the batching and scheduling of the jobs is highly non-trivial and can greatly affect the production rate.

Many batch scheduling problems are NP-hard, i.e., for many of them there does not exist any polynomial time algorithm unless P = NP. Researchers therefore turn to studying approximation algorithms for these kinds of problems. The quality of an approximation algorithm is often measured by its *worst-case ratio*: the smaller the ratio is, the better the algorithm will be. We say that an algorithm has a worst-case ratio ρ (or is a ρ -approximation algorithm) if for any input instance, it always returns in polynomial time of the input size a feasible solution with an objective value not greater than ρ times of the optimal value. Furthermore, a family of approximation algorithms is called a *polynomial-time approximation scheme* (PTAS) if, for any fixed $\varepsilon > 0$, at least one of the algorithms has a worst-case ratio no more than $1+\varepsilon$.

The scheduling problem considered in this paper is described as follows: There is a set $J = \{1,2,\ldots,n\}$ of n jobs that can be processed on m batch processing machines. Each job, j, is characterized by a triple of real numbers (r_j, p_j, s_j), where r_j is the release time before which job j cannot be scheduled, p_j is the processing time which specifies the minimum time needed to process job j without interruption on any one of the machines, and $s_j \in (0,1]$ is the size of job j. Each batch processing machine has a capacity 1 and can process a number of jobs simultaneously as a batch as long as the total size of jobs in the batch does not exceed 1. The available time and processing time of the batch are represented by the latest release time and longest processing time among the jobs in the batch, respectively. Jobs processed in the same batch have the same completion time (the completion time of the batch in which they are contained), i.e., their common start time (the start time of the batch in which they are contained) plus the processing time of the batch. Once the process begins, it cannot be interrupted until the process is completed. Our goal is to find a schedule for the jobs so that the makespan, defined as the completion time of the last job, is minimized. This model is expressed as $P \mid r_j, s_j, b = 1 \mid C_{\max}$.

Recently, many research efforts have been devoted to scheduling problems concerned with batch processing machines. These problems have either identical or non-identical job size characteristics.

With regard to batch-processing machine scheduling problems with identical job size characteristics, Chandru, Lee, and Uzsoy [1] proposed a branch-and-bound method to minimize total completion time on a single batch-processing machine and presented several heuristics for identical parallel batch-processing machines as well. Lee,

doi:10.4304/jsw.7.6.1203-1210

Uzsoy, and Martin-Vega [2] studied the single batch-processing machine problem and provided dynamic programming-based algorithms to minimize the number of tardy jobs and the maximum tardiness under a number of assumptions. They also provided two heuristic algorithms for the problem of parallel batch-processing machines with makespan criterion. Sung and Choung [3] proposed a branch-and-bound method to minimize the makespan for a single batch-processing machine problem. Lee and Uzsoy [4] presented a number of efficient heuristics to solve the single batch-processing machine problem with unequal release times. In addition, Li et al. [5] extended the study of the single batch-processing machine problem by Lee and Uzsoy [4] to involve an examination of the identical parallel batch processing machines problem and proposed a polynomial time approximation scheme (PTAS). They also obtained the first PTAS for the problem of minimizing maximum lateness on identical parallel batch processing machines [6]. Studies in identical job sizes were also done by Dupont and Ghazvini [7], Qi and Tu [8] and Wang and Uzsoy [9].

With regard to batch-processing machine scheduling problems with non-identical job size characteristics, Uzsoy [10] derived complexity results for makespan and total completion time criteria and provided some heuristics and a branch and bound algorithm for the case of a single batch processing machine. Zhang et al. [11] examined the worst-case performance of the heuristics addressed by Uzsoy [10] for the single machine makespan problem. They also proposed an improved algorithm with a 3/2 worst-case ratio. Li et al. [12] presented a $(2+\varepsilon)$ -approximation algorithm for the single machine problem with release times, where $\varepsilon > 0$ can be made arbitrarily small. Nong et al. [13] studied the problem of scheduling family jobs on a batch processing machine to minimize the makespan and presented an approximation algorithm with a 5/2 worst-case ratio. Dupont and Dhaenens-Flipo [14], on the other hand, presented some dominance properties and proposed a branch-and-bound method to solve the single batch-processing machine scheduling problem with non-identical job sizes. Chung, Tai, and Pearn [15] considered the parallel batch-processing machines with unequal release times and non-identical job sizes, which is motivated by the aging test operation in the manufacture of TFT-LCD. For this problem, they proposed a mixed integer programming model and three heuristic algorithms to minimize makespan. Wang et al. [16] proposed the mixed integer programming model, genetic algorithm and simulated annealing algorithm to solve the scheduling problem of parallel batch-processing machines with unequal release times, non-identical job sizes, and different machine capacities. Studies which discussed the total completion time objective were done by Chang and Wang [17] and Ghazvini and Dupont [18].

Recently, metaheuristics such as simulated annealing (SA), tabu search (TS), and genetic algorithm (GA) have been successfully employed in solving difficult combinatorial optimization problems. A number of researchers have applied metaheuristics to solve batch processing machine problems. Melouk et al. [21] provided a simulated annealing approach to minimize makespan for scheduling a batch processing machine with different job sizes. An effective hybrid genetic algorithm is developed by Husseinzadeh Kashan et al. [22], using a representation that could dominate a random-key based genetic algorithm and also the simulated annealing approach by Melouk et al. [21]. Kohetal. [23], proposed some heuristics and a random key based representation genetic algorithm for the problems of minimizing makespan and total weighted completion time on a batch processing machine within compatible job families. A hybrid genetic algorithm is proposed by Chou et al. [24], to minimize makespan for the dynamic case of the single batch processing machine problem. Chou [25] developed a joint approach for scheduling in the presence of job ready times, based on the genetic algorithm in which the dynamic programming algorithm is used to evaluate the fitness of the generated solutions. Parsa et al. [26] presented a branch and bound algorithm to minimize makespan on a single batch processing machine with non-identical job sizes. The scheduling problem with bi-criteria of makespan and maximum tardiness by considering arbitrary size for jobs is also addressed by Husseinzadeh Kashan et al. [27]. Some researchers have also focused on scheduling with non-identical job sizes on identical parallel batch processing machines (Koh et al. [28], Chang et al. [29] and Husseinzadeh Kashan et al. [30]).

To the best of our knowledge, there has been no constant-ratio approximation algorithm for the general $P \mid r_j, s_j, b = 1 \mid C_{\max}$ problem to date. In this paper we combine the techniques of [5, 6, 12] to solve this problem and present an approximation algorithm with worst-case ratio $2+\varepsilon$, where $\varepsilon > 0$ can be made arbitrarily small.

We use *BPP* (Batch Processing Problem) to denote the general problem $P \mid r_j, s_j, b = 1 \mid C_{\max}$ and use *SBPP* to denote the problem which is the same as *BPP* except that all jobs can be split in size. The outline of our main idea is as follows: we first get a PTAS for *SBPP* in Section 2, and then use it to get a $(2+\varepsilon)$ -approximation algorithm for *BPP* in Section 3.

II. A PTAS FOR PROBLEM *SBPP*

In this section, we present a polynomial time approximation scheme for problem *SBPP*. We use opt to denote the optimal makespan of problem *SBPP*. Throughout this section, if a job has been split in size and some part of it has been scheduled, the remaining part of it will be treated as a single job.

The special case of $P \mid r_j, s_j, b = 1 \mid C_{\max}$ where all $r_j = 0$ and all $s_j = \frac{1}{B}$ is already strongly NP-hard [2], where B ($1 \le B < n$) is an integer. Lee et al. [2] observed that there exists an optimal schedule for this

special case in which all jobs are pre-assigned into batches according to the BLPT (full-batch-longest-processing-time) rule: rank the jobs in non-increasing order of processing times, and then batch the jobs by successively placing the B (or as many as possible) jobs with the longest processing times into the same batch.

To solve the general $P \mid r_j, s_j, b=1 \mid C_{\max}$ problem, we need the following modified version of the FBLPT rule.

MFBLPT Rule

Index the jobs in non-increasing order of their processing times. Place the job with the longest processing time in a batch. If the batch has enough room for the next job in the job list, then put the job in the batch; otherwise, place part of the job in the batch such that the batch is completely full and put the remaining part of the job at the head of the remaining job list and continue.

A job is called a *split job* if it is split in size. We call a job *available* if it has been released but not yet assigned into a batch. We call an available job *suitable* for a given batch if it can be added in that batch. We call a batch *available* if all the jobs in it have been released and it has not been scheduled.

We will perform several transformations on the given input to form an approximate problem instance that has a simpler structure. Each transformation potentially increases the objective function value by $O(\varepsilon)\cdot opt$, so we can perform a constant number of them while still staying within a $1+O(\varepsilon)$ factor of the original optimum. When we describe such a transformation, similar to [19], we shall say that it produces $1+O(\varepsilon)$ loss. To simplify notations we will assume throughout the paper that $1/\varepsilon$ is integral.

In the remainder of this section, we first simplify the problem by applying the rounding method. We proceed to define *short* and *long* jobs and then present a PTAS for the case where all jobs are short. Finally, we get a PTAS for problem *SBPP*.

A. Simplifying the Input

We use the FBLPT rule for all the jobs and get a series of batches. Denote by d the total processing time of these batches. Let $r_{\max}=\max_{1\le j\le n} r_j$. Then we get the following bounds for the optimal makespan of problem *SBPP*:

Lemma 1.

$$\max\{r_{\max}, p_{\max}, \frac{d}{m}\} \le opt \le r_{\max} + p_{\max} + \frac{d}{m}.$$

Proof. It is obvious that $opt \ge \max\{r_{\max}, p_{\max}\}$. By a job-interchange argument, we observe that for the special case of problem *SBPP* in which all $r_j = 0$, there exists an optimal schedule in which all jobs are pre-assigned into batches according to the MBLPT. Hence we get $opt \ge \frac{d}{m}$.

We use the MBLPT rule for all the jobs and get a number of batches. Starting from time $r_{\max}$ we schedule these batches by *List Scheduling* algorithm [20]: whenever a machine is idle, choose any available batch to start processing on that machine. Suppose that batch A is the last batch to finish in the List Scheduling schedule. It must be the case that from time $r_{\max}$ on, no machine is idle prior to the start of batch A, otherwise we would have scheduled A earlier. So A must start no later than $r_{\max}+\frac{d}{m}$. Then A must finish no later than $r_{\max} + p_{\max} + \frac{d}{m}$. Hence we get $opt \le r_{\max} + p_{\max} + \frac{d}{m}$, which completes the proof of the lemma.

Let $\delta = \varepsilon \cdot \max\{r_{\max}, p_{\max}, \frac{d}{m}\}$. Round each release time down to the nearest multiple of δ. After getting a schedule for the rounded problem, we can increase each batch's start time by δ in the output to obtain a feasible schedule for the original problem. As $\delta \le \varepsilon \cdot opt$, we get the following lemma.

Lemma 2. With $1+\varepsilon$ loss, we can assume that all the release times in an instance are multiple of δ, and the number of distinct release times is at most $1/\varepsilon + 1$.

One can see that all the jobs in J can be scheduled in the time interval $[0, r_{\max} + p_{\max} + \frac{d}{m}]$. We partition this time interval into $h = \left\lceil (r_{\max} + p_{\max} + \frac{d}{m})/\delta \right\rceil$ disjoint intervals in the form $[R_i, R_{i+1})$, where $R_i = (i-1)\delta$ for each $1 \le i \le h$ and $R_{h+1} = r_{\max} + p_{\max} + \frac{d}{m}$. Since $\delta = \varepsilon \cdot \max\{r_{\max}, p_{\max}, \frac{d}{m}\}$, we have $h \le 3/\varepsilon + 1$. Note that each of the first $h-1$ intervals has a length δ, and the last one has a length at most δ. By Lemma 2, we can assume that every job in J is released at some R_i $(1 \le i \le 1/\varepsilon + 1)$.

We say that a job (or a batch) is *short* if its processing time is smaller than $\varepsilon\delta$; and *long*, otherwise.

We can assume that there are only a constant number of distinct processing times of long jobs, as the following lemma states.

Lemma 3. With $1+3\varepsilon$ loss, the number of distinct processing times of long jobs, k, can be bounded from above by $1/\varepsilon^3 - 1/\varepsilon + 1$.

Proof. By Lemma 1 and the definition of long jobs, we know that for each long job j, $\varepsilon\delta \le p_j \le (1/\varepsilon)\delta$. We round each long job's processing time down to the nearest integral multiple of $\varepsilon^2\delta$. This creates a rounded instance in which there are at most $[(1/\varepsilon)\delta]/(\varepsilon^2\delta) - (\varepsilon\delta)/(\varepsilon^2\delta) + 1 = 1/\varepsilon^3 - 1/\varepsilon + 1$ distinct processing times of long jobs. Hence we get $k \le 1/\varepsilon^3 - 1/\varepsilon + 1$. Consider the optimal value of the rounded instance. Clearly, this value cannot be greater than opt, the optimal makespan of problem *SBPP*. As there are at most $3/\varepsilon^2$ long batches in any optimal schedule in the rounded instance, by replacing the rounded values with the original ones we may increase the solution value by at most $(3/\varepsilon^2)(\varepsilon^2\delta) = 3\delta \le 3\varepsilon \cdot opt$.

B. Short Jobs

In this subsection we concentrate on the case in which all the jobs are short. Based on the ideas of [5, 12], we present a very simple and easy to analyze approximation scheme for this case.

Denote by J_i the subset of jobs in J that are released at R_i $(1 \le i \le 1/\varepsilon + 1)$.

Algorithm ScheduleShort

Step 1. Use the MFBLPT rule for all the jobs in $J_1, J_2, \ldots, J_{1/\varepsilon+1}$, respectively.

Step 2. Use the List Scheduling algorithm [20] to schedule the obtained batches.

Theorem 1. If all the jobs are short, then Algorithm ScheduleShort is a PTAS for problem *SBPP*.

Proof. Let π be the schedule produced by Algorithm ScheduleShort. Suppose that $B_{i,1}, B_{i,2}, \ldots, B_{i,k_i}$ are the batches in π whose jobs are from J_i $(1 \le i \le 1/\varepsilon + 1)$ such that $B_{i,1}, B_{i,2}, \ldots, B_{i,k_i-1}$ are full batches and $q(B_{i,j}) \ge p(B_{i,j+1})$, where $p(B_{i,j})$ denotes the processing time of the longest job in $B_{i,j}$, and $q(B_{i,j})$ denotes the processing time of the shortest job in $B_{i,j}$ if $B_{i,j}$ is full and is set to zero otherwise. Then we have the following observation:

$$\sum_{j=1}^{k_i} (p(B_{i,j}) - q(B_{i,j})) \le p(B_{i,1}) < \varepsilon\delta. \qquad (1)$$

We modify all the batches $B_{i,j}$ in π as follows: reduce the processing time of each job in $B_{i,j}$ to $q(B_{i,j})$, $1 \le j \le k_i$, $1 \le i \le 1/\varepsilon + 1$. We call the obtained batches *modified batches*. Each original job is now modified into one new job if it has not been split, or two new jobs if it has been split. (Any original short job will be split at most once.) We call the new jobs *modified jobs*. Then we define two accessory problems:

SBPP1: To schedule the modified batches to minimize makespan.

SBPP2: To schedule the modified jobs to minimize makespan.

Both these problems deal with the modified jobs. But while *SBPP1* demands to leave the grouping of the modified jobs into batches as dictated by the MFBLPT rule, *SBPP2* allows the re-opening of the batches and playing with the grouping into batches. Hence, *SBPP2* might obtain a better makespan. However, we are going to prove that this is not the case by showing that $opt1 = opt2 \le opt$, where $opt1$ and $opt2$ denote the optimum values to *SBPP1* and *SBPP2*, respectively.

Any optimal solution to *SBPP1* is a feasible solution to *SBPP2*, therefore we get $opt1 \ge opt2$. On the other hand, any optimal solution to *SBPP2* can be transformed into a feasible solution to *SBPP1* without increasing the objective value, which implies that $opt1 \le opt2$. To show this, let us fix an optimal solution, π_2^*, to *SBPP2*. Suppose that A is the batch which starts earliest among the batches in π_2^* with the longest processing time. Suppose that A' is the batch which becomes available earliest among the modified batches with the longest processing time. We exchange the modified jobs which are in A but not in A' and the modified jobs which are in A' but not in A without increasing the completion time of any batch in π_2^*. Consequently, A' appears in modified π_2^*. Repeat this procedure until all the modified batches except those with processing time zero appear in modified π_2^*. The modified jobs with processing time zero are fully negligible and thus can be batched in such a way that the modified batches with processing time zero appear in modified π_2^*. We eventually achieve a feasible solution to *SBPP1*, whose makespan is not greater than that of π_2^*. It follows that $opt1 \le opt2$. Therefore we get $opt1 = opt2$. It is obvious that $opt2 \le opt$. Hence we get $opt1 = opt2 \le opt$.

Consider a schedule, denoted by π', which is obtained by using the List Scheduling algorithm for all the modified batches. Then we have

$$C_{\max}(\pi) \le C_{\max}(\pi') + \sum_{i=1}^{1/\varepsilon+1} \sum_{j=1}^{k_i} (p(B_{i,j}) - q(B_{i,j})).$$

Inequality (1) implies that the second term on the right-hand side of the above inequality is bounded from above by $(1/\varepsilon+1)\cdot\varepsilon\delta \le (\varepsilon+\varepsilon^2)\cdot opt$. Hence we get:

$$C_{\max}(\pi) \le C_{\max}(\pi') + (\varepsilon+\varepsilon^2)\cdot opt. \quad (2)$$

On the other hand, we claim that $C_{\max}(\pi') \le (1+2\varepsilon^2)\cdot opt.$ Suppose that A is the last batch to finish in π'. Consider the latest idle time point t_1 prior to the start of batch A. It is easy to see that t_1 must be a release time, i.e., one of the ends of the first $1/\varepsilon$ intervals. Since all batches in π' are short, any batch that starts before t_1 must finish earlier than $t_1+\varepsilon\delta$. By the rule of List Scheduling algorithm, any batch which starts after t_1 cannot be released earlier than t_1, otherwise it should be scheduled earlier. From t_1 onwards, no machine is idle prior to the start of batch A. It follows that $C_{\max}(\pi') \le opt1+2\varepsilon\delta \le (1+2\varepsilon^2)\cdot opt.$ Thus the claim holds. The claim, together with inequality (2), implies that $C_{\max}(\pi) \le (1+\varepsilon+3\varepsilon^2)\cdot opt$, completing the proof of the theorem.

C. General Case

We are now going to establish a PTAS to solve the general *SBPP* problem.

By the job interchange argument, we get the following lemma which plays an important role in design and analysis of our algorithm.

Lemma 4. There exists an optimal schedule with the following properties:

(1) on any one machine, the batches started (but not necessarily finished) in the same interval are processed successively in the order of non-increasing batch processing times, and

(2) from time 0 onwards, interval by interval, the batches started in the same interval are filled in the order of non-increasing batch processing times such that each batch contains as many as possible of the longest suitable jobs, and

(3) any job can be split in size whenever necessary, therefore all the batches in the same interval are full batches except possibly the shortest one.

The following lemma is useful:

Lemma 5. With $1+3\varepsilon+\varepsilon^2$ loss, we can assume that no short job is included in long batches.

Proof. By Lemma 4, there exists an optimal schedule in which only the last long batch in each interval may contain short jobs. Therefore, we can stretch those intervals to make extra spaces with length $\varepsilon\delta$ for the short jobs that are included in the long batches. Since there are $3/\varepsilon+1$ intervals, we may increase the solution value by at most $(3+\varepsilon)\delta$, which is no more than $(3\varepsilon+\varepsilon^2)\cdot opt$. This completes the proof of the lemma.

Combining Lemma 5 and Theorem 1, we can determine the batch structure of short jobs at the beginning of the algorithm as follows: use the MFBLPT rule for all the short jobs in J_i $(1 \le i \le 1/\varepsilon+1)$ and get a series of short batches.

The idea for dealing with long jobs is essentially based on enumeration. Recall that the number of distinct processing times of long jobs, k, has been bounded from above by $1/\varepsilon^3 - 1/\varepsilon + 1$ (Lemma 3). Without loss of generality, let $P_1, P_2, \ldots, P_k$ be the k distinct processing times of long jobs. Suppose further that $P_1 < P_2 < \cdots < P_k$. We now turn to the concepts of machine configurations and execution profiles.

let us fix a schedule, π.We delete from π all the jobs and the short batches, but retain all the empty long batches, which are represented, respectively, by their processing times. For a particular machine, we define a *machine configuration*, with respect to π, as a vector $(c_1, c_2, \ldots, c_{3/\varepsilon+1})$, where c_i consists of all the empty long batches started on that machine in interval $[R_i, R_{i+1})$, $1 \le i \le 3/\varepsilon+1$. For the sake of clarity, we define c_i equivalently as a k-tuple $(x_{i1}, x_{i2}, \ldots, x_{ik})$, where x_{ij} is the number of empty long batches started in interval $[R_i, R_{i+1})$ on the machine with P_j as their processing times, $1 \le i \le 3/\varepsilon+1$, $1 \le j \le k$.

The processing time of a long batch is chosen from the $k \le 1/\varepsilon^3 - 1/\varepsilon + 1$ values. When c_i contains l empty long batches (i.e., $\sum_{j=1}^{k} x_{ij} = l$), the number of different possibilities is not greater than k^l. Since a feasible schedule has the property that on any one machine, at most $1/\varepsilon$ long batches are started in each of the intervals, the number of machine configurations to consider, Γ, can be roughly bounded from above by $(1+k+k^2+\cdots+k^l)^{3/\varepsilon+1} < 2^{3/\varepsilon+1}\cdot k^{3/\varepsilon+1}$.

This allows us to say that, for a given schedule, a particular machine has a certain configuration. We denote the configurations as $1,2,\ldots,\Gamma$. Then for any schedule, we define an *execution profile* as a tuple $(m_1, m_2, \ldots, m_\Gamma)$, where m_i is the number of machines with configuration i for that schedule. Therefore, there are at most $(m+1)^\Gamma$ execution profiles to consider, a polynomial in m.

We next present our algorithm.

Algorithm ScheduleSplit

Step 1. Get all possible execution profiles.

Step 2. For each of them, do the following:

(a) Assign a configuration for each machine according to the profile. If this is not possible, delete the profile.

(b) On each machine in each interval, start the specified empty long batches as early as possible in the order of non-increasing processing times. If some batch has to be delayed to start in one of the next intervals, then delete the profile.

(c) From time 0 onwards, interval by interval, fill the empty long batches started in the same interval in the order of non-increasing batch processing times such that each of them contains as many as possible of the longest suitable jobs (any job can be split in size whenever necessary). If some long job cannot be assigned into a batch and has to be left, then delete the profile.

(d) Run Algorithm ScheduleShort in the spaces left by the long batches and get a feasible schedule. If a short batch crosses an interval, we stretch the end of the interval to make an extra space with length $\varepsilon\delta$ for it such that it need no longer cross the interval.

Step 3. From among the obtained feasible schedules, select the one with the smallest makespan.

Theorem 2. Algorithm ScheduleSplit is a PTAS for the general *SBPP* problem.

Proof. By Lemma 4, the long batches started in the same interval on the same machine can be arranged in the order of non-increasing batch processing times. Note that we can stretch the end of an interval to make an extra space with length $\varepsilon\delta$ for a crossing short batch such that it need no longer cross the interval. Therefore given an execution profile, we can first start the empty long batches as early as possible while keeping them in the specified intervals, and then run Algorithm ScheduleShort in the spaces between them.

Any optimal schedule is associated with one of the $(m+1)^{\Gamma}$ execution profiles. Given an execution profile that can lead to an optimal schedule, our way to deal with long jobs in Algorithm ScheduleSplit is optimal, while invoking Algorithm ScheduleShort will yield at most $1+\varepsilon+3\varepsilon^2$ loss. Combining Lemmas 2, 3 and 5, by taking the smallest one among all obtained feasible schedules, Algorithm ScheduleSplit can be executed with at most $1+8\varepsilon+4\varepsilon^2$ loss.

It is easy to see that the time complexity of Algorithm ScheduleSplit is $O(n\log n+n\cdot(m+1)^{\Gamma+1})$.

III. AN ALGORITHM FOR PROBLEM *BPP*

Now we start to construct an approximation algorithm for *BPP*. We say that a batch *splits* a job if it contains some part but not the last part of the job, and the batch is now called a *splitting batch*.

Algorithm ScheduleWhole

Step 1: Get a $(1+\frac{\varepsilon}{2})$-approximation schedule π_1 for *SBPP* by Algorithm ScheduleSplit.

Step 2: Move out all split jobs from π_1 and open a new batch for each of them.

Step 3: Process the new batches successively at the end of π_1', on the same machines as the corresponding splitting batches in π_1, where π_1' is the schedule that is obtained from π_1 after removing from it all split jobs.

Theorem 3. Algorithm ScheduleWhole is a $(2+\varepsilon)$-approximation algorithm for problem *BPP*, where $\varepsilon>0$ can be made arbitrarily small.

Proof. Denote by π the schedule given by Algorithm ScheduleWhole. Let $C_{\max}$ and $C^*_{\max}$ be the makespans of π and an optimal schedule for *BPP*, respectively. Recall that opt denotes the optimal makespan of problem *SBPP*. It is obvious that $opt\le C^*_{\max}$ and π is a feasible schedule for *BPP*. Note that π consists of two parts, one of which is π_1' and another consists of the new batches opened for the split jobs. The completion time C_1 of the former part is no more than $(1+\frac{\varepsilon}{2})\cdot opt$. Let us consider the maximum total processing time C_2 on any machine of the latter part. From Algorithm ScheduleSplit, each batch splits at most one job and each job can be split at most once in π_1. Since the processing time of a split job cannot be greater than the corresponding splitting batch, it follows that $C_2\le C_1$. Thus we get

$$C_{\max}\le C_1+C_2\le 2(1+\frac{\varepsilon}{2})\cdot opt \le (2+\varepsilon)\cdot opt\le (2+\varepsilon)\cdot C^*_{\max}.$$

This completes the proof of the theorem.

Note that in the algorithm the treatment of the split jobs is very trivial (each one in its own batch and all the new batches are processed at the end of π_1'). Is it possible to improve this and get a better worst-case ratio? In [12], the authors showed an example to explain why more involved techniques for batching the split jobs do not seem to yield a better worst-case ratio. One might expect that we can make a more educated choice of the new batches' start times to improve the ratio. For example, each new batch starts immediately after the completion of the corresponding splitting batch. However, this is not the case, because the generic bad cases are the same.

In Algorithm ScheduleWhole, Step 1 can be executed in $O(n\log n+n\cdot(m+1)^{\Gamma+1})$ time, while Steps 2 and 3 can be executed in $O(n)$ time, therefore this algorithm can be implemented in $O(n\log n+n\cdot(m+1)^{\Gamma+1})$ time.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (60970105), National Natural Science Foundation of China for Distinguished Young Scholar (11161035), Special Fund of Shandong Provincial Information Industry Department (No. 2008X00039), and Shandong Provincial Soft Science Research Program (2011RKGB5040).

REFERENCES

[1] Chandru, V., Lee, C.-Y., & Uzsoy, R. (1993). Minimizing total completion time on batch processing machines. International Journal of Production Research, 31, 2097–2121.

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

[3] Sung, C. S., & Choung, Y. I. (2000). Minimizing makespan on a single burn-in oven in semiconductor manufacturing. European Journal of Operational Research, 120, 559–574.

[4] Lee, C.-Y., & Uzsoy, R. (1999). Minimizing makespan on a single batch processing machine with dynamic job arrivals. International Journal of Production Research, 37, 219–236.

[5] Shuguang Li, Guojun Li, Shaoqiang Zhang. (2005). Minimizing makespan with release times on identical parallel batching machines. Discrete Applied Mathematics, 148, 127–134.

[6] Shuguang Li, Guojun Li, Shaoqiang Zhang. Minimizing maximum lateness on identical parallel batch processing machines. Lecture Notes in Computer Science 3106: Proceedings of the 10th Annual International Conference on Computing and Combinatorics, 229–237, 2004.

[7] Dupont, L., & Ghazvini, F. J. (1997). A branch and bound algorithm for minimizing mean flow time on a single batch processing machine. International Journal of Industrial Engineering, 4, 197–203.

[8] Qi, X., & Tu, F. (1999). Earliness and tardiness scheduling problems on a batch processor. Discrete Applied Mathematics, 98, 131–145.

[9] Wang, C.-S., & Uzsoy, R. (2002). A genetic algorithm to minimize maximum lateness on a batch processing machine. Computers &Operations Research, 29, 1621–1640.

[10] Uzsoy, R. (1994). Scheduling a single batch processing machine with non-identical job sizes. International Journal of Production Research, 32, 1615–1635.

[11] Zhang, G., Cai, X., Lee, C.-Y., & Wong, C. K. (2001). Minimizing makespan on a single batch processing machine with nonidentical job sizes. Naval Research Logistics, 48, 226–240.

[12] Shuguang Li, Guojun Li, Xiaoli Wang, Qiming Liu. Minimizing Makespan on a Single Batching Machine with Release Times and Non-Identical Job Sizes. Operations Research Letters, 33(2): 157–164, 2005.

[13] Q.Q. Nong, C.T. Ng and T.C.E. Cheng (2008). The bounded single-machine parallel-batching scheduling problem with family jobs and release dates to minimize makespan, Operations Research Letters, 36(1), 61-66.

[14] Dupont, L., & Dhaenens-Flipo, C. (2002). Minimizing the makespan on a batch machine with non-identical job sizes: An exact procedure. Computers & Operations Research, 29, 807–819.

[15] Chung, S. H., Tai, Y. T., & Pearn, W. L. (2008). Minimising makespan on parallel batch processing machines with non-identical ready time and arbitrary job sizes. International Journal of Production Research. doi:10.1080/00207540802010807.

[16] Wang Hui-Mei and Fuh-Der Chou (2010). Solving the parallel batch-processing machines with different job sizes, and capacity limits by metaheuristics. Expert Systems with Applications, 37, 1510-1521.

[17] Chang, P.-C., & Wang, H.-M. (2004). A heuristic for a batch processing machine scheduled to minimize total completion time with non-identical job sizes. International Journal of Advanced Manufacturing Technology, 24, 615–620.

[18] Ghazvini, F. J., & Dupont, L. (1998). Minimizing mean flow times criteria on a single batch processing machine with non-identical jobs sizes. International Journal of Production Economics, 55, 273–280.

[19] F. Afrati, E., C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, M. Sviridenko (1999). Approximation schemes for minimizing average weighted completion time with release dates, Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, New York, October, 32–43.

[20] R. L. Graham (1966). Bounds for certain multiprocessor anomalies. Bell System Technical Journal, 45: 1563–1581.

[21] Melouk S, Damodaran P, Chang P-Y. Minimizing makespan for single machine batch processing with non-identical job sizes using simulated annealing. International Journal of Production Economics, 2004, 87: 141–7.

[22] Husseinzadeh Kashan A, Karimi B, Jolai F. Effective hybrid genetic algorithm for minimizing makespan on a single batch processing machine with non-identical job sizes. International Journal of Production Research, 2006, 44: 2337–60.

[23] Koh S-G, Koo P-H, Kim D-C, Hur W-S. Scheduling a single batch processing machine with arbitrary job sizes and incompatible job families. International Journal of Production Economics, 2005, 98: 81–96.

[24] Chou FD, Chang PC, Wang HM. A hybrid genetic algorithm to minimize makespan for the single batch machine dynamic scheduling problem. International Journal of Advanced Manufacturing Technology, 2006, 31: 350–9.

[25] Chou FD. A joint GA+DP approach for single burn-in oven scheduling problems with makespan criterion. International Journal of Advanced Manufacturing Technology, 2007, 35: 587–95.

[26] N. Rafiee Parsa, B. Karimi, A. Husseinzedeh Kashan. A branch and bound algorithm to minimize makespan on a single batch processing machine with non-identical job sizes. Computers & Operations Research, 2010, 37 (10): 1720-1730.

[27] Husseinzadeh Kashan A, Karimi B, Jolai F. Bi-criteria scheduling on a single batch processing machine with non-identical job sizes. In: Proceeding of the 12th IFAC symposium on information control problems in manufacturing, INCOM'2006, St-Etienne, France, 2006b.

[28] Koh S-G, Koo P-H, Ha J-W, Lee W-S. Scheduling parallel batch processing machines with arbitrary job sizes and incompatible job families. International Journal of Production Research, 2004, 42: 4091–41107.

[29] Chang P Y, Damodaran P, Melouk S. Minimizing makespan on parallel batch processing machines. International Journal of Production Research, 2004, 42: 4211–20.

[30] Husseinzadeh Kashan A, Karimi B, Jenabi M. A hybrid genetic heuristic for scheduling parallel batch processing

machines with arbitrary job sizes. Computers & Operations Research, 2008, 35: 1084–98.

Shuguang Li was born in Shandong, China in 1970. He received the PhD degree in operations research and cybernetics from the Shandong University, Jinan, China, in 2007. He is an associate professor in the College of Computer Science and Technology, Shandong Institute of Business and Technology, Yantai, China. His current research areas are combinatorial optimization and theoretical computer science.