Note

A note on minimizing makespan on a single batch processing machine with nonidentical job sizes

Ali Husseinzadeh Kashan, Behrooz Karimi *, S.M.T. Fatemi Ghomi

Department of Industrial Engineering, Amirkabir University of Technology, P.O.Box: 15875-4413, Tehran, Iran

A R T I C L E   I N F O

Article history:
Received 26 October 2008
Received in revised form 18 February 2009
Accepted 19 February 2009
Communicated by D.-Z. Du

Keywords:
Scheduling
Batch processing machine
Worst-case ratio
Makespan

A B S T R A C T

In a relatively recent paper (G. Zhang, X. Cai, C.Y. Lee, C.K. Wong, Minimizing makespan on a single batch processing machine with nonidentical job sizes, Naval Research Logistics 48 (2001) 226–240), authors considered minimizing makespan on a single batching machine having unit capacity. For the restricted version of the problem in which the processing times of the jobs with sizes greater than 1/2 are not less than those of jobs with sizes not greater than 1/2, they proposed an \(O(n \log n)\) algorithm with absolute worst-case ratio 3/2. We propose an algorithm with absolute worst-case ratio 3/2 and asymptotic worst-case ratio \((m + 1)/m (m \geq 2\) and integer) for a more general version in which the processing times of the jobs of sizes greater than \(1/m\) are not less than the remaining (the case of \(m = 2\) has been considered by Zhang et al.). This general assumption is particularly held for those problem instances in which the job sizes and job processing times are agreeable. We obtain an \(O(n \log n)\) algorithm with asymptotic worst-case ratio 4/3 for these problems leading to a more dependable algorithm than that of Zhang et al.

© 2009 Elsevier B.V. All rights reserved.

1. Introduction

We study the problem of scheduling jobs with nonidentical sizes on a single batch processing machine (BPM) with unit capacity. We are given a list \(J\) of \(n\) jobs each of which has a processing time \(p_i\) and a size \(s_i \in (0, 1]\), and a single batch processing machine. The machine can simultaneously process a number of jobs as a batch as long as the total size of jobs in the batch is not greater than the capacity of the machine. The processing time of batch \(k\), i.e., \(P_k\) is given by the longest job in the batch. No preemption is allowed. The goal is to schedule the given jobs as batches on the machine to minimize the makespan. After batch construction, the order in which the batches meet the machine is arbitrary. For simplicity we use CMAX to denote this problem. For more details on BPM scheduling models readers may refer to a recently published review paper [3].

We address the optimal value of makespan by \(C^*\). By \(C^A\) we denote the makespan value obtained by algorithm \(A\). The subscript \(Q\) in \(C^*_Q\) or \(C^A_Q\) denotes the optimal makespan or the makespan obtained by algorithm \(A\) that is relevant to the job set \(Q\). By \(C(\pi)\) we denote the makespan value of a batching configuration \(\pi\).

Given an instance \(I\) of CMAX, the absolute worst-case ratio of algorithm \(A\) is given by

\[R_A = \sup_I \{C^A_I / C^*_I\}\].

The asymptotic worst-case ratio of algorithm \(A\) is defined to be

\[R_A^\infty = \lim_{\nu \to \infty} \sup_I \{C^A_I / C^*_I \mid C^*_I \geq \nu\}\].

* Corresponding author. Tel.: +98 21 66413034; fax: +98 21 66413025.
E-mail addresses: a.kashani@aut.ac.ir (A. Husseinzadeh Kashan), B.Karimi@aut.ac.ir (B. Karimi), Fatemi@aut.ac.ir (S.M.T. Fatemi Ghomi).

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.02.014
2. Generalization of Zhang et al., proportional assumptions

Under proportional assumption, Zhang et al. [5] considered instances of CMAX in which there are two types of jobs: large jobs (jobs of size greater than 1/2) and small jobs. They further assume that the processing time of each large job is not less than that of small jobs, and provided an \( O(n \log n) \) algorithm, called split and rearrange (SR) with worst-case ratio 3/2 (for the sake of consistency, we address this algorithm as SR2).

We investigate the generalization of the proportional assumption in [5]. Under the generalized assumption, there are two types of jobs: the first type \( (T_m^1) \) is jobs with size greater than \( 1/m \) (\( m \geq 2 \) and integer) and the second type \( (T_m^2) \) is the remaining jobs. We further assume that the processing time of a job of type \( T_m^1 \) is not less than the processing time of a job of type \( T_m^2 \). The case of \( m = 2 \) is the situation investigated by Zhang et al. [5]. They believe that the proportional assumption is reasonable since in many cases a large job will require more work, which will result in a longer processing time.

For a given integer \( m \geq 2 \) we address instances of CMAX satisfying the generalized assumption as CMAX-m. Also we denote a relaxed version of CMAX-m in which splitting jobs of type \( T_m^2 \) is allowed as CMAX-m-s. The resulting parts of a split job are called segments (the processing time of a job segment is equal to the original job processing time).

A practical situation in which our generalized assumption is verifiable is the case of having agreeable job sizes and job processing times, i.e., \( s_i < s_j \) implies \( p_i \leq p_j \). In this situation a larger job needs more or equal processing time in comparison with the smaller job. This situation exactly satisfies the generalized assumption. For a given problem in which job sizes and job processing times are agreeable, the generalized assumption is held, independent of the value of \( m \).

In the following, we first provide an algorithm called \( A_m \) which takes an integer \( m \geq 2 \) and an instance of CMAX-m as input.

Algorithm \( A_m \). Step 1. Obtain the optimal batching configuration for the set of jobs of type \( T_m^1 \). Arrange all batches in decreasing order of their processing time.

Step 2. Re-index the jobs of type \( T_m^2 \) in non-decreasing order of their processing time. Starting at the head of the list, place the first job into the lowest indexed non-full batch. If the batch has not enough room, place part of the job into the batch such that it is completely full. Then put the remaining part of the job in the next lowest indexed non-full batch and continue in this way (In the case when there is no non-full batch, open a new batch). Repeat this step for the remaining jobs in the list.

A simple example to show how the algorithm works can be found in [5] for the case of \( m = 2 \). In the following, we prove that \( A_m \) returns the optimal solution for CMAX-m-s problem.

Lemma 1. Algorithm \( A_m \) solves all instances of CMAX-m-s, optimally.

Proof. When there is no new batch constructed at Step 2 of \( A_m \), the proof is trivial. Otherwise, it is clear that there must be some optimal schedules for CMAX-m-s in which at most one batch is not completely full. Because we can always fill a batch as full as possible unless there is no job left any longer. Let \( \pi \) be the batching solution obtained by \( A_m \) and \( \delta \) be any arbitrary feasible batching solution for CMAX-m-s. Also let \( k_\delta \) be the set of batches constructed at Step 1 of \( A_m \). By \( k_\delta \) we denote the set of batches in \( \delta \) that contain a job of type \( T_m^1 \). In a finite number of job interchanges, we can reconstruct the batches of \( k_\delta \) in a way that there are batches containing jobs of type \( T_m^1 \) similar to batches in \( k_\delta \). Assuming \( s_i \geq s_j \), when interchanging job \( i \) (a job of type \( T_m^1 \) in batch \( B_n \)) with job \( j \) (a job of type \( T_m^1 \) in batch \( B_v \)) we assign job \( i \) to \( B_n \) and assign job \( j \) to \( B_v \). Considering capacity restriction, we may also pick a set of jobs of type \( T_m^2 \) (or their segments) from \( B_n \) with total size \( s_i - s_j \) and assign them to \( B_n \) whenever necessary. We may also need to pick only a job \( i \) and assign it to one of existing or newly opened batch. Let us denote the batching solution obtained after interchange process by \( \delta' \). It is not difficult to see that \( C(\pi) \leq C(\delta') \). Because through interchange process we construct batches similar to batches of \( k_\delta \), their sum of processing times is less than the total processing times of batches in \( k_\delta \) (recall that the result of Step 1 of \( A_m \) is optimal and the processing time of all batches in both \( k_\delta \) and \( \delta_\pi \) is determined by a job of type \( T_m^1 \)).

Algorithm \( A_m \) provides a lower bound for the optimal makespan of CMAX-m.

Proof. Since any optimal batching solution for CMAX-m is also a feasible batching for CMAX-m-s, the result follows immediately.
Now we provide the algorithm $\text{SR}_m$ for $\text{CMA}\text{X}-m$. The motivation of our algorithm is based on the idea behind the algorithm $\text{SR}_2$, that is: solving $\text{CMA}\text{X}-m$ by $A_n$, then moving out all jobs of type $T_2$ split in size and assigning them to some new batches. Let us define $\pi$ as the optimal batching solution for $\text{CMA}\text{X}-m$ provided by algorithm $A_m$. Assume that $\pi$ results in $K$ batches $B_1, \ldots, B_K$. Clearly, $P_1 \geq P_2 \geq \cdots \geq P_K$. Let $q$ be the number of split jobs resulted by $A_n$. We have $q \leq K - 1$. For $f = 1, 2, \ldots, q$ let us denote by $j_f$, a job split in size, whose first part is in $B_f$ (1 $\leq i_1 \leq \cdots \leq i_q \leq K - 1$). The processing time of this job is denoted by $t_f$. The formal description of $\text{SR}_m$ is as follows.

**Algorithm $\text{SR}_m$.** Get the optimal batching configuration $\pi$ for $\text{CMA}\text{X}-m$ by algorithm $A_m$. Remove all split jobs from $\pi$. For $f = 1, 2, \ldots, \lfloor q/m \rfloor$, put jobs $j_{inf-(m-1)}$, $j_{inf-(m-2)}$, $\ldots$, $j_{inf}$ together in a batch with processing time $t_{inf-(m-1)}$. Then get a batching configuration $\pi'$ with $K + \lfloor q/m \rfloor$ batches.

The following Lemma builds the main result on the worst-case performance of $\text{SR}_m$.

**Lemma 3.** Given an instance of $\text{CMA}\text{X}-m$, we have $\frac{C_{\text{SR}_m}}{C^*} \leq \frac{m + 1}{m} + \frac{(m - 2)p_1'}{mc^*}$.

**Proof.** The batching configuration resulted by $\text{SR}_m$ consists of two parts, one is the $K$ batches left by moving all split jobs and the other is the $\lfloor q/m \rfloor$ batches by repacking split jobs. The completion time of the first part (i.e., $C_1$) is less than $\sum_{k=1}^K P_k$. It was also proven by Lemma 2 that $\sum_{k=1}^K P_k$ is less than $C^*$. The completion time of the second part (i.e., $C_2$) is equal to $\sum_{f=1}^{\lfloor q/m \rfloor} P_{inf-(m-1)}$. By a lemma (Lemma 2, page 230) from [5] we have $P_{inf-(m-1)} \leq P_{inf-(m-1)} + 1$. Also for $f = 2, \ldots, \lfloor q/m \rfloor$ we have $P_{inf-(m-1)} - 1 \leq P_{inf-(m-1)} - 1$, $P_{inf-(m-1)} - 2 \leq P_{inf-(m-1)} - 2$, $\ldots$, $P_{inf-(m-1)} + 1 \leq P_{inf-(m-1)} + 1$.

Thus we get:

\[ C_{\text{SR}_m} = C_1 + C_2 \leq \sum_{f=1}^K P_f + \sum_{f=1}^{\lfloor q/m \rfloor} P_f + \sum_{f=1}^{\lfloor q/m \rfloor} (P_{inf-(m-1)} - 2) + \cdots + P_{inf-(m-1)} + P_{inf-(m-1)} + 1/m \]

\[ \leq \frac{(m - 2)p_1'}{m} + \frac{m + 1}{m} \sum_{f=1}^K P_f \]

We conclude that $\frac{C_{\text{SR}_m}}{C^*} \leq \frac{m + 1}{m} + \frac{(m - 2)p_1'}{mc^*}$.

To show that the inequality is tight, consider an instance with $2m - 2$ jobs of size $1/m$ and another job of size $3/2m$. Also all jobs have the same processing time $p$ (note that such a choice satisfies the proportional assumption). Regardless of the value of $m$, the optimal makespan for this instance is $C^* = 2p$. Applying $\text{SR}_m$ results in a batching configuration with $C_{\text{SR}_m} = 3p$. Substituting both values into the inequality reveals that it is tight. \(\square\)

**Theorem 1.** Given an instance of $\text{CMA}\text{X}-m$, we have $R_{\text{SR}_m} = 3/2$ and $R_{\text{SR}_m}^\infty = (m + 1)/m$.

**Proof.** It is easy to see that $\frac{3}{2} \geq \frac{m + 1}{m} + \frac{(m - 2)p_1'}{mc^*}$, for any integer $m \geq 2$. As we showed in Lemma 3, this inequality is tight yielding $R_{\text{SR}_m} = 3/2$. The ratio $C_{\text{SR}_m}/C^*$ tends asymptotically to $(m + 1)/m$ as $n$ (and consequently $C^*$) tends to infinity resulting $R_{\text{SR}_m}^\infty = (m + 1)/m$. \(\square\)

The complexity of executing $\text{SR}_m$ is dominated by the complexity of determining the optimal batching at Step 1 of $A_m$. Unfortunately, due to NP-hardness of determining the optimal batching in Step 1 of $A_n$ it is difficult to do Step 1 of $A_m$ efficiently for $m \geq 4$ (three-dimensional matching can be reduced to an special case of $\text{CMA}\text{X}-4$ where all jobs have identical processing times).

The case of $m = 2$ (investigated in [5]) is trivial. Here, all jobs of type $T_2$ have a size greater than $1/2$. Therefore, the optimal batching in Step 1 of $A_2$ is determined in $O(n)$ time by assigning each of the large jobs to a single batch. In the following we show that the case of $m = 3$ is also tractable leading to an algorithm with asymptotic worst-case ratio $4/3$ (algorithm $\text{SR}_3$).

Step 1 of $A_3$ needs to solve efficiently an instance of $\text{CMA}\text{X}$ in which all jobs have a size greater than $1/3$. Let us address this problem by $\text{CMA}\text{X}/3$. For this problem any feasible batch should accommodate at most two jobs. Let us represent a given instance of $\text{CMA}\text{X}/3$ by a simple graph. For this purpose, let a node stands for each job. There is an edge between nodes (jobs) $i$ and $j$ if they could be accommodated in a batch, that is $s_i + s_j \leq 1$. By definition all jobs having size greater than $1/2$ form a *stable set* in the graph since they share no edge. Also all jobs of size not greater than $1/2$ constitute a *clique* since they are mutually adjacent. In this way, the representative graph is a *split graph*. One can see that finding the optimal
solution of a given instance of CMAX/1/3 is equal to finding a partitioning of its corresponding graph with cliques of size not greater than 2, having minimum total processing times of all cliques (The size of a clique is the number of nodes it contains). Indeed each clique could be seen as a batch. This problem is solvable based on finding the maximum weight matching of the split graph [1]. The following algorithm executes the first step of A3 optimally in $O(|X|^{2.5})$ time where, $X$ is the set of jobs of type $T_1$.

Step 1. Construct the valued version of the split graph where each edge is valued by $\min(p_i, p_j)$ if the edge is incident to the vertices $i$ and $j$.

Step 2. Find a maximum weight matching in the graph.

Step 3. Form the batches as follows:
- For each set of the matching, process the corresponding two jobs in the same batch,
- Other jobs are processed as single job batches.

Step 4. $C^*_X$ is the sum of the processing time of all jobs in $X$, minus the value of the maximum weight matching.

**Corollary 1.** Given an instance of CMAX-2, SR2 solves the problem approximately in $O(n \log n)$ time with absolute worst-case ratio 3/2.

**Corollary 2.** Given an instance of CMAX-3, SR3 solves the problem approximately in $O(n^{2.5})$ with absolute and asymptotic worst-case ratio 3/2 and 4/3, respectively.

**Corollary 3.** The optimal solution of an instance of CMAX in which each job has a size greater than $1/3$ can be obtained in $O(n^{2.5})$.

### 3. A special case of CMAX with agreeable job sizes and job processing times

As we pointed out before, all instances of CMAX with agreeable job sizes and job processing times (i.e., $s_i < s_j$ implies $p_i \leq p_j$) exactly satisfy our generalized assumption for any integer $m \geq 2$. This means that we can apply either SR2 or SR3 (or in general the class of SRm) on these problems. Since then, we use the term CMAX-agreeable to denote an instance of CMAX with agreeable job sizes and job processing times.

Because the only tractable cases are $m = 2$ and $m = 3$, our focus is mainly on SR2 and SR3. Given an instance of CMAX-agreeable, the main motivation of this section is to show that the time complexity of SR3 is now $O(n \log n)$ (a modification on the $O(n^{2.5})$ time stated in Corollary 2). This reflects that both SR2 and SR3 are similar in terms of time complexity. However in terms of the worst-case performance, SR3 may be more dependable.

Let us consider an instance of CMAX-agreeable, in which all jobs have a size greater than $1/3$. In other words there are only jobs of type $T_1$. Let us partition the set of jobs into two sets, namely $L_1$ and $L_2$. $L_1$ denotes the set of jobs having their size greater than $1/2$ and $L_2$ denotes the set of jobs having their size in $(1/3, 1/2]$. Doing Step 1 of A3, clearly each job in $L_1$ requests for a separate batch. Each job in $L_2$ can be grouped either with one job of $L_1$ or with one job of $L_2$. Let $U_2$ be the set of leftover jobs from $L_2$ that are not batched with any job from $L_1$ in a feasible batching configuration. Let us define $V_2 = L_2 \cup U_2$ as the complement of $U_2$. Clearly every two jobs in $U_2$ should be packed in one batch. Hence, to do Step 1 of A3 optimally it is sufficient to devise a procedure that returns the minimal $U_2$ or maximal $V_2$. In the case of optimal batching configuration we use $U_2^*$ and $V_2^*$. Following [2] we say $V$ is maximal if for all set $V'$ we have $|V| \geq |V'|$ and $p_i \geq p_i'$ for $1 \leq i \leq |V'|$ (assume jobs in $V$ and $V'$ are sorted based on decreasing order of their processing times). By analogy we may define the minimal set $U$. Clearly $U_2^*$ is minimal and hence $V_2^*$ is maximal. In the following we prove that the simple FFLPT algorithm [4] produces a set $V_2$ that is maximal and hence is an optimal algorithm for doing Step 1 of A3. For simplicity we use $V_2^*$ to denote the set $V_2$ resulted by FFLPT algorithm.

**Algorithm FFLPT.** Step 1. Arrange the jobs in decreasing order of their processing times (in the case of equal processing times arrange jobs based on decreasing order of their sizes).

Step 2. Select the job at the head of the list and place it in the first batch with enough space to accommodate it. If it fits in no existing batch, create a new batch. Repeat step 2 until all jobs have been assigned to a batch.

Executing Step 1 of FFLPT algorithm on a given instance of CMAX-agreeable with only jobs of type $T_1$, the longest is also the largest. Hence, FFLPT algorithm batches the longest (and hence largest) remaining job in $L_2$ with the longest (largest) possible job remaining in $L_1$. Intuitively FFLPT algorithm is driven by $L_2$ since each job in $L_1$ should be placed in a separate batch.

**Lemma 4.** Given an instance of CMAX-agreeable, in which all jobs have a size greater than $1/3$ (only jobs of type $T_1$), $V_2^*$ is maximal.

**Proof.** Let $V_2$ (as defined formerly) be relevant to any batching solution of jobs. We prove that $V_2^*$ is maximal over $V_2$ (assume that $V_2$ is obtained by an arbitrary algorithm, say A). Let $j_i$ and $j_i$ denote the $i$th job in $V_2^*$ and $V_2$, respectively. Recall that jobs in $V_2^*$ and $V_2$ are arranged based on Step 1 of FFLPT algorithm. The proof is based on induction.

**Basis ($i = 1$):** If $j_1$ is the longest job in $V_2^*$, none of the longer (or larger) jobs in $L_2$ could have been placed in a batch containing a job of $L_1$. Thus $j_1$ is the longest possible job that can be placed in the same batch with a job of $L_1$, leading to $p_{j_1} \geq p_{j_1}$.
Inductive step: Assuming \( p_{j_i} \geq p_{j_k} \), \( \forall 1 \leq i \leq k \) and that \( V_2 \) has \( (k + 1) \) th job, we show that \( p_{j_{k+1}} \geq p_{j_{k+1}} \). Assuming contradiction \( p_{j_{k+1}} < p_{j_{k+1}} \), by hypothesis we have \( p_k \geq p_{j_k} \geq p_{j_{k+1}} > p_{j_{k+1}} \). Algorithm A put job \( j_{k+1} \) in some batch containing a job of \( L_1 \), but FFLPT cannot, because if this is the case it would have put job \( j_{k+1} \) and not \( j_{k+1} \). At this point all jobs of \( L_1 \) that have no neighbor in their batch must be larger than \( 1 - s_{j_k} \). Since the first \( k \) jobs of \( V_2 ^k \) are not smaller than \( j_{k+1} \), each job of \( L_1 \) that is paired by one of the first \( k \) jobs of \( V_2 ^k \) must have a size not greater than \( 1 - s_{j_{k+1}} \). This implies \( L_1 \) contains exactly \( k \) jobs with sizes not greater than \( 1 - s_{j_{k+1}} \). Let \( Y_k \) denotes the set of these jobs. But the first \( k \) jobs in \( V_2 \) are not smaller than \( j_{k+1} \) and these jobs must be paired with jobs of \( V_k \). Thus, there is no job in \( L_1 \) small enough to be paired with \( j_{k+1} \). So algorithm A cannot puts \( j_{k+1} \) in some batch containing a job of \( L_1 \), thus leading to a contradiction.

If \( |V_2 ^k| < |V_2| \), then \( V_2 \) has at least \( |V_2 ^k| + 1 \) jobs. \( V_2 ^k \) cannot include job \( j_{|V_2 ^k|+1} \). However, the above induction shows that \( V_2 \) cannot include job \( j_{|V_2 ^k|+1} \) too, leading to a contradiction. Thus \( |V_2 ^k| \geq |V_2| \). □

**Theorem 2.** Given an instance of CMAX-agreeable, \( SR_2 \) solves the problem approximately in \( O(n \log n) \) time with absolute worst-case ratio \( 3/2 \) while, \( SR_3 \) solves the problem approximately in \( O(n \log n) \) time with absolute and asymptotic worst-case ratio \( 3/2 \) and \( 4/3 \), respectively.

**Proof.** The statement about \( SR_2 \) is trivial. By Lemma 4, we proved that FFLPT optimally batches jobs when they are only of type \( T_1 ^1 \). In this sense, the first step of \( A_3 \) can be done optimally in \( O(n \log n) \) time by FFLPT algorithm. Since the second step of \( A_3 \) can also be done in \( O(n \log n) \) time, the whole complexity of \( A_3 \) is \( O(n \log n) \) leading to a time complexity \( O(n \log n) \) for \( SR_3 \).

Although it may seem that \( SR_3 \) intuitively should produce a schedule with smaller makespan than \( SR_2 \) for a given instance with agreeable job sizes and job processing times, our finding proves that this is not the case for all instances of CMAX-agreeable. Indeed, for a simple 5 jobs instance wherein \( s_1 = 0.8; s_2 = 0.7; s_3 = 0.6; s_4 = 0.5; s_5 = 0.3 \), the schedule obtained by \( SR_3 \) consists of five batches where each job is sit in a separate batch while, the schedule constructed by \( SR_2 \) consists of four batches where jobs 3 and 5 share the same batch and each of the three remaining jobs sits in a separate batch. Disregarding the value of job processing times, \( SR_2 \) always reports a smaller makespan than \( SR_3 \) for this instance.

**Acknowledgements**

The authors would like to acknowledge the Iran National Science Foundation (INFS) for the financial support of this work.

**References**


