Scheduling jobs with equal-processing-time on parallel machines with non-identical capacities to minimize makespan

Jun-Qiang Wang a,b,*, Joseph Y.-T. Leung c

a Performance Analysis Center of Production and Operations Systems (PacPos), Northwestern Polytechnical University, Xi’an 710072, PR China
b Key Laboratory of Contemporary Design and Integrated Manufacturing Technology, Ministry of Education, Northwestern Polytechnical University, Xi’an, Shaanxi 710072, PR China
c Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, United States

A R T I C L E   I N F O

Article history:
Received 27 August 2013
Accepted 27 May 2014
Available online 28 June 2014

Keywords:
Batch machines
Makespan
NP-hard
Approximation algorithm
Absolute worst-case ratio
Asymptotic worst-case ratio

A B S T R A C T

We consider the problem of scheduling a set of equal-processing-time jobs with arbitrary job sizes on a set of batch machines with different capacities. A job can only be assigned to a machine whose capacity is not smaller than the size of the job. Our goal is to minimize the schedule length (makespan). We show that there is no polynomial-time approximation algorithm with an absolute worst-case ratio less than 2, unless \( P=NP \). We then give a polynomial-time approximation algorithm with an absolute worst-case ratio exactly 2. Moreover, we give a polynomial-time approximation algorithm with asymptotic worst-case ratio no more than 3/2. Finally, we perform a computational experiment and show that our approximation algorithm performs very well in practice.

1. Introduction

In a parallel batch machine, several jobs can be processed simultaneously as a batch. Parallel batch machines are encountered in many industries such as semiconductor industry, metal industry, pharmaceutical industry, etc., see Mathirajan and Sivakumar (2006) for more information. As an example, consider the burn-in operations in a semiconductor company, in which integrated circuits are put into an oven for an extended period of time. Each oven (machine) has a finite capacity. Each integrated circuit (job) has a size and a specified burn-in time. Several circuits can be grouped together and put into an oven at the same time, provided that the total size of the circuits does not exceed the capacity of the oven. The processing time of a batch is the largest burn-in time among all the circuits in the batch. The problem is to group the integrated circuits into batches and assign the batches to the ovens so as to minimize the makespan.

We are given a set of \( n \) equal-processing-time jobs, \( J = \{j_1, j_2, \ldots, j_n\} \). Each job \( j_i \) has a size \( s_i \) and a processing time of 1 unit. There is a set of \( m \) machines, \( M = \{M_1, M_2, \ldots, M_m\} \). Each machine \( M_i \) has a capacity \( C_i \). Our problem is to group the jobs into batches and assign the batches to the machines so as to minimize the makespan. Note that batch \( B_k \) can be assigned to machine \( M_i \) if and only if the total size of all the jobs in \( B_k \) does not exceed \( C_i \). This problem is strongly NP-hard even for one machine, since it reduces to the bin packing problem which is known to be strongly NP-hard (Garey and Johnson, 1979).

Suppose the \( m \) machines have \( \ell \) distinct capacities, \( K_1 < K_2 < \cdots < K_\ell \). Thus, \( C_i \in \{K_1, K_2, \ldots, K_\ell\} \) for each \( 1 \leq i \leq m \). Without loss of generality, we may assume that the machines are sorted in ascending order of their capacities; i.e., \( C_1 \leq C_2 \leq \cdots \leq C_m \). Furthermore, we assume that \( s_j \leq K_\ell \) for all \( 1 \leq j \leq n \). Define \( a_0 \) to be 1 and \( b_1 \) to be the largest index such that \( C_1 = C_2 = \cdots = C_{b_1} = K_1 \). For each \( 1 < k \leq \ell \), define \( a_k \) to be \( b_{k-1} + 1 \) and \( b_k \) to be the largest index such that \( C_1 = C_2 = \cdots = C_{b_k} = K_k \). Let \( J_1 = \{j_j \mid s_j \leq K_1\} \), and for each \( 1 < k \leq \ell \), let \( J_k = \{j_j \mid s_j < K_k \leq s_j \leq K_{k-1}\} \). Thus, \( J = J_1 \cup J_2 \cup \cdots \cup J_\ell \). Note that \( J_\ell \) could be an empty set for some \( k \). Furthermore, since the size of each job in \( J_k \) is greater than the capacity of each machine in \( \{M_1, M_2, \ldots, M_{b_k}\} \), a job in \( J_k \) can only be assigned to a machine in \( \{M_{b_k}, M_{b_k+1}, \ldots, M_m\} \), but not to any machine in \( \{M_1, M_2, \ldots, M_{b_k}\} \). Fig. 1 illustrates the problem.

Fig. 2 shows an instance of the scheduling problem along with a schedule. Fig. 2(a) shows the input data and Fig. 2(b) shows a schedule. In this example, there are 12 jobs and five machines. The jobs have the same processing time (1 unit). The sizes of the jobs and the capacities of the machines are shown in Fig. 2(a). Fig. 2(b) shows a feasible schedule. As shown in Fig. 2(b), jobs \( J_2 \) and \( J_3 \) are batched together and assigned to machine \( M_1 \). The total size of jobs \( J_2 \) and \( J_3 \) is 9, which is less than the capacity of machine \( M_1 \). Similarly, jobs \( J_5 \) and \( J_6 \) are batched together and assigned to machine \( M_2 \). Since the total size of jobs \( J_2 \) and \( J_3 \) is 19, it is less than
the capacity of machine $M_2$ and hence it is a feasible assignment. The makespan of the schedule is 2. It is not difficult to show that the schedule is an optimum schedule.

Because of the computational complexity of our problem, we shall be studying polynomial-time approximation algorithms. In the following we will define absolute worst-case performance ratio and asymptotic worst-case performance ratio. For a given instance $I$ and algorithm $A$, let $A(I)$ be the makespan obtained when algorithm $A$ is applied to $I$; let $OPT(I)$ denote the optimum makespan of $I$; and let $R_A(I) = A(I)/OPT(I)$. The absolute worst-case performance ratio $R_A$ for algorithm $A$ is defined as

$$R_A = \inf \{ r \geq 1 : R_A(I) \leq r \text{ for all } I \}$$

The asymptotic worst-case performance ratio $R_A^\infty$ is defined as

$$R_A^\infty = \inf \{ r \geq 1 : \text{for some } N > 0, R_A(I) \leq r \text{ for all } I \text{ with } OPT(I) \geq N \}$$

In this paper, we will show that there is no polynomial-time approximation algorithm $A$ with $R_A < 2$, unless $P=NP$. We then give a polynomial-time approximation Algorithm A1 such that $R_A^1 = 2$. Furthermore, we give another polynomial-time approximation Algorithm A2 such that $R_A^2 \leq 3/2$. Finally, we perform a computational study and show that Algorithm A2 performs very well in practice.

The problem we are going to study occurs naturally in the real world. For example, consider a semiconductor company where integrated circuits are subject to burn-in operations. The company has a number of ovens with different capacities. Some ovens are older models and have smaller capacities, while others are newer models and have larger capacities. The integrated circuits have different sizes and similar burn-in times. Since the integrated circuits have similar burn-in times, we may assume that their processing times are equal. Moreover, our results indicate that even this special case is difficult to approximate; i.e., it does not have a polynomial-time approximation algorithm with an absolute worst-case performance ratio less than 2.

The organization of the paper is as follows. In the next section, we will review the literature related to our problem. In Section 3, we will give the main theoretical results: (1) there is no polynomial-time approximation algorithm $A$ with $R_A < 2$; (2) there is a polynomial-time approximation Algorithm A1 with $R_A^1 = 2$; and (3) there is a polynomial-time approximation Algorithm A2 with $R_A^2 \leq 3/2$. In Section 4, we will describe an experiment and show that Algorithm A2 performs very well in practice. Finally, we draw some concluding remarks in Section 5.

**Fig. 1.** Problem description.

**Fig. 2.** An example illustrating the scheduling problem. (a) Input data and (b) A schedule.
2. Literature review

Scheduling jobs on batch machines have been studied since the 1980s. In this section we will review only those papers that deal with makespan minimization. Initially, researchers studied a single batch machine. Ikura and Gimple (1986) are probably the first to study the scheduling of a single batch machine; they provided an $O(n^2)$ algorithm to determine whether a feasible schedule (i.e., one where all jobs are completed by their due dates) exists for a set of equal-processing-time jobs, where release times and due dates are agreeable (i.e., $r_i \leq r_j$ implies $d_i \leq d_j$). If a feasible schedule exists, the algorithm will find a schedule with the minimum makespan.

Uzsoy (1994) considered the scheduling on a single batch machine of a set of jobs with arbitrary processing times and arbitrary job sizes so as to minimize the makespan and the total completion time. He showed that both problems are NP-hard. Motivated by the complexity of the problem, he proposed a number of heuristics and performed computational experiments to demonstrate the effectiveness of the heuristics. No worst-case bounds were derived for his heuristics. Later, Zhang et al. (2001) analyzed the worst-case ratios of the heuristics proposed by Uzsoy. They showed that some of the heuristics have unbounded worst-case ratio, while others have worst-case ratio at most 2. They gave an approximation algorithm with worst-case ratio of 3/2 for the special case where the processing times of large jobs (i.e., jobs with sizes greater than 1/2) are not less than those of small jobs (i.e., jobs with sizes not greater than 1/2). Finally, they gave an approximation algorithm with worst-case ratio of 7/4 for the general case.

Kashan et al. (2009) generalize the 3/2-approximation algorithm of Zhang et al. to a $(m+1)/m$-approximation algorithm for a more general version in which the processing times of the jobs with sizes greater than $1/m$ are not less than the processing times of the remaining jobs.

Li et al. (2005) considered the problem of scheduling a set of jobs with release times and arbitrary job sizes on a single batch machine; their objective is to minimize the makespan. They gave an approximate algorithm with worst-case ratio of $2+\epsilon$ and running time $O(n \log n + f(1/\epsilon))$.

Dupont and Dhaenens-Flipo (2002) developed a branch-and-bound procedure for minimizing makespan. Finally, a number of meta-heuristics have been proposed for the single batch machine. Kashan et al. (2006) and Damodaran et al. (2006) proposed genetic algorithms, while Melouk et al. (2004) proposed a simulated annealing method for minimizing the makespan.

Recently, there are some interests in scheduling jobs on parallel batch machines. Initially, the studies are concentrated on batch machines with identical capacities, say B. Lee et al. (1992) proposed an approximation algorithm, BLPT, for a set of jobs with identical job sizes, say 1 unit. Algorithm BLPT works by sorting the jobs in descending order of their processing times. Then the first B jobs are assigned to the first batch, the next B jobs are assigned to the next batch, and so on. After the batches are formed, they are sorted in descending order of the batch processing times, and assigned to the machines as they become free for assignment. They showed that the worst-case ratio is no more than $4/3 - 1/(3m)$.

Koh et al. (2004) studied the problem with arbitrary job sizes and incompatible job families. Jobs belonging to the same family have the same processing time, while jobs belonging to different families have different processing times. A number of deterministic algorithms and genetic algorithms were proposed, and computational studies were conducted to study the effectiveness of the heuristics.

Chang et al. (2004) proposed a simulated annealing algorithm for a set of jobs with arbitrary processing times and arbitrary job sizes. Computational experiments were conducted. The result of the simulated annealing algorithm was compared with the solution obtained by CPLEX, and found to be favorable.

Kashan et al. (2008) proposed a hybrid genetic algorithm; the results of the algorithm were analyzed empirically.

Damodaran and Chang (2008) proposed four deterministic algorithms for a set of jobs with arbitrary processing times and arbitrary job sizes. First, jobs are batched by the First-Fit-Decreasing (FFD) or Best-Fit-Decreasing (BFD) rule (these rules are designed for the bin packing problem). Then the batches are assigned to the machines by either the Largest-Processing-Time (LPT) rule or the Multi-fit (MF) rule. Thus, we have four heuristics: FFD-LPT, FFD-MF, BFD-LPT and BFD-MF. These four heuristics were compared with CPLEX and a simulated annealing algorithm. It was found that the four heuristics were as good as (or better than) the one obtained by CPLEX and the simulated annealing algorithms. Moreover, the four heuristics run much faster than CPLEX and the simulated annealing algorithms.

Very recently, there are a couple of papers dealing with parallel batch machines with non-identical capacities. Xu and Bean (2007) gave a simulated annealing algorithm while Damodaran et al. (2012) gave a particle swarm algorithm. They both compared their algorithms against the one obtained by CPLEX, and found that their algorithms are better.

Our scheduling problem is also related to the problem of scheduling jobs with inclusive processing set restriction; see the survey paper by Leung and Li (2008) for more details. In this problem, each job $j_i$ has a processing time $p_i$ and a machine index $a_i$. Job $j_i$ can be processed by any one of the machines $M_1$, $M_2$, ..., $M_m$. $\ldots M_m$, but not by any of the machines $M_1$, $M_2$, ..., $M_m$. Ou et al. (2008) have proposed a heuristic to minimize makespan; their heuristic has a worst-case ratio of $4/3$.

3. The main results

We first show that there is no polynomial-time approximation algorithm $A$ with $R_A < 2$, unless $P=NP$. We shall show that if there were such an algorithm, then we can solve the 3-Partition problem in polynomial time. This is a contradiction, since 3-Partition is strongly NP-complete (Garey and Johnson, 1979) and hence cannot be solved in polynomial time unless $P=NP$. The 3-Partition problem is defined as follows.

3-Partition: Given a list $L = (a_1, a_2, \ldots, a_{3m})$ of 3m integers such that $\sum_{i=1}^{3m} a_i = mB$ and $B/4 < a_i < B/2$ for each $1 \leq j \leq 3m$, is there a partition of $L$ into $m$ sublists $L_1, L_2, \ldots, L_m$ such that $\sum_{i \in L_k} a_i = B$ for each $1 \leq k \leq m$?

Note that if the given instance $L$ has a solution, then each $L_k$, $1 \leq k \leq m$, has exactly three integers. This follows from the restrictions on the size of $a_i$. The following theorem is our first result.

Theorem 1. There is no polynomial-time approximation algorithm $A$ with $R_A < 2$, unless $P=NP$.

Proof. By contradiction. Suppose there is a polynomial-time approximation algorithm $A$ with $R_A < 2$. We will show that there is a polynomial-time algorithm $A'$ to solve the 3-Partition problem. Given an instance $L = (a_1, a_2, \ldots, a_{3m})$ of the 3-Partition problem, $A'$ will construct an instance of the scheduling problem as follows. There are $m$ machines $\{M_1, M_2, \ldots, M_m\}$ with $C_i = B$ for each $1 \leq i \leq m$, and $3m$ jobs $\{j_1, j_2, \ldots, j_{3m}\}$ with $s_i = a_i$ for each $1 \leq j \leq 3m$. The processing time of each job is 1 unit. Algorithm $A'$ will then call Algorithm $A$ to construct a schedule for the instance of scheduling problem.

We claim that $A$ will return a schedule with makespan equal to 1 if and only if the 3-Partition problem has a solution. Suppose the 3-Partition problem has a solution. Then the optimal makespan will be 1. Since $R_A < 2$, algorithm $A$ will return a schedule with makespan less than 2. Since the schedule must have integral makespan, algorithm $A$ will return a schedule with makespan equal to 1. Conversely, if the 3-Partition problem does not have a
solution, then the optimal makespan must be at least 2. Since algorithm $A$ is an approximation algorithm, it will return a schedule with makespan not better than the optimal makespan. Hence, algorithm $A$ will return a schedule with makespan at least 2. Thus, algorithm $A$ can be used to solve the 3-Partition problem in polynomial time. This is the contradiction we seek since 3-Partition is strongly NP-complete.

Note that Theorem 1 holds even if all machines have the same capacity. We now give an algorithm, $A1$, with absolute worst-case performance ratio exactly 2. We assume that $J$ has been split into $J1, J2, \ldots, J_r$. Moreover, we assume that for each $1 \leq k \leq r$, $J_k$ is treated as a list of jobs; the list is in arbitrary order. Furthermore, we assume that we have computed $a_k$ and $b_k$ for all $1 \leq k \leq r$. A job assigned to a machine occupies a time slot. In general, there could be several jobs assigned to the same time slot on the same machine because these jobs are batched together. In the algorithm given below, we use $v$ to represent time slot.

**Algorithm A1** starts by assigning jobs to the first time slot across all the machines. First, it takes the jobs in $J_1$ and assigns them to the list $A$. Then it schedules the jobs in $A$ one at a time, to the first time slot of machines $M_1, M_2, \ldots, M_{r\ell}$, during the assignment, if a job $j_i$ exceeds the capacity of a machine, say $M_k$, then $j_i$ is split into two portions. The first portion stays in the first time slot of $M_k$ and the total size of all jobs in this time slot is exactly $C_k$. We also mark this portion of $j_i$. The second portion of $j_i$ will be put back as the first job in A. We continue this assignment until either $A$ becomes empty or we reach machine $b_1 + 1$, whichever occurs first. Then, we concatenate $J_2$ with $A$ and assign the jobs in $A$ to the first time slot of machines $M_1, M_2, \ldots, M_{r\ell}$ in the same manner as before. We repeat this process until we assign jobs in $J_r$.

After we finish assigning jobs in the first time slot, if $A$ is not empty, then we continue to assign jobs in the second time slot. We repeat this process until all jobs have been assigned. Note that the schedule obtained so far is not a feasible schedule, since some jobs may be split into two parts and put into different batches. To obtain a feasible schedule, we take all those split jobs and assign each split job in a new time slot. The final schedule will be a feasible schedule. Below is the description of Algorithm A1.

**Algorithm A1.**

1. $v = 0$.
2. If $J_1 \parallel J_2 \parallel \ldots \parallel J_r = \emptyset$, then goto Step 14. $v = 0$.
3. $v = v + 1$.
4. $k = k + 1$.
5. If $k > r$, then goto Step 12.
6. $A = J_1 \parallel A$.
7. $i = a_k$.
8. If $i > b_k$, then goto Step 4.
9. If $A = \emptyset$, then goto Step 4.
10. Take the first job $j_1$ from $A$ and assign it to a batch $B_1$ in time slot $v$ of machine $M_k$.
11. If the total size of all jobs in $B_1$ is less than $C_k$, then goto Step 9.
   Else if the total size of all jobs in $B_1$ is equal to $C_k$, then $i = i + 1$ and goto Step 8.
   Else remove a portion of $j_1$ from $B_1$ so that total size of all jobs in $B_1$ is exactly $C_k$. Mark this portion of $j_1$ in $B_2$. Put the removed portion of $j_1$ as the first job in $A$. $i = i + 1$ and goto Step 8.
12. Split $A$ into $J_1, J_2, \ldots, J_r$ and goto Step 2.
14. For $i$ from 1 to $m$ do
   If there is a marked job $j_i$ in $M_k$, then take $j_i$ in $M_k$, and the other parts of $j_i$ and assign them to a new time slot in $M_k$. Update $v$.
15. Return $A1(\mathcal{J}) = v$.

Let us illustrate Algorithm A1 by means of the set of jobs given in Fig. 2(a). In this example, $a_1 = b_1 = 1$, $a_2 = 2$ and $b_2 = 3$, $a_3 = 4$ and $b_3 = 5$. We have $K_1 = 10$, $K_2 = 20$ and $K_3 = 25$. The jobs are grouped in three lists: $J_1 = (j_1, j_2, j_3, j_4, j_5)$, $J_2 = (j_6, j_7, j_8, j_9)$, and $J_3 = (j_{10}, j_{11}, j_{12})$. Fig. 3(a) shows the schedule obtained after running Step 1 to Step 13 of Algorithm A1, and Fig. 3(b) shows the schedule obtained after running Step 14 of Algorithm A1.

In the first iteration, jobs are scheduled in the first time slot across all machines. At the beginning, $A$ is set to be $A = (j_1, j_2, j_3, j_4, j_5)$. Jobs $j_1$, $j_2$ and $j_3$ are assigned to the first time slot of $M_1$. When $j_3$ is assigned, the total size exceeds $C_1$. Therefore, $j_3$ is split into two parts, the first part has size 3 and the second part has size 2. The first part of $j_3$ is marked and put in $M_1$ while the second part is put back as the first job of $A$. In the next step, we concatenate $J_2$ with $A$ and hence $A = (j_1, j_2, j_3, j_4, j_5)$. Then $j_6$ and $j_7$ are scheduled on $M_2$. When $j_7$ is scheduled, it exceeds the capacity of $M_2$. Therefore, $j_7$ is split into two parts, the first part has size 6 and the second part has size 8. The first part is marked and assigned to $M_2$ while the second part is put back into $A$. Next, $j_1$ and $j_8$ are scheduled on $M_3$. When $j_8$ is scheduled, it exceeds the capacity of $M_3$. Thus, $j_8$ is again split into two parts. After one iteration, we have $A = (j_{12}, j_6, j_9, j_{13}, j_{14}, j_{15})$.

In the next iteration, we split $A$ into three sublists, $J_1 = (j_{13}, j_{14}, j_{15})$, $J_2 = (j_6, j_9)$, and $J_3 = (j_{12})$. These jobs can all be assigned to the second time slot, as Fig. 2(a) shows. Note that the makespan at the present time is 2, which is a lower bound of the optimal makespan. Then we execute Step 14 of Algorithm A1. Starting from $M_1$, we examine each machine and try to put the split job into a new time slot. In the first time slot, we have $j_3$ being the split job. Hence we take the two parts of $j_3$ and assign them to a new time slot. Similarly, in the second time slot, $j_5$ is the split job and we put the two parts of $j_5$ into a new time slot. Continuing in this manner, we obtain the final schedule as shown in Fig. 3(b); its makespan is 4.
Theorem 2. \( R_{A1} = 2 \).

Proof. Let \( S \) be the schedule obtained after running Step 1 to Step 13 of Algorithm A1, and \( S' \) be the schedule obtained after running Step 14 of Algorithm A1. The schedule \( S \) is a relaxation of the original problem, where jobs can be split. Thus, the makespan of \( S \) is a lower bound of the optimal makespan. In Step 14 of Algorithm A1, we take each split job in \( S \) and assign it to a new time slot. Since each time slot in \( S \) has at most one split job, \( S' \) will have makespan at most twice that of \( S \) and hence at most twice the lower bound of the optimal makespan. Therefore, we have \( R_{A1} = 2 \). \( \square \)

We now present another approximation Algorithm A2, and prove that \( R_{A2} \leq 3/2 \). As in Algorithm A1, we assume that \( a_k \) and \( b_k \) have been computed for each \( 1 \leq k \leq \ell \). Furthermore, the jobs have been split into \( \ell \) sublists, \( J_1, J_2, \ldots, J_{\ell} \). We assume that the jobs in \( J_{\ell} \), \( 1 \leq k \leq \ell \), have been sorted in descending order of their sizes. We use \( v \) to represent time slot.

Algorithm A2 starts by assigning jobs to the first time slot across all machines. First, it takes the jobs in \( J \) and assigns them to the list \( A \). Then it schedules the jobs in \( J \) on machine \( M_1 \). The list \( A \) is scanned from the first job until the last job, and for each job encountered in the scan, the job will be scheduled on machine \( M_1 \) if \( C_1 \) is not exceeded; otherwise, the job will stay in the list \( A \). Then the jobs in \( A \) will be assigned to the next machine \( M_2 \). We continue this assignment until either \( A \) becomes empty or we reach machine \( M_m \). At that time, we concatenate \( J_2 \) to \( A \) and starts assigning jobs to machine \( M_2 \). This process is repeated until all jobs have been assigned. Below is a description of Algorithm A2.

Algorithm A2.

1. \( v = 0 \).
2. If \( J_1 \cup J_2 \cup \ldots \cup J_\ell = \emptyset \) then goto Step 15.
3. \( k = 0; A = \{ \} \); \( v = v + 1 \).
4. \( k = k + 1 \).
5. If \( k > \ell \) then goto Step 13.
6. \( A = J_k \cup A \).
7. \( l = 0 \).
8. If \( l > b_k \) then goto Step 4.
9. If \( A = ( \) then goto Step 4.
10. Scan \( A \) from the first job until the last job. For each job \( J_i \) encountered in the scan do
    a. If job \( J_i \) can be scheduled in time slot \( v \) in \( M_i \) without exceeding \( C_i \) then (schedule \( J_i \) in time slot \( v \) of \( M_i \) ) else (leave \( J_i \) in \( A \) )
    b. \( l = l + 1 \).
12. Split \( A \) into \( J_1, J_2, \ldots, J_\ell \). For each \( 1 \leq k \leq \ell \), the jobs in \( J_k \) are sorted in descending order of their processing times.
14. Return \( A(I) = v \).

Let us illustrate Algorithm A2 by means of the set of jobs given in Fig. 2(a). The jobs are in three sublists: \( J_1 = \{ J_3, J_4, J_5, J_6 \} \), \( J_2 = \{ J_8, J_9, J_{10}, J_{11} \} \), and \( J_3 = \{ J_{12}, J_{13}, J_{14}, J_{15} \} \). Fig. 2(b) shows the schedule obtained by Algorithm A2.

In the first iteration, \( A \) is set to be \( J_1 \). Jobs in \( A \) are assigned to machine \( M_1 \). Note that after \( J_5 \) is assigned to \( M_1 \), none of the jobs in \( J_4, J_3, J_2 \) can be assigned to \( M_1 \) since the capacity of \( M_1 \) would be exceeded. Only job \( J_1 \) can be assigned to \( M_1 \). In the next step, we concatenate \( J_2 \), and hence \( A = \{ J_3, J_4, J_5, J_6, J_8 \} \). Then \( J_5 \) is assigned to \( M_2 \) and \( J_6 \) is assigned to \( M_3 \). After that, we concatenate \( J_3 \) to \( A \), and hence \( A = \{ J_3, J_4, J_5, J_6, J_8, J_9, J_{10} \} \). Continuing in this manner, we finally obtain the schedule as shown in Fig. 2(b). Note that Algorithm A2 obtains an optimal schedule for this set of jobs.

Lemma 1. Let \( v \) be any time slot. Suppose \( J_i \) is the first job assigned to the time slot \( v \) on machine \( M_m \), \( 1 \leq k \leq \ell \). Then, for any job \( J_d \) assigned to the time slot \( v \) on any machine between \( M_m \) and \( M_b \), we have \( s_d \geq s_g \).

Proof. Before we assign \( J_d \) to the time slot \( v \) on machine \( M_m \), we concatenate \( J_k \) to \( A \). If \( J_k \neq \emptyset \), then \( J_d \) has the largest size among all jobs in \( J_k \) since \( J_k \) is in the descending order of their sizes. Moreover, any job in \( J_k \) has a size larger than any job in \( J_j \), \( 1 \leq j < k \). Therefore, any job assigned after \( J_k \) would have size less than or equal to \( s_g \). If \( J_k = \emptyset \), then the first job \( J_i \) in \( A \) has the largest size among all jobs in \( A \) and hence \( s_c \geq s_d \) for any job \( J_d \) assigned after \( J_i \). \( \square \)

Our next theorem shows that Algorithm A2 has an asymptotic worst-case ratio no more than \( 3/2 \).

Theorem 3. For any instance \( I \) and any set of machines \( M \), we have
\[
A(I) \leq \frac{3}{2} \cdot \text{OPT}(I) + 1
\]
Hence, we have \( R_{A2} \leq 3/2 \).

Proof. We will prove the theorem by contradiction. Let \( I \) be the smallest (in terms of the number of jobs) counterexample with set of jobs \( J \). We want to characterize the nature of such a counterexample. Let \( J_i \) be the job with the largest size among all jobs scheduled in the time slot \( A(I) \). If there are several jobs with the same size, then choose the job scheduled on the lowest indexed machine, say \( M_i \). By Lemma 1, \( J_i \) must be scheduled on machine \( M_i \) for some \( 1 \leq k \leq \ell \); i.e., \( M_i = M_{k-1} \). We now claim that \( k = 1 \). Suppose not. Then \( k > 1 \). If there were some jobs scheduled in the time slot \( A(I) \) on any machine in \( \{ M_1, M_2, \ldots, M_{k-1} \} \), then we can delete these jobs from \( I \) to obtain \( I' \). Now, \( A(I') = A(I) \) and \( \text{OPT}(I') \leq \text{OPT}(I) \). Thus, \( I' \) is a smaller counterexample, contradicting our assumption that \( I \) is the smallest counterexample. Therefore, there is no job scheduled on machines \( M_1, M_2, \ldots, M_{k-1} \) in the time slot \( A(I) \). Clearly, \( J_i \notin J_k \). If we now delete the jobs in \( J_{k-1} \cup J_k \cup \ldots \cup J_{k-1} \) from \( I \) to obtain \( I'' \), then \( A(I'') = A(I) \) and \( \text{OPT}(I'') \leq \text{OPT}(I) \). Thus, \( I'' \) is a smaller counterexample, contradicting our assumption that \( I \) is the smallest counterexample. Therefore, \( k = 1 \).

Next, \( J_i \) is scheduled in the time slot \( A(I) \) on machine \( M_1 \). We claim that there is no job, other than \( J_i \), scheduled in the time slot \( A(I) \) on any machine. If there were, we can delete them to obtain a smaller counterexample, contradicting our assumption that \( I \) is the smallest counterexample. Moreover, we may assume that every job in \( I \) has size at least \( s_j \); otherwise, we can delete those jobs with a smaller size than \( s_j \) to obtain a smaller counterexample. Thus, \( J_i \) has the smallest size among all jobs in \( I \).

We now consider the following three cases separately.

Case 1: \( s_j \leq \frac{1}{3} K_1 \).

Since \( K_1 < K_2 < \cdots < K_{\ell} \), we have \( s_j \leq \frac{1}{3} K_1 \) for all \( 1 \leq k \leq \ell \). Since we have tried to schedule \( J_i \) in a time slot earlier than \( A(I) \) on each machine, it is clear that each machine has been assigned jobs with total size larger than \( \frac{1}{3} \) of its capacity in any time slot earlier than \( A(I) \). Thus, we have
\[
\text{OPT}(I) \geq \frac{2}{3} (A(I) - 1)
\]
Or equivalently,

$$A2(\mathcal{I}_3) \leq \frac{2}{3} \text{OPT}(\mathcal{I}_3) + 1$$

Case II: $s_j > \frac{1}{3} K_f$ for all $1 \leq j \leq \ell$.

In this case, each machine has either one job or two jobs scheduled in any time slot less than $A2(\mathcal{I}_3)$. It is clear that an optimal schedule cannot schedule the jobs any better than this. Hence, $A2(\mathcal{I}_3) = \text{OPT}(\mathcal{I}_3)$, contradicting our assumption that $\mathcal{I}_3$ is a counterexample.

Case III: There is an index $k$, $1 \leq k < \ell$, such that $s_j > \frac{1}{3} K_f$ for all $1 \leq j \leq k$ and $s_j = \frac{1}{3} K_f$ for all $k < j < \ell$.

For the machines $M_1, M_2, \ldots, M_\ell$, each machine has either one job or two jobs scheduled in any time slot less than $A2(\mathcal{I}_3)$. Thus, an optimal schedule cannot schedule the jobs any better than this. For the machines $M_{\ell+1}$ to $M_m$, each machine is filled with jobs in each time slot with total size at least $\frac{1}{3}$ of the capacity of the machine. Thus, we can resort to the argument in Case I to show that

$$A2(\mathcal{I}_3) \leq \frac{2}{3} \text{OPT}(\mathcal{I}_3) + 1. \quad \Box$$

4. Computational results

Computational experiments are conducted to evaluate the performance of Algorithm A2 in practice. First we run Step 1 to Step 13 of Algorithm A1 to obtain a lower bound for a given instance $\mathcal{I}$. $A1(\mathcal{I})$ represents the value of a lower bound when Algorithm A1 is applied to $\mathcal{I}$, $A2(\mathcal{I})$ represents the makespan obtained by Algorithm A2. We use the ratio $R$ to evaluate the performance of Algorithm A2. The $R$ is defined as follows:

$$R = \frac{A2(\mathcal{I})}{A1(\mathcal{I})}$$

Algorithm A2 is programmed in MATLAB version 7.10.0 and run on Intel(R) Pentium(R) CPU G640 @ 2.80 GHz with 2GB RAM under Windows 7 environment.

We set the total number of machines at 10, i.e. $m = 10$. The machines have three distinct capacities, i.e. $f = 3$. According to practical experience, new machines capacity is approximately 2.5 times as large as older models. Specifically, we choose $K_1 = 10$, $K_2 = 25$ and $K_3 = 65$.

Considering the fact that machines with bigger capacity are more expensive, we choose a smaller number of machines with bigger capacity. Thus, we choose five machines with capacity $K_1$, three machines with capacity $K_2$ and two machines with capacity $K_3$. Jobs in the first group can be assigned to all 10 machines. Jobs in the second group can be assigned to only five machines, while jobs in the third group can be assigned to only two machines. Thus, if the total number of jobs is $n$, the numbers of jobs in $J_1, J_2$ and $J_3$ are chosen to be $\frac{2}{3} n, \frac{2}{3} n$ and $\frac{1}{3} n$, respectively.

The determination of the sizes of the jobs in the experiment is a key issue. If we use uniform distribution or normal distribution to generate job sizes, it has the same probability for obtaining the small size and the big size. Actually, we expect that there are more small jobs than big jobs in each group. Only then can Algorithm A2 assign more jobs with small size to other bigger group, which will make the scheduling process more complicated. Finally, the sizes of the jobs in the experiment are generated by Poisson distribution, $s_j \sim P(\lambda)$, $\lambda = K_1/2, J_j \in J_\ell$. Here, $\lambda_1 = 5$, $\lambda_2 = 12.5$, $\lambda_3 = 32.5$. Considering the bound of job size corresponding to the capacity, $s_j$ is set as follows:

$$s_j = \begin{cases} K_{\ell-1} & s_j < K_{\ell-1} \\ s_j' & K_{\ell-1} \leq s_j' \leq K_{\ell} \\ s_j \sim P(\lambda) & K_0 = 0 \\ K_k & s_j' > K_k \end{cases}$$

Furthermore, we select 70% data from the interval of $(K_{\ell-1}, \lambda]$ and 30% data from the interval of $(\lambda, K_\ell]$ to ensure that there are a lot of small size jobs to fill out the machine with a bigger capacity.

Finally, we start with 450 jobs and increase by 45 jobs every time, until we reach 1305 jobs. Thus, we have 20 different numbers of jobs. For each number of jobs, we randomly generate 100 instances and run 100 times. For each instance, we calculate the ratio $R$. At the end, we find the best ratio, the worst ratio and the average ratio for the 100 instances. The computational results are shown in Fig. 4. The average of the worst ratio of all 2000 instances is 1.159585, while the averages of the average ratio and the best ratio are 1.07313 and 1.018765, respectively. Considering the fact that we are comparing the performance of Algorithm A2 against a lower bound, our results show that Algorithm A2 performs very well in practice.

5. Concluding remark

In this paper we have studied the problem of scheduling a set of equal-processing-time jobs on a set of machines with different capacities, with the objective of minimizing the makespan. We showed that there is no polynomial-time approximation algorithm with an absolute worst-case ratio less than 2. We gave a polynomial-time Algorithm A1 with an absolute worst-case ratio exactly 2. Moreover, we gave a polynomial-time Algorithm A2 with an asymptotic worst-case ratio no more than 3/2. Finally, we conducted an experiment and showed that Algorithm A2 performs very well in practice. The average ratio of Algorithm A2 when compared with a lower bound is approximately 1.07. This is quite far away from the asymptotic worst-case ratio of 1.5.

As mentioned in Section 1, when there is only one machine, our problem becomes the classical one-dimensional bin packing problem (Garey and Johnson, 1979). Algorithm A2 then becomes the First-Fit-Decreasing (FFD) rule that has been studied for the bin packing problem. It is known that the asymptotic worst-case ratio of the FFD rule is $11/9$ for the bin packing problem (Johnson, 1974). Thus, the asymptotic worst-case ratio of Algorithm A2 lies between $11/9$ and $3/2$. For future research, it will be interesting to determine the exact value of the asymptotic worst-case ratio of Algorithm A2.

Another interesting problem is to study the general problem where jobs have arbitrary processing times. Designing a good approximation algorithm for the general problem does not seem to be easy, and proving the asymptotic worst-case bound appears to be a formidable task.

Finally, there are other scheduling objectives that are totally unexplored. For example, minimizing total completion time ($\sum C_j$) would be an interesting problem to study.

![Fig. 4. Computational results of 2000 instances.](image-url)
Acknowledgements

This work was supported by the National Natural Science Foundation of China (Grant No.51275421).

References


