Optim Lett (2018) 12:251–263
https://doi.org/10.1007/s11590-017-1191-0

ORIGINAL PAPER

Online C-benevolent job scheduling on multiple machines

Ge Yu[1] · **Sheldon H. Jacobson**[2]

Received: 15 March 2017 / Accepted: 29 August 2017 / Published online: 8 September 2017

Abstract We consider scheduling a sequence of C-benevolent jobs on multiple homogeneous machines. For two machines, we propose a 2-competitive Cooperative Greedy algorithm and provide a lower bound of 2 for the competitive ratio of any deterministic online scheduling algorithms on two machines. For multiple machines, we propose a Pairing-m Greedy algorithm, which is deterministic 2-competitive for even number of machines and randomized $(2 + 2/m)$-competitive for odd number of machines. We provide a lower bound of 1.436 for the competitive ratio of any deterministic online scheduling algorithms on three machines, which is the best known lower bound for competitive ratios of deterministic scheduling algorithms on three machines.

Keywords Online scheduling · C-benevolent jobs · Approximation algorithm

1 Introduction

The interval scheduling problem arises in many real-life applications, such as online keyword auctions, memory caching, and time-constrained scheduling [9]. For an interval scheduling problem, there is a sequence of jobs to be scheduled on a single machine or multiple machines. Each job consists of the following characteristics: (a) an arrival time; (b) an execution time; (c) a deadline; (d) a value, the reward to be obtained

✉ Ge Yu
geyu3@illinois.edu

1 Department of Electrical and Computer Engineering, University of Illinois at Urbana Champaign, Urbana, IL 61801, USA

2 Department of Computer Science, University of Illinois at Urbana Champaign, Urbana, IL 61801, USA

after a job is completed. We study the case where the deadline of a job is equal to the sum of the arrival time and the execution time of the job, and hence, each job can be represented by an interval along the time axis.

The interval scheduling problem and its many variations have been extensively studied. The offline interval scheduling problem on non-identical machines is NP-complete [1]. Other results for the offline interval scheduling problem can be found in [2,7,8,15,16]. For the online scheduling problem for unweighted jobs, Faigle and Nawijn [10] proposes a simple Greedy algorithm, GOL, which is optimal for scheduling unweighted jobs on multiple identical machines. The general online interval scheduling problem for weighted jobs does not have any approximation algorithms with finite worst case guarantees [20]. Additional assumptions are needed in order to obtain algorithms with finite competitive ratios. For a thorough review on results for variations of interval scheduling problems, see [14].

One such example is given by [20], who considers scheduling *f-benevolent* jobs on a single machine. The values and the execution times of f-benevolent jobs are subject to a fixed function f, called *f-benevolent cases*. The C-benevolent jobs (see Definition 2) we consider in this paper is a subclass of f-benevolent jobs. A Greedy algorithm, HEU, is provided and proven to be 4-competitive for C-benevolent jobs. This HEU algorithm achieves the best possible competitive ratio among deterministic algorithms for scheduling C-benevolent jobs on a single machine. Lipton and Tomkins [17] proposes a 2-competitive algorithm for scheduling jobs with only two possible execution times and job values proportional to execution times. For job sequences with arbitrary execution times, a $\Omega((\log \Delta)^{1+\epsilon})$-competitive algorithm is provided, where Δ is the ratio of the longest to shortest execution times. Baruah et al. [3] proposes a 4-competitive algorithm on a single machine, where job values are assumed to be proportional to execution times.

Another widely used technique to reduce the competitive ratios of online scheduling algorithms is *randomization*. Seiden [19] first proposes a randomized $(2 + \sqrt{3})$-competitive algorithm for scheduling C-benevolent jobs on a single machine. Miyazawa and Erlebach [18] proposes a 3-competitive algorithm on a single machine for *monotone instances*, where the order of right points of job intervals coincides with the order of left points of jobs and job values are non-decreasing. Fung et al. [12] proposes a barely random 3.5822-competitive algorithm for equal-execution-time jobs with arbitrary values. Fung et al. [13] proposes 2-competitive barely random algorithms for equal-execution-time and C-benevolent job sequences, respectively.

The general online interval scheduling problem on multiple machines is more complicated than scheduling on a single machine, and hence, there does not exist an approximation algorithm with finite worst-case guarantees [5]. The problem of scheduling on two machines has been extensively studied over the years. Baruah et al. [3] proposes a cooperative 2-competitive algorithm on two identical machines for jobs with values uniformly proportional to execution times. Our Cooperative Greedy algorithm for two machines is inspired by their algorithm. Fung et al. [12] proposes a 3.5822-competitive algorithm on two machines for jobs with equal execution times and arbitrary values. They provide a lower bound of 4/3 (2) for scheduling C-benevolent jobs on multiple (two) machines. As for scheduling on more than two machines, Epstein et al. [5] proposes a 4-competitive Greedy algorithm, ALG, for scheduling

C-benevolent jobs on multiple *uniformly related machines* (i.e., each machine has a service speed). Fung et al. [11] proposes a 2(2 + 2/(2m − 1))-competitive algorithm for scheduling equal-execution-time jobs on even (odd) number of machines. Our proposed algorithm follows a similar idea as [11], but we handle C-benevolent jobs with variable execution times. The best lower bound for competitive ratios of randomized algorithms for scheduling C-benevolent jobs on multiple machines is $1 + \ln 2$ [6], which is proven for scheduling C-benevolent jobs on a single machine. Fung et al. [11] point out that the lower bound of $1 + \ln 2$ also applies to multiple homogeneous machines. We provide a lower bound for competitive ratios of deterministic algorithms on two and three machines, respectively, using a different approach.

This paper considers scheduling a sequence of C-benevolent jobs on multiple machines, generalizing the problem proposed by [20] on a single machine. Machines are treated as homogeneous. For two machines, we propose a 2-competitive Cooperative Greedy algorithm. We further generalize the algorithm to multiple machines and propose a Pairing-m algorithm, which is deterministic 2-competitive for even number of machines and randomized (2+2/m)-competitive for odd number of machines. The Pairing-m algorithm improves the 4-competitive algorithm given by [5]. We provide a lower bound of 2 and 1.436 for the competitive ratio of any deterministic online scheduling algorithms on two and three machines, respectively. Therefore, the Cooperative Greedy algorithm achieves the best possible competitive ratio for scheduling C-benevolent jobs on two machines.

The paper is organized as follows. Section 2 clarifies the basic variable definitions and problem formulations for the online interval scheduling problem considered in this paper. Section 3 provides the Cooperative Greedy algorithm for scheduling on two machines. Section 4 proposes a Pairing-m Greedy algorithm for scheduling on multiple machines. Section 5 provides a lower bound of 2 and 1.436 for competitive ratios of any deterministic online algorithms for scheduling on two and three machines, respectively. Section 6 summarizes the paper with future research directions.

2 Preliminary

Consider a fixed set of machines available for executing jobs. Let m denote the total number of machines. A sequence of N (unknown) jobs will arrive, one after another, to be scheduled on one of the available machines. Let $\mathbf{I} = \{J_1, \ldots, J_N\}$ denote the list of arriving jobs, where J_i represents the ith arriving job. The scheduling of a job to a machine is referred to as an *assignment*. One machine can execute at most one job at a time and one job can be assigned to at most one machine. Each assignment should be made *online* (i.e., the assignment is made immediately when a job arrives without knowledge of jobs arriving in the future). The assignment is *preemptive*, which means a job assigned to a machine can be terminated later in favor of a later arriving job, but the terminated job cannot be reassigned. In this case, we say the terminated job is *aborted*. A machine that is executing some job is said to be *busy*; otherwise, it is said to be *available*.

A *job vector* is revealed upon each job arrival. Let $J_i = (a_i, l_i, v_i)$ denote the job vector of the ith job arrival, for $i = 1, 2, \ldots, N$. For each job vector, a_i denotes the *arrival time* of the ith job, l_i denotes the *execution time* of the ith job, and v_i denotes the *value* of the ith job. Therefore, if a job is assigned to a machine, the *completion time*, is given by $f_i = a_i + l_i$. Moreover, we refer to the interval defined by $[a_i, f_i)$ as the *job interval*. We assume that no two jobs share the same arrival time. If two jobs J_{i_1} and J_{i_2} satisfy $[a_{i_1}, f_{i_1}) \bigcap [a_{i_2}, f_{i_2}) \neq \emptyset$, then J_{i_1} and J_{i_2} are said to *conflict* with each other.

The objective of this online scheduling problem is to maximize the total value of completed jobs, subject to the constraint of the number of available machines.

Let $OPT(\mathbf{I})$ denote the value of the optimal schedule for a job instance $\mathbf{I}$, which is obtained with the complete knowledge of $\mathbf{I}$, and hence, is the *optimal off-line reward*. Let $A(\mathbf{I})$ denote the value obtained by algorithm $\mathcal{A}$ for a job instance $\mathbf{I}$. We employ the standard definition for competitive ratios in this work, as given in Definition 1.

Definition 1 An online algorithm $\mathcal{A}$ is said to have a competitive ratio of γ (i.e., γ-competitive) if $A(\mathbf{I}) \geq OPT(\mathbf{I})/\gamma$ for any job instance $\mathbf{I}$ generated by an adapted adversary.

By Definition 1, we must have $\gamma \geq 1$. We focus on *C-benevolent* jobs, as defined in Definition 2.

Definition 2 A job $J = (a, l, v)$ is a C-benevolent job if $v = f(l)$ and $f(l)$ is a positive, convex, strictly increasing and continuous function of l. In other words, the convexity property of C-benevolent function $f(l)$ implies that

$$f(a+\epsilon) + f(b-\epsilon) \leq f(a) + f(b),$$

for $0 < \epsilon \leq a \leq b$.

The class of C-benevolent jobs embraces many applications (see [5,13,20]), including the jobs with values proportional to execution times (see [3,4]).

3 Cooperative Greedy algorithm for two machines

This section considers two machines, which is the basic case for multiple machines. The analysis method used here provides insights for multiple machines, and this algorithm is later generalized to multiple machines in Sect. 4.

We propose a deterministic algorithm for two machines, referred to as the *Cooperative Greedy* algorithm. This algorithm is inspired by the 2-competitive algorithm given by [3], which is designed for jobs with values proportional to execution times (a special class of C-benevolent functions). Independent from our work, [13] uses the same idea in proposing a randomized 2-competitive algorithm for scheduling C-benevolent jobs on a single machine. The proof of Theorem 1 follows from the proof of *Theorem 3.3* [13], and hence, is omitted here.

Theorem 1 *The Cooperative Greedy algorithm is 2-competitive for scheduling C-benevolent jobs on two machines.*

Cooperative Greedy Algorithm

```
Arbitrarily pick one machine as the primary machine (PM) and the other one as the secondary machine (SM).
for all job intervals $J_i$ in an instance I do
    if PM has just completed executing some job
      Switch the role of PM and SM.
      if PM is not executing any job
        Assign $J_i$ to PM, and let $J_i$ be executed till it is completed.
      else
        Assign $J_i$ to SM temporarily.
      endif
    elseif PM is executing some job while SM is not executing any job
      Assign $J_i$ to SM temporarily.
    elseif PM is executing some job and SM is also executing some job $J_j$
      Abort and assign $J_i$ to SM only if $v_i > v_j$.
    endif
end for
```

4 Pairing-m algorithm for multiple machines

This section considers scheduling C-benevolent jobs on multiple machines (i.e., $m \geq 3$). We generalize the Cooperative Greedy algorithm for two machines to even and odd number of machines, respectively. The algorithm is referred to as the *Pairing-m* algorithm. We show that the Pairing-m algorithm is 2-competitive for even number of machines and (2 + 2/m)-competitive for odd number of machines. For even number of machines, the Pairing-m algorithm is deterministic. For odd number of machines, the Pairing-m algorithm is randomized.

4.1 Pairing-m algorithm for even number of machines

Let $m = 2k$, where $k \in \mathbb{Z}^+$. The Pairing-m algorithm works in a similar way as the Cooperative Greedy algorithm. It dynamically pairs up m machines, k machines as the primary machines (PM) and the other as the secondary machines (SM). The pairing between machines is not fixed and changed at the completion of jobs assigned on PMs. First, we divide the time axis into *sections*: the time interval starting from one of PMs gets assigned a job till the first time all machines are available. Therefore, the time axis can be divided into non-overlapping sections with no job arriving between two successive sections. We describe how the Pairing-m algorithm assign jobs in each section in the following.

Consider any section. In the beginning, we pick k machines as PMs and the other k machines as SMs arbitrarily. Suppose M_1 to M_k are PMs and M_{k+1} to M_{2k} are SMs. The Pairing-m algorithm starts by assigning arriving jobs to an available PM until all PMs are busy. If this never happens, then the Pairing-m algorithm completes all arrived jobs in this section, and hence, is the same as the optimal schedule. Otherwise, let $J_1, J_2, \ldots$ denote the jobs completed by the Pairing-m algorithm, indexed in the increasing order of completion times.

We further divide the section into non-overlapping *segments* based on the completion times of jobs assigned to PMs: the first segment is defined as the time interval between the start of the section and the completion time of the J_1; successive segments are defined as the time interval between two subsequent completion times of jobs assigned on PMs. Therefore, for instance, $[f(J_i), f(J_{i+1}))$ is a segment for $i = 1, 2, \ldots$. In each segment, jobs assigned on the k PMs will be guaranteed completion (remain un-preempted). If there is at least one available PM in the segment, then assign jobs arriving in this segment to these available PMs until all PMs are busy. If there is no available PM in the segment, then the k SMs greedily schedule jobs arriving in this segment: a newly arrived job J'_j is only scheduled on some SM if $v(J'_j) > \min v(J_{SM})$, where $\min v(J_{SM})$ is the minimum value of jobs executed on SMs when J'_j arrives (if a machine has no job assigned to it, we consider it as executing a virtual job of value zero and length zero). In this way, the k SMs will be assigned jobs with the top k values out of all the jobs arriving in the segment (if there are $k' < k$ jobs arriving in the segment, we consider another $k - k'$ virtual jobs with value zero and length zero arriving in this segment). At the beginning of each segment, two machines (one PM and one SM) switch their roles: the PM which just completes its assigned job becomes an available SM, and the SM assigned the largest-value job among all jobs currently executed on SMs becomes an PM. Therefore, the Pairing-m algorithm keeps k PMs and k SMs in each time segment. This process continues till all $2k$ machines are available again, which is the end of a section. When the next job arrives, a new section begins in the same way.

Theorem 2 *The Pairing-m algorithm is 2-competitive for scheduling C-benevolent jobs on even number of machines.*

Proof Since a job sequence can be divided into non-overlapping sections, we compute the competitive ratio of the Pairing-m algorithm in any section, which is the same as the competitive ratio of the Pairing-m algorithm for the job sequence.

Let $OPT(M_i, n)$ denote the optimal schedule on machine M_i (with abuse of notation, $OPT(M_i, n)$ also denotes the value for the optimal schedule depending on the context) when the Pairing-m algorithm completes n jobs in a section, for $i = 1, 2, \ldots, m$ and $n \geq 1$. Let $\{OPT_i(n)\}$ denote the order statistics of $\{OPT(M_i, n)\}$ such that $OPT_i(n) \leq OPT_{i-1}(n)$ for $i = 2, 3, \ldots, m$. Let $\{J_1, J_2, \ldots, J_n\}$ denote jobs completed by the Pairing-m algorithm, indexed in the increasing order of completion times with $f(J_1) \leq f(J_2) \leq \cdots \leq f(J_n)$. Note that the index of these jobs may not coincide with their arrival orders. Therefore, the completion time of the last job in the optimal schedule $OPT(M_i, n)$ is less than $f(J_n)$ for $i = 1, 2, \ldots, n$.

We prove $\sum_{i=1}^{k} OPT_i(n) \leq \sum_{j=1}^{n} v(J_j)$ by induction on the total number of jobs completed by the Pairing-m algorithm in a section. Consider the base case of $n \leq k$. Then, there are only k jobs arriving in this section, and all jobs are completed by PMs. Therefore, $\sum_{i=1}^{k} OPT_i(n) \leq \sum_{j=1}^{n} v(J_j)$ holds trivially.

Assume $\sum_{i=1}^{k} OPT_i(n) \leq \sum_{j=1}^{n} v(J_j)$ holds for some n. We want to show that $\sum_{i=1}^{k} OPT_i(n+1) \leq \sum_{j=1}^{n+1} v(J_j)$ holds. Consider the last completed job J_{n+1} by the Pairing-m algorithm. We compare the set of jobs arriving in this section $\mathbf{I}_{n+1}$ with $\{J_1, J_2, \ldots, J_{n+1}\}$ completed by the Pairing-m algorithm and another set of jobs arriving in this section $\mathbf{I}_n$ constructed as follows, with $\mathbf{I}_n \subset \mathbf{I}_{n+1}$. Let S_{n+1} denote the segment where J_{n+1} arrives and $\mathbf{I}_s$ denote the set of jobs that arrive in segment S_{n+1}.

Note that there is no job arriving after segment S_{n+1}. Then define $\mathbf{I}_n \triangleq \mathbf{I}_{n+1} \backslash \mathbf{I}_s$. The Pairing-m algorithm will complete at most n jobs for the set of jobs $\mathbf{I}_n$ in this section. Let $\{J_1', J_2', \ldots, J_n'\}$ denote the set of jobs (indexed in increasing order of completion times) completed by the Pairing-m algorithm for $\mathbf{I}_n$ (if the number of completed jobs is smaller than n, append virtual jobs with value zero and length zero to the end).

If $\mathbf{I}_s = \{J_{n+1}\}$, then $J_i = J_i'$ for $i = 1, 2, \ldots, n$. By the induction assumption,

$$\sum_{i=1}^{k} OPT_i(n+1) \leq \sum_{i=1}^{k} OPT_i(n) + v(J_{n+1}) \leq \sum_{j=1}^{n+1} v(J_j).$$

Otherwise, $\mathbf{I}_s$ contains more than one job. Let $\{K_1', K_2', \ldots, K_k'\} \subset \{J_1', J_2', \ldots, J_n'\}$ denote the k jobs assigned to the k SMs at the end of segment S_{n+1} for $\mathbf{I}_n$ (if some SM is available during this segment, then we consider this SM as executing a virtual job with value zero and length zero). Then $\{K_1', K_2', \ldots, K_k'\}$ will be completed by the Pairing-m algorithm for $\mathbf{I}_n$ since there is no more job arriving. However, for $\mathbf{I}_{n+1}$, these k SMs will be updated with jobs with the top k values out of $\mathbf{I}_s \bigcup \{K_1', K_2', \ldots, K_k'\}$. Let $\{K_1, K_2, \ldots, K_k\}$ denote the k jobs being executed on the k SMs at the end of segment S_{n+1} for $\mathbf{I}_{n+1}$. Then $J_{n+1} \in \{K_1, K_2, \ldots, K_k\}$. Consider $OPT_i(n+1)$, for $i = 1, 2, \ldots, k$. If there exists K_j such that $K_j \in OPT_i(n+1)$, then define $OPT_i'(n+1) \triangleq OPT_i(n+1) \backslash K_j$. Otherwise, if $OPT_i(n+1)$ does not contain any K_j, then if $OPT_i(n+1)$ does not schedule any job that conflicts with any K_j, then $OPT_i'(n+1) \triangleq OPT_i(n+1)$. Otherwise, suppose $OPT_i(n+1)$ schedules a set of jobs $\{H_j^i\}_{i=1}^{l_j}$ (indexed by i in the increasing order of completion times), which conflict with some K_j. Then, the completion time of $H_j^{l_j-1}$ is within the segment S_{n+1}, since $H_j^{l_j}$ arrives in the segment of S_{n+1}. By the scheduling policy of the Pairing-m algorithm, $v(H_j^i) < v(K_j)$ for any i. Define $OPT_i'(n+1) \triangleq OPT_i(n+1) \backslash H_j^{l_j}$. Consider the set of schedules $\{OPT_i'(n+1)\}_{i=1}^{k}$. Then the largest completion time of jobs in schedules $\{OPT_i'(n+1)\}_{i=1}^{k}$ is within S_{n+1}. Therefore, from the induction assumption,

$$\sum_{i=1}^{k} OPT_i'(n+1) \leq \sum_{j=1}^{n} v\left(J_j'\right) - \sum_{j=1}^{k} v\left(K_j'\right),$$

which follows from that the largest completion time of jobs in schedules $\{OPT_i'(n+1)\}_{i=1}^{k}$ is already covered by the job completed at the end of segment S_{n+1}, and the completion times of $\{K_1', K_2', \ldots, K_k'\}$ are all beyond S_{n+1}. Therefore,

$$\begin{aligned}\sum_{i=1}^{k} OPT_i(n+1) &\leq \sum_{i=1}^{k} OPT_i'(n+1) + \sum_{j=1}^{k} v(K_j) \\ &\leq \sum_{j=1}^{n} v\left(J_j'\right) - \sum_{j=1}^{k} v\left(K_j'\right) + \sum_{j=1}^{k} v(K_j) \leq \sum_{j=1}^{n+1} v(J_j),\end{aligned}$$

where: the first inequality follows from the construction of $\{OPT'_i(n+1)\}_{i=1}^k$; the last inequality follows from $\sum_{j=1}^k v(K'_j) \le \sum_{j=1}^k v(K_j)$ and $\{J_j\}_{j=1}^n \backslash \{K_i\}_{i=1}^k = \{J'_j\}_{j=1}^n \backslash \{K'_i\}_{i=1}^k$ from the construction of $\mathbf{I}_n$. □

4.2 Pairing-m algorithm for odd number of machines

When the number of machines is odd, the Pairing-m algorithm for even number of machines cannot be directly applied. To overcome this difficulty, we introduce randomization and generalize the Pairing-m algorithm for even number of machines to odd number of machines.

Let $m = 2k + 1$ for $k \in \mathbb{Z}^+$. We create a virtual machine, add it to the pool of real machines and treat this virtual machine the same as real machines. Then the Pairing-m algorithm can be applied to these $m + 1$ machines. Let $OPT(M_i, \mathbf{I})$ and $P(M_i, \mathbf{I})$ denote the optimal schedule and the schedule using the Pairing-(m + 1) algorithm on machine M_i for instance $\mathbf{I}$, respectively, for $i = 1, 2, \ldots, m+1$ (machine M_{m+1} is the virtual machine). Then arbitrarily pick m schedules out of $\{P(M_i, \mathbf{I})\}_{i=1}^{m+1}$ with equal probability, and schedule jobs to the m real machines according to these m selected schedules. This algorithm is referred to as the Pairing-m algorithm for odd-number of machines.

Theorem 3 *The Pairing-m algorithm is* $(2 + 2/m)$*-competitive for scheduling C-benevolent jobs on odd number of machines.*

Proof Let $\{OPT_i(\mathbf{I})\}$ denote the order statistics of $\{OPT(M_i, \mathbf{I})\}$, with $OPT_i(\mathbf{I}) \ge OPT_{i+1}(\mathbf{I})$ for $i = 1, 2, \ldots, m$ (with abuse of notation, $\{OPT(M_i, \mathbf{I})\}$ also denotes the value of the schedule depending on the context). Then from Theorem 2,

$$\sum_{i=1}^{k+1} OPT_i(\mathbf{I}) \le \sum_{i=1}^{m+1} v(P(M_i, \mathbf{I})),$$

where $v(P(M_i, \mathbf{I}))$ denotes the total value of completed jobs by schedule $P(M_i, \mathbf{I})$. Since the Pairing-m algorithm randomly selects m schedules from $\{P(M_i, \mathbf{I})\}_{i=1}^{m+1}$ with equal probability, then the expected reward using the Pairing-m algorithm, denoted by $R_m(\mathbf{I})$, is lower bounded by

$$R_m(\mathbf{I}) = \frac{m}{m+1} \sum_{i=1}^{m+1} v(P(M_i, \mathbf{I})) \ge \frac{m}{m+1} \sum_{i=1}^{k+1} OPT_i(\mathbf{I}).$$

The optimal reward on m machines for instance $\mathbf{I}$, denoted by $O_m(\mathbf{I})$, is upper bounded by

$$O_m(\mathbf{I}) \le \sum_{i=1}^{m+1} OPT_i(\mathbf{I}) \le 2 \sum_{i=1}^{k+1} OPT_i(\mathbf{I}).$$

Therefore, the competitive ratio for the Pairing-m algorithm on odd number machines is given by $2(m+1)/m = 2 + 2/m$. □

5 Lower bounds for deterministic algorithms

This section gives a lower bound of 2 and 1.436 for the competitive ratio of any deterministic algorithm for scheduling C-benevolent jobs on two and three machines, respectively. Since the Cooperative Greedy algorithm is 2-competitive for two machines, it is the best obtainable deterministic algorithm for scheduling C-benevolent jobs on two machines. From Theorem 3, the competitive ratio of the Pairing-m algorithm on three machines is $2 + 2/3 = 2.67$. Although there is still a gap between the competitive ratio of our proposed algorithm and the lower bound for three machines, the Pairing-m algorithm is the first-known 2.67-competitive randomized algorithm for scheduling C-benevolent jobs on three machines.

Theorem 4 gives a lower bound for any deterministic algorithms on two machines. The proof of Theorem 4 uses the same technique as the proof of *Theorem 6* [12], and hence, is omitted here.

Theorem 4 *No deterministic algorithm for C-benevolent jobs on two machines can achieve a competitive ratio lower than 2.*

Theorem 5 provides a lower bound for any deterministic algorithm for scheduling C-benevolent job sequences on three machines. We use a similar approach as that in [12], but we handle more complicated cases for three machines.

We use the *W-set*, the job set originally defined in [20] to prove this upper bound. A W-set of jobs is defined as a sequence of jobs that satisfy the following conditions: (a) jobs arrive in sequence but conflict with one another within the set; (b) the value of the first arriving job is set to be one; (c) the values of subsequent jobs differ from each other by a small amount $\delta > 0$ (monotonically increasing); (d) the arrival times of subsequent jobs differ from each other by a small time $\epsilon > 0$. Let $\bar{v}$ denote the value of the last job in a W-set (also the largest job value in this W-set by construction). Note that by setting the values of ϵ and δ, then $\bar{v}$ can be made to be arbitrarily large for C-benevolent job sequences.

Theorem 5 *No deterministic algorithm for C-benevolent job sequences on three machines can achieve a competitive ratio lower than 1.436.*

Proof We prove this lower bound by considering a sequential game between a deterministic algorithm and an adaptive adversary. We will show that there exists a strategy for the adversary to drive inverse of the competitive ratio of any deterministic algorithm to $0.696 + \zeta$, where $\zeta > 0$ can be arbitrarily small.

First, the adversary generates three identical W-sets, with sets arriving as one slightly after another by a delay of $\epsilon' \ll \epsilon$. We make this difference between these three sets only to conform to the assumption that no two jobs share the same arrival time; we will ignore this ϵ' difference in the arrival times of two jobs in the following. Now a deterministic algorithm, denoted by A, has several choices of which jobs to assign to each machine. Let $J_x(x)$, $J_y(y)$, and $J_z(z)$ denote the job (job value) that A assigns to M_1, M_2, and M_3, respectively. Let OPT denote the optimal reward and $r(A)$ denote the reward for algorithm A. Then inverse of the competitive ratio of A is $\gamma_A = r(A)/OPT$. The adversary will react adaptively to different choices made by A, as discussed case by case.

Note that if at least one machine does not have any job that has been scheduled by A, then γ_A should be no greater than $2/3 < 17/24$. Moreover, each machine can have at most one job from the three W-sets since jobs in these W-sets conflict with each other. Therefore, we assume that all three machines have one job from the three W-sets scheduled by A in the following.

Case 1 All of the scheduled jobs have a value of one. That is, $x = y = z = 1$. In this case, the adversary generates no more jobs. Therefore, $OPT = 3\bar{v}$, $r(A) = 3$, and hence, $\gamma_A = 1/\bar{v} \leq 1/2$, for $\bar{v} \geq 2$.

Case 2 Two of the scheduled jobs have a value of one. Suppose $x = y = 1$ and $z > 1$. In this case, the adversary generates no more jobs. Therefore, $OPT = 3\bar{v}$, $r(A) = 2 + z$, and hence, $\gamma_A = (2 + z)/(3\bar{v}) \leq 1/3 + 2/(3\bar{v}) \leq 2/3$, for $\bar{v} \geq 2$.

Case 3 One of the scheduled jobs has a value of one. Suppose $x = 1$. Then we have two sub-cases for the other two scheduled jobs: (a) $y = z$ and (b) the value of one job is strictly smaller than the other. For sub-case (b), without loss of generality, suppose $y < z$. If $y \leq \bar{v}/2$ and $z \leq \bar{v}/2$, then $\gamma_A \leq 1/2$. Therefore, we consider $y = z > \bar{v}/2$ for case (a) and $z > \bar{v}/2$ for case (b).

For Case 3 (a), the adversary generates three additional identical jobs that arrive right before the completion time of J_y and J_z but after the completion time of the preceding job in the W-set (Jobs that that arrive right before the completion time of some job J but after the completion time of the job preceding J in the W-set are referred to as the *challenger jobs* for J). The values of these three new jobs are all equal to y. Then, the reward for A is at most $r(A) = 1 + 3y$ (since job J_x does not conflict with the new arrivals). However, the optimal reward is $OPT = 3(2y - \delta)$. Therefore, $\gamma_A = (1 + 3y)/\,(3(2y - \delta)) \leq 2/3 + \zeta$, for δ sufficiently small and $\bar{v}$ sufficiently large.

For Case 3 (b), if $y > z/2$, then the adversary generates three additional challenger jobs with value z for J_y. Then, the reward for A is at most $r(A) = 1 + 3z$. However, the optimal reward is $OPT = 3(y + z - \delta)$. Therefore, $\gamma_A = (1 + 3z)/\,(3(y + z - \delta)) \leq 2/3 + \zeta$, for δ sufficiently small and $\bar{v}$ sufficiently large. Otherwise, if $y \leq z/2$, then the adversary generates three additional challenger jobs with value z for J_z. Then, the reward for A is at most $r(A) = 1 + y + 3z$. However, the optimal reward is $OPT = 3(z + z - \delta)$. Therefore, $\gamma_A = (1 + y + 3z)/\,(3(z + z - \delta)) \leq 7/12 + \zeta$, for δ sufficiently small and $\bar{v}$ sufficiently large.

Case 4 None of the scheduled jobs has a value of one. That is, $x, y, z > 1$. In this case, we have four sub-cases: (a) All the scheduled jobs have the same value, $x = y = z$. (b) Two of the scheduled jobs have the same value, and this value is greater than the other job value; suppose $x < y = z$. (c) Two of the scheduled jobs have the same value, and this value is smaller than the other job value; suppose $x = y < z$. (d) None of the scheduled jobs has the same value; suppose $x < y < z$. We provide an upper bound for inverse of the competitive ratio γ_A for each sub-case separately.

For Case 4 (a), the adversary generates three additional challenger jobs with value x for J_x. Then, the reward for A is at most $r(A) = 3x$. However, the optimal reward is $OPT = 3(2x - \delta)$. Therefore, $\gamma_A = 3x/(3(2x - \delta)) \leq 1/2 + \zeta$, for δ sufficiently small.

For Case 4 (b), if $x > y/2$, then the adversary generates three additional challenger jobs with value y for J_x. Then, the reward for A is at most $r(A) = 3y$. However, the optimal reward is $OPT = 3(x + y - \delta)$. Therefore, $\gamma_A = 3y/(3(x + y - \delta)) \leq 2/3 + \zeta$, for δ sufficiently small and $\bar{v}$ sufficiently large. Otherwise, if $x \leq y/2$, then the adversary generates three additional challenger jobs with value y for J_y. Then, the reward for A is at most $r(A) = x + 3y$. However, the optimal reward is $OPT = 3(2y - \delta)$. Therefore, $\gamma_A = (x + 3y)/(3(2y - \delta)) \leq 7/12 + \zeta$, for δ sufficiently small and $\bar{v}$ sufficiently large.

For Case 4 (c), if $x > z/2$, then the adversary generates three additional challenger jobs with value z for J_x. Then, the reward for A is at most $r(A) = 3z$. However, the optimal reward is $OPT = 3(x + z - \delta)$. Therefore, $\gamma_A = 3z/(3(x + z - \delta)) \leq 2/3 + \zeta$, for δ sufficiently small and $\bar{v}$ sufficiently large. Otherwise, if $x \leq z/2$, then the adversary generates three additional challenger jobs with value z for J_z. Then, the reward for A is at most $r(A) = 2x + 3z$. However, the optimal reward is $OPT = 3(2z - \delta)$. Therefore, $\gamma_A = (2x + 3z)/(3(2z - \delta)) \leq 2/3 + \zeta$, for δ sufficiently small and $\bar{v}$ sufficiently large.

For Case 4 (d), if $x > z/2$, then the adversary generates three additional challenger jobs with value z for J_x. Then, the reward for A is at most $r(A) = 3z$. However, the optimal reward is $OPT = 3(x + z - \delta)$. Therefore, $\gamma_A = 3z/(3(x + z - \delta)) \leq 2/3+\zeta$, for δ sufficiently small and $\bar{v}$ sufficiently large. Otherwise, if $x \leq z/2$ and $y > 0.676z$, then the adversary generates three additional challenger jobs with value z for J_y. Then, the reward for A is at most $r(A) = x + 3z$. However, the optimal reward is $OPT = 3(y + z - \delta)$. Therefore, $\gamma_A = (x + 3z)/(3(y + z - \delta)) \leq 0.697 + \zeta$, for δ sufficiently small and $\bar{v}$ sufficiently large. Otherwise, if $x \leq z/2$ and $y \leq 0.676z$, then the adversary generates three additional challenger jobs with value z for J_z. Then, the reward for A is at most $r(A) = x + y + 3z$. However, the optimal reward is $OPT = 3(2z - \delta)$. Therefore, $\gamma_A = (x + y + 3z)/(3(2z - \delta)) \leq 0.696 + \zeta$, for δ sufficiently small and $\bar{v}$ sufficiently large.

Summarizing all the possible cases, since ζ can be arbitrarily small, no deterministic algorithm can achieve a competitive ratio lower than $1/0.696 = 1.436$ on three machines for C-benevolent job sequences versus adaptive adversaries.

$\square$

6 Conclusion

We consider scheduling C-benevolent jobs on multiple machines. For scheduling on even number of machines, we provide a 2-competitive Pairing-m algorithm. For odd number of machines, we provide a $(2+2/\text{m})$-competitive Pairing-m algorithm. Lower bounds of 2 and 1.436 for the competitive ratio of deterministic algorithms for scheduling C-benevolent jobs on two and three machines are provided, respectively. We show that the Cooperative Greedy algorithm is 2-competitive for scheduling C-benevolent jobs on two machines, and hence, is the best obtainable deterministic online algorithm for two machines.

There are several directions for future work. One direction under investigation is to consider other classes of job sequences, such as D-benevolent job sequences [20] and equal-execution-time arbitrary-value job sequences [12]. Another research direction is to use other techniques to develope other algorithms with better competitive ratios, such as the Primal-Dual method for linear programming.

Acknowledgements This research has been supported in part by the Air Force Office of Scientific Research under Grant No. FA9550-15-1-0100. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Government, or the Air Force Office of Scientific Research.

References

1. Arkin, E.M., Silverberg, E.B.: Scheduling jobs with fixed start and end times. Discrete Appl. Math. **18**(1), 1–8 (1987)
2. Bar-Noy, A., Guha, S., Naor, J., Schieber, B.: Approximating the throughput of multiple machines in real-time scheduling. SIAM J. Comput. **31**(2), 331–352 (2001)
3. Baruah, S., Koren, G., Mao, D., Mishra, B., Raghunathan, A., Rosier, L., Shasha, D., Wang, F.: On the competitiveness of on-line real-time task scheduling. Real Time Syst. **4**(2), 125–144 (1992)
4. DasGupta, B., Palis, M.A.: Online real-time preemptive scheduling of jobs with deadlines on multiple machines. J. Sched. **4**(6), 297–312 (2001)
5. Epstein, L., Jeż, Ł., Sgall, J., van Stee, R.: Online scheduling of jobs with fixed start times on related machines. Algorithmica **74**(1), 156–176 (2016)
6. Epstein, L., Levin, A.: Improved randomized results for the interval selection problem. Theor. Comput. Sci. **411**(34–36), 3129–3135 (2010)
7. Epstein, L., Sgall, J., et al.: Approximation schemes for scheduling on uniformly related and identical parallel machines. Algorithmica **39**(1), 43–57 (2004)
8. Erlebach, T., Spieksma, F.C.: Simple algorithms for a weighted interval selection problem. In: International Symposium on Algorithms and Computation, pp. 228–240. Springer (2000)
9. Erlebach, T., Spieksma, F.C.: Interval selection: applications, algorithms, and lower bounds. J. Algorithms **46**(1), 27–53 (2003)
10. Faigle, U., Nawijn, W.M.: Note on scheduling intervals on-line. Discrete Appl. Math. **58**(1), 13–17 (1995)
11. Fung, S.P., Poon, C.K., Yung, D.K.: On-line scheduling of equal-length intervals on parallel machines. Inf. Process. Lett. **112**(10), 376–379 (2012)
12. Fung, S.P., Poon, C.K., Zheng, F.: Online interval scheduling: randomized and multiprocessor cases. J. Comb. Optim. **16**(3), 248–262 (2008)
13. Fung, S.P., Poon, C.K., Zheng, F.: Improved randomized online scheduling of intervals and jobs. Theory Comput. Syst. **55**(1), 202–228 (2014)
14. Kovalyov, M.Y., Ng, C., Cheng, T.E.: Fixed interval scheduling: models, applications, computational complexity and algorithms. Eur. J. Oper. Res. **178**(2), 331–342 (2007)
15. Krumke, S.O., Thielen, C., Westphal, S.: Interval scheduling on related machines. Comput. Oper. Res. **38**(12), 1836–1844 (2011)
16. Lawler, E.L.: A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. Ann. Oper. Res. **26**(1), 125–133 (1990)
17. Lipton, R.J., Tomkins, A.: Online interval scheduling. In: SODA, vol. 94, pp. 302–311 (1994)
18. Miyazawa, H., Erlebach, T.: An improved randomized on-line algorithm for a weighted interval selection problem. J. Sched. **7**(4), 293–311 (2004)

19. Seiden, S.S.: Randomized online interval scheduling. Oper. Res. Lett. **22**(4), 171–177 (1998)
20. Woeginger, G.J.: On-line scheduling of jobs with fixed start and end times. Theor. Comput. Sci. **130**(1), 5–16 (1994)