ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.10, October 2006, pp.2063−2068
DOI: 10.1360/jos172063

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

极小化加权完工时间和的无界批量机器并行调度问题*

李曙光[1,2+], 李国君[1,3], 王秀红[4]

[1](山东大学 数学与系统科学学院,山东 济南 250100)

[2](烟台大学 数学与信息科学系,山东 烟台 264005)

[3](中国科学院 软件研究所,北京 100080)

[4](鲁东大学 数学与信息学院,山东 烟台 264025)

Minimizing Total Weighted Completion Time on Parallel Unbounded Batch Machines

LI Shu-Guang[1,2+], LI Guo-Jun[1,3], WANG Xiu-Hong[4]

[1](School of Mathematics and Systems Science, Shandong University, Ji'nan 250100, China)

[2](Department of Mathematics and Information Science, Yantai University, Yantai 264005, China)

[3](Institute of Software, Chinese Academy of Sciences, Beijing 100080, China)

[4](School of Mathematics and Information, Ludong University, Yantai 264025, China)

+ Corresponding author: Phn: +86-535-6903498, E-mail: sgliytu@hotmail.com

Li SG, Li GJ, Wang XH. Minimizing total weighted completion time on parallel unbounded batch machines. *Journal of Software*, 2006,17(10):2063−2068. http://www.jos.org.cn/1000-9825/17/2063.htm

Abstract: This paper considers the problem of scheduling n jobs on m parallel unbounded batch machines to minimize the total weighted completion time. Each job is characterized by a positive weight, a release time and a processing time. Each unbounded batch machine can process up to B ($B \geq n$) jobs as a batch simultaneously. The processing time of a batch is the longest processing time among jobs in the batch. Jobs processed in the same batch have the same completion time, i.e., their common starting time plus the processing time of the batch. A polynomial time approximation scheme (PTAS) for this problem is presented.

Key words: polynomial time approximation scheme; scheduling; parallel unbounded batch machines; total weighted completion time; release times

摘 要: 考虑无界批量机器并行调度中极小化加权完工时间和问题.设有 n 个工件和 m 台批加工同型机.每个工件具有一个正权因子、一个释放时间和一个加工时间.每台机器可以同时加工 $B \geq n$ 个工件.一个批次的加工时间是该批次所包含的所有工件的加工时间的最大者.在同一批次中加工的工件有相同的完工时间,即它们的共同开始时间加上该批次的加工时间.给出了一个多项式时间近似方案(PTAS).

关键词: 多项式时间近似方案;调度;无界批量并行机;加权完工时间和;释放时间

中图法分类号: TP301 文献标识码: A

* Supported by the National Natural Science Foundation of China under Grant Nos.10271065, 60373025 (国家自然科学基金); the Science and Technology Development Foundation of Tianjin Municipal Education Commission of China under Grant No.20051519 (天津市教委科技发展基金)

Received 2004-05-08; Accepted 2005-07-08

A batch machine is a machine that can process up to B jobs simultaneously as a batch. The research on batch machine scheduling is motivated by burn-in operations in semiconductor manufacturing[1]. There are two variants of the burn-in model: the *unbounded model*, in which $B\geq n$ so that there is effectively no upper bound on the number of jobs that can be processed in the same batch; and the *bounded model*, in which $B<n$ so that there is a restrictive upper bound. The unbounded model arises for instance in situations where compositions need to be hardened in kilns, and a kiln is sufficiently large that it does not restrict batch seizes[2]. In this paper, we consider the unbounded model.

The problem that we study can be formulated as follows. There is a set of n jobs where job j is associated with a positive weight w_j, a processing time p_j that specifies the minimum time needed to process the job, and a release time r_j before which it cannot be scheduled. The processing time of a batch is the longest processing time among the jobs in the batch. The completion time of a batch is equal to its start time plus its processing time. Jobs processed in the same batch have the same completion time, which is the completion time of the batch in which the jobs contained. Once processing is begun on a batch, no job can be removed from the batch until the processing of the batch is complete. The objective is to schedule the jobs on a set of m identical parallel unbounded batch machines so as to minimize $\sum w_jC_j$, where C_j denotes the completion time of job j in the schedule. This objective function is referred to as the *total weighted completion time*. Using the notation of Graham *et al.*[3], we denote this problem as $P|r_j,B\geq n|\sum w_jC_j$, where P denotes that the scheduling environment contains m identical parallel machines. In comparison, the single unbounded batch machine case (m=1) is denoted as $1|r_j,B\geq n|\sum w_jC_j$.

The single unbounded batch machine case has been extensively studied. Along with other results, P. Brucker *et al.*[2] presented an $O(n\log n)$-time exact algorithm for $1|B\geq n|\sum w_jC_j$ (all r_j=0). X. Deng and Y. Zhang[4] proved that the problem $1|r_j,B\geq n|\sum w_jC_j$ is NP-hard. X. Deng *et al.*[5] and S. Li *et al.*[6] presented the first PTAS for $1|r_j, B\geq n|\sum C_j$ (all w_j=1) and $1|r_j,B\geq n|\sum w_jC_j$ respectively. In fact, the ideas used in Ref.[6] can be generalized to get a PTAS for $Pm|r_j,B\geq n|\sum w_jC_j$ (m is a fixed number).

To the best of our knowledge, the problem $P|r_j,B\geq n|\sum w_jC_j$ (m is part of the input) has not been studied to date. In this paper, we present a PTAS for this problem. The result is based on the techniques developed in Ref.[7], where the authors presented the first PTASs for the classical scheduling problem (B=1) of minimizing total weighted completion time in the presence of release times in various machine models. Even though the general framework in our study follows that in Ref.[7], the special property of unbounded batch machines makes the detailed analysis quite non-trivial. For example, dealing with small jobs becomes different from their methods. Scheduling jobs within a block cannot trivially follow their method and demands special treatment.

This paper is organized as follows: In Section 1, we discuss some general techniques and lemmas that apply throughout the paper. In Section 2, we present an overview of the dynamic programming framework. In Section 3, we show that there exists a $(1+\varepsilon)$-approximate schedule such that one can represent compactly the information about jobs remaining after each block. In Section 4, we focus on the case of scheduling jobs within a block. We conclude this paper in Section 5.

1 Preliminaries

To establish a *polynomial time approximation scheme* (PTAS), for any given positive number ε, we need find a solution within a $(1+\varepsilon)$ factor of the optimum in polynomial time. In this section, we aim to transform any input into one with a simple structure. We say, as in Ref.[7], a transformation produces $1+O(\varepsilon)$ *loss* if it potentially increases the objective function value by $1+O(\varepsilon)$. To simplify notations we will assume throughout the paper that $1/\varepsilon$ is integral and $\varepsilon\leq 1/4$ (otherwise we simply work with ε' such that $\varepsilon'<\varepsilon$, $1/\varepsilon'$ is integral and $\varepsilon'\leq 1/4$). We call a job *available* if it has been released but not yet been scheduled.

Some of the lemmas mentioned in subsequent sections, introduced by Ref.[7] for the classical scheduling problems (B=1), are still effective for our problem. For simplicity, we omit proofs when we present these lemmas.

Lemma 1.1[7]. With $1+\varepsilon$ loss, we can assume that all processing times and release times are integer powers of $1+\varepsilon$.

For an arbitrary integer x, we define $R_x:=(1+\varepsilon)^x$. As a result of Lemma 1.1, we can assume that all release times are of the form R_x for some integer x. We partition the time interval $(0,\infty)$ into disjoint intervals of the form $I_x:=[R_x,R_{x+1}]$ (Lemma 1.2 below ensures that no jobs are released at time 0). We will use I_x to refer to both the interval and the size $(R_{x+1}-R_x)$ of the interval. We will often use the fact that $I_x=\varepsilon R_x$.

Lemma 1.2[7]. With $1+\varepsilon$ loss, we can enforce $r_j\geq\varepsilon p_j$ for all jobs j.

Lemma 1.3[7]. Each batch crosses at most $s:=\lceil\log_{1+\varepsilon}(1+1/\varepsilon)\rceil$ intervals.

As in Ref.[7], we classify jobs (batches) as small and large. We say that a job j is *small* with respect to an interval I_x if $p_j\leq\varepsilon I_x$, and *large* otherwise. We say that a batch is *large* if it contains at least one large job, and *small* otherwise.

Lemma 1.4[7]. With $1+\varepsilon$ loss, we restrict attention to schedules in which no small batch crosses an interval.

The following lemma shows that each job becomes small after waiting a constant number of intervals.

Lemma 1.5. Let $k:=\lfloor 1+\log_{1+\varepsilon}(1/\varepsilon^3)\rfloor$, $r_j=R_x$. Then job j is small in interval I_{x+k}.

Proof: By Lemma 1.2, we get $p_j\leq\dfrac{R_x}{\varepsilon}=\dfrac{\varepsilon(1+\varepsilon)^k R_x}{\varepsilon^2(1+\varepsilon)^k}<\varepsilon I_{x+k}$. □

Lemma 1.6. The number of distinct processing times of available large jobs in each interval is at most $k=\lfloor 1+\log_{1+\varepsilon}(1/\varepsilon^3)\rfloor$.

Proof: Consider interval I_x. For each job j in the set of the available large jobs in I_x, Lemma 1.2 yields $R_x\geq\varepsilon p_j$. On the other hand, since j is large, we get $p_j\geq\varepsilon I_x=\varepsilon^2 R_x$. Since all job processing times are integer powers of $1+\varepsilon$, the number is as claimed. □

Let $P_{x1}<\ldots<P_{xk}$ be the k distinct processing times of the available large jobs in interval I_x, regardless of the fact that some of them may not exist. We call the jobs with processing time P_{xl} the P_{xl}-*jobs*, l=1,2,…,k.

We combine Lemmas 1.5 and 1.6 to get a simple observation.

Lemma 1.7. The P_{xl}-jobs cannot be released earlier than R_{x-k+l} $(1\leq l\leq k)$.

2 The Dynamic Programming Framework

In this section, we give an overview of the dynamic programming framework from Ref.[7].

The basic idea is to decompose the time horizon into a sequence of *blocks*, say $B_1,B_2,\ldots$, each containing $s=\lceil\log_{1+\varepsilon}(1+1/\varepsilon)\rceil$ consecutive intervals. By the choice of the block size and Lemma 1.3, no batch crosses an entire block. In other words batches that start in B_i finish either in B_i or B_{i+1}. A *frontier* describes the potential ways that batches in one block finish in the next.

Lemma 2.1.[7] There exists a $(1+\varepsilon)$-approximate schedule which considers only $(m+1)^{s/\varepsilon}$ feasible frontiers between any two blocks.

Proof: By Lemma 1.4, we can restrict attention to schedules in which no small batch crosses an interval. Each block consists of a fixed number s of intervals. Fix an optimal schedule and consider any machine in a block B_i. A large batch B_j continuing from the preceding block finishes in one of the s intervals of block B_i which we denote by $I_{x(B_j)}$. We can round up the completion time of B_j, $C(B_j)$, to $C'(B_j)$ where $C'(B_j)=R_{x(B_j)}+i\cdot\varepsilon\cdot I_{x(B_j)}$ for some integer $0\leq i\leq 1/\varepsilon-1$. This will increase the schedule value by only a $1+\varepsilon$ factor. Thus we can restrict the completion times of crossing batches to s/ε discrete time instants. Each machine realizes one of these possibilities. A

frontier can thus be described as a tuple $(m_1,m_2,\ldots,m_{s/\varepsilon})$ where m_i is the number of machines with crossing batches finishing at the *ith* discrete time instant. Therefore there are at most $(m+1)^{s/\varepsilon}$ frontiers to consider. □

Let F_{i+1} denote the set of feasible frontiers between blocks B_i and B_{i+1}. We are now ready to describe the dynamic programming framework from Ref.[7]. The dynamic programming table entry $O(i,F,U)$ stores the minimum total weighted completion time achievable by starting the set U of jobs before the end of block B_i while leaving a frontier of $F \in F_{i+1}$ for block B_{i+1}. Given all the table entries for some i, the values for $i+1$ can be computed as follows. Let $W(i,F_1,F_2,V)$ be the minimum total weighted completion time achievable by scheduling the set of jobs V in block B_i, with F_1 as the incoming frontier from block B_{i-1} and F_2 the outgoing frontier to block B_{i+1}. We obtain the following equation.

$$O(i+1,F,U)=\min_{F'\in F_{i+1},V\subset U}(O(i,F',V)+W(i+1,F',F,U-V)),\quad F\in F_{i+2}. \tag{1}$$

There are two difficulties in implementing the dynamic programming: First, we cannot maintain the table entries for each possible subset of jobs in polynomial time. Therefore we need to show the existence of approximate schedules that have compact representations for the set of subsets of jobs remaining after each block; Second, we need a procedure that computes the quantity $W(i,F_1,F_2,V)$. The next two sections describe how to achieve these two objectives.

3 Compact Representation of Job Subsets

The first difficulty in implementing the dynamic programming is to show that it is sufficient to maintain information in the table for only a few (polynomial) subsets of jobs. To get around this difficulty, we are going to show that there exists a $(1+\varepsilon)$-approximate schedule such that one can represent compactly the information about jobs remaining after each block.

Suppose that the current block under consideration is B_i. Denote by I_y the first interval in block B_{i+1}. It's obvious that the jobs released no earlier than R_y cannot be scheduled in block B_i. We also have the following lemma.

Lemma 3.1. With $1+\varepsilon$ *loss* we can assume that there is no small job available in I_y after we process block B_i.

Proof: If there are small jobs available in I_y, we can start a batch with processing time at most εI_y to schedule all of them at the end of the frontier of block B_i, in particular right after the crossing batch with the smallest completion time among all crossing batches. This will increase the schedule value by only a $1+\varepsilon$ factor. □

Thus we need only consider the different possibilities of the large jobs available in I_y. Consider the P_{yl}-jobs available in I_y $(1\le l\le k=\lfloor 1+\log_{1+\varepsilon}(1/\varepsilon^3)\rfloor)$. By Lemma 1.7, these jobs cannot be released earlier than R_{y-k+l}. Based on the special property of parallel unbounded batch machines, we have the following observation: if a P_{yl}-job is started at time t, then all the P_{yl}-jobs released no later than t should be started no later than t. (as for the case of a single unbounded batch machine, the following observation is true: if a job j is started at time t, then all the jobs which are released no later than t and with processing time no more than p_j should be started no later than t. However, this observation doesn't hold for the case of parallel unbounded batch machines). Therefore the number of different possibilities of the P_{yl}-jobs available in I_y is at most $k-l+1$. Together with Lemma 1.6, the number of different possibilities of the large jobs available in I_y can be up-bounded by $k!$. Thus we get the following lemma.

Lemma 3.2. There is a $(1+\varepsilon)$-approximate schedule S such that for each block B_i the following is true:

- There are $g=k!$ sets $G_i^1,G_i^2,\ldots,G_i^g$ that can be constructed in polynomial time, and
- G_i, the set of jobs remaining in S after block B_i, is one of the $\{G_i^1,G_i^2,\ldots,G_i^g\}$.

4 Scheduling Jobs within a Block

We turn to the second difficulty in implementing the dynamic programming. We will describe how to compute

$W(i,F_1,F_2,V)$. We settle this problem for a relaxation. A $(1+\varepsilon+\varepsilon^2)$ decision procedure for computing $W(i,F_1,F_2,V)$ outputs a schedule that is within $(1+\varepsilon+\varepsilon^2)$ of $W(i,F_1,F_2,V)$ and shifts the frontier F_2 by at most a $(1+\varepsilon+\varepsilon^2)$ factor. Clearly such a procedure suffices in order to compute a $(1+O(\varepsilon))$-optimal solution to the dynamic program given above. We now describe a $(1+\varepsilon+\varepsilon^2)$ decision procedure that runs in polynomial time for each fixed ε. For simplicity, we use the convention in this section that an optimal schedule refers to a best schedule for the set of jobs V that are scheduled in block B_i with incoming and outgoing frontiers specified by F_1 and F_2. We use the analogous convention for a schedule or an approximate schedule.

Let us fix a particular schedule. We delete all the jobs and small batches from this schedule, but retain all the empty large batches. We move these batches to the left as far as possible while keep them in the specified intervals. Thereby we get an *outline*, which specifies a start time, a processing time and a machine for each of the empty large batches. If an outline is obtained from an optimal schedule, we call it an *optimal outline*.

Procedure FillingOutline

Given an outline, do the following:

Step 1. Fill the empty large batches in the order of non-decreasing completion times such that each of them contains all the currently available large jobs with processing times no more than that of the batch.

Step 2. Stretch each interval I_x to create an extra space with length εI_x right before the batch with the minimum start time among all the batches started in I_x. From time $\min_j r_j$ onwards, start a batch in each extra space to schedule all the currently available small jobs (if an interval is entirely covered by a crossing batch on each machine, then we need not stretch this interval).

Step 3. Start a batch with length at most εI_y to schedule the small jobs available in I_y at the end of the frontier of block B_i, where I_y denotes the first interval in block B_{i+1} (by Lemma 3.1).

Lemma 4.1. Given an optimal outline, Procedure FillingOutline will find a $(1+\varepsilon+\varepsilon^2)$-approximate schedule.

Proof: Denote by C_j^{opt} the completion time of job j in the optimal schedule from which the given optimal outline is obtained. Denote by $x(j)$ the index of the interval in which job j starts. Denote by C_j the completion time of job j in the schedule generated by Procedure FillingOutline. It's easy to see that when Step 1 terminates, each large job j completes no later than C_j^{opt}. On the other hand, a simple volume summation argument shows that $\sum_{x\le x(j)} \varepsilon I_x \le \varepsilon R_{x(j)+1} = \varepsilon(1+\varepsilon)\cdot R_{x(j)} \le (\varepsilon+\varepsilon^2)C_j^{opt}$. It follows that for all jobs j, Step 2 and Step 3 ensure $C_j \le (1+\varepsilon+\varepsilon^2)\cdot C_j^{opt}$. Hence the lemma holds. □

Lemma 4.2. An optimal outline has the following properties:

1) Any two processing times of the batches which are started in the same interval on each machine are distinct; and
2) The batches started in the same interval on each machine are processed successively in the order of increasing processing times.

We define a *machine configuration* to be the restriction of an outline to a particular machine. We are going to enumerate over all potential outlines, among which there exists an optimal outline. We observe that there are at most $1/\varepsilon$ large batches started in each interval. By Lemma 1.6, we know that the processing time of each large batch is chosen from $k=\lfloor 1+\log_{1+\varepsilon}(1/\varepsilon^3)\rfloor$ values. Recall that block B_i consists of $s=\lceil \log_{1+\varepsilon}(1+1/\varepsilon)\rceil$ consecutive intervals. Combining Lemma 4.2, we can up-bound the number of different machine configurations by

$$\Gamma = \left[\binom{k}{0}+\binom{k}{1}+\ldots+\binom{k}{1/\varepsilon}\right]^s < k^{s/\varepsilon}.$$

We denote the different machine configurations as $1,2,\ldots,\Gamma$. An outline can now be defined as a tuple $(m_1,$

$m_2,\ldots,m_\Gamma$), where m_i is the number of machines with configuration i. Therefore there are at most $(m+1)^\Gamma$ outlines to consider, a polynomial in m.

It should be stressed that in each outline, the empty large batches started in the same interval on each machine are arranged in the order of increasing processing times (by Lemma 4.2). Out of these outlines we consider only those that are compatible with the incoming and outgoing frontiers F_1 and F_2. We invoke Procedure FillingOutline to deal with each such outline. From among the generated feasible schedules, we select the one with the minimum objective value. Clearly, if there is no feasible schedule generated, we simply set $W(i,F_1,F_2,V)=+\infty$. Note that there are at most ms/ε empty large batches in each outline, and it takes $O((ms/\varepsilon)\log(ms/\varepsilon))$ time to determine the order in which these batches are filled. By Lemma 4.1, we get the following lemma.

Lemma 4.3. There is a $(1+\varepsilon+\varepsilon^2)$ decision procedure to compute $W(i,F_1,F_2,V)$ that runs in time $O((m+1)^\Gamma(ms/\varepsilon)\log(ms/\varepsilon))$ where $\Gamma<k^{s/\varepsilon}$.

5 Conclusion

We now return to the dynamic programming Eq.(1). By Lemmas 1.5 and 3.1, we know that any job will be processed after waiting at most 3 blocks. Therefore we need handle no more than $3n$ blocks by ignoring the empty blocks within which no job is processed. Combining Lemmas 2.1, 3.2 and 4.3, we get our main result.

Theorem 5.1. There is a PTAS for $P|r_j,B\geq n|\sum w_jC_j$ that runs in time $O(k!\cdot(m+1)^{2\Gamma}\cdot n)$, where $k=\lfloor 1+\log_{1+\varepsilon}(1/\varepsilon^3)\rfloor$, $s=\lceil \log_{1+\varepsilon}(1+1/\varepsilon)\rceil$, $\Gamma<k^{s/\varepsilon}$.

References:

[1] Lee CY, Uzsoy R, Martin Vega LA. Efficient algorithms for scheduling semiconductor burn-in operations. Operations Research, 1992,40:764–775.

[2] Brucker P, Gladky A, Hoogeveen H, Kovalyvov MY, Potts CN, Tautenhahn T, van de Velde SL. Scheduling a batching machine. Journal of Scheduling, 1998,1:31–54.

[3] Graham RL, Lawler, Lenstra JK, Rinnooy Kan AHG. Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics, 1979,5:287–326.

[4] Deng X, Zhang Y. Minimizing mean response time in batch processing system. In: Asano T, *et al.*, eds. COCOON'99. LNCS 1627, Springer-Verlag, 1999. 231–240.

[5] DengX, Feng H, Zhang P, Zhao H. A polynomial time approximation scheme for minimizing total completion time of unbounded batch scheduling. In: Eades P, Takaoka T, eds. ISAAC 2001. LNCS 2223, Springer-Verlag, 2001. 26–35.

[6] Li S, Li G, Zhao H. A linear time approximation scheme for minimizing total weighted completion time of unbounded batch scheduling. OR Trans, 2004,8(4):27–32.

[7] Afrati F, Evripidis, Chekuri C, Karger D, Kenyon C, Khanna S, Milis I, Queyranne M, Skutella M, Stein C, Sviridenko M. Approximation schemes for minimizing average weighted completion time with release dates. In: Proc. of the 40th Annual Symp. on Foundations of Computer Science. New York, 1999. 32–43.

LI Shu-Guang was born in 1970. He is a Ph.D. candidate at the School of Math. & Sys. Sci., Shandong University. His current research areas are combinatorial optimization and theoretical computer science.

LI Guo-Jun was born in 1958. He is a professor and doctoral supervisor of School of Math. & Sys. Sci., Shandong University. His current research areas are graph theory, combinatorial optimization and theoretical computer science.

WANG Xiu-Hong was born in 1964. She is a professor of School of Math. & Info., Ludong University. Her current research areas are theoretical computer science and computer networks.