Theoretical Computer Science 543 (2014) 46–51

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

An approximation algorithm based on game theory for scheduling simple linear deteriorating jobs

Kenli Li[a,*], Chubo Liu[a], Keqin Li[a,b]

[a] College of Information Science and Engineering, Hunan University, National Supercomputing Center in Changsha, Changsha, 410082, China
[b] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Article history:
Received 5 August 2013
Received in revised form 2 May 2014
Accepted 28 May 2014
Communicated by D.-Z. Du

Keywords:
Approximation algorithm
Non-cooperative game theory
Price of anarchy
Simple linear deterioration

ABSTRACT

We consider the scheduling of simple linear deteriorating jobs on parallel machines from a new perspective based on game theory. In scheduling, jobs are often controlled by independent and selfish agents, in which each agent tries to select a machine for processing that optimizes its own payoff while ignoring the others. We formalize this situation as a game in which the players are job owners, the strategies are machines, and a player's utility is inversely proportional to the total completion time of the machine selected by the agent. The price of anarchy is the ratio between the worst-case equilibrium makespan and the optimal makespan. In this paper, we design a game theoretic approximation algorithm ***A*** and prove that it converges to a pure-strategy Nash equilibrium in a linear number of rounds. We also derive the upper bound on the price of anarchy of ***A*** and further show that the ratio obtained by ***A*** is tight. Finally, we analyze the time complexity of the proposed algorithm.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

It is a classical problem to schedule jobs with fixed processing times. However, there are many situations in which the processing time of a job increases with the delay of the starting time. Examples can be found in fire fighting, steel production, financial management [1,2], where delay in dealing with a job results in an increasing effort to complete the job. Such problems are generally known as scheduling with deterioration effect. The reader is referred to Kunnathur and Gupta [1], and Mosheiov [2,3] for motivations to model job deterioration in such a manner.

Scheduling of deteriorating jobs was initiated by Browne and Yechiali [4], Gupta and Gupta [5]. They defined a linear deteriorating job as a job whose actual processing time linearly increases when its starting time postpones. More precisely, the processing time of job J_i is expressed as $p_i = a_i + b_i t_i$, where a_i is the basic processing time, b_i (> 0) the deteriorating rate, and t_i the starting time. They showed that scheduling jobs in a non-decreasing order of a_i/b_i minimizes the makespan. Mosheiov [2] further introduced the concept of simple linear deteriorating job in which $p_i = b_i t_i$. Several polynomial time algorithms were proposed for the objective to minimize makespan, flow time, total completion time, total general completion time, and so on. Yu and Wong [6] proposed an algorithm DSDR (Delayed Smallest Deteriorating Rate) to minimize the total general completion time. The above research focused on a single machine for linear deteriorating jobs scheduling.

With respect to simple linear deteriorating jobs scheduling on m parallel machines, the problem becomes more complicated. Mosheiov [3] proved that the makespan minimization problem is strongly NP-hard even for a two-machine case. An

* Corresponding author.
E-mail addresses: lkl@hnu.edu.cn (K. Li), liuchubo@hnu.edu.cn (C. Liu), lik@newpaltz.edu (K. Li).

http://dx.doi.org/10.1016/j.tcs.2014.05.023
0304-3975/© 2014 Elsevier B.V. All rights reserved.

asymptotically optimal heuristic algorithm was given in that paper. Ji and Cheng studied the problem in [7] to minimize the total completion time and they further [8] showed that the problems to minimize makespan, total machine load, and total completion time are strongly NP-hard with an arbitrary number of machines and NP-hard in the ordinary sense with a fixed number of machines. They proved the non-existence of polynomial time approximation algorithm with a constant ratio when the number of machines is arbitrary for the former two problems, and then proposed two similar fully polynomial time approximation schemes (FPTAS). Miao, Zhang and Cao [9] considered the makespan minimization problem with simple linear deteriorating function (i.e., $p_i = b_i t_i$), and also proposed an FPTAS. For more results on the scheduling of linear deteriorating jobs, the reader is referred to [10–14].

In this paper, we consider the scheduling of simple linear deteriorating jobs on m parallel machines from a new perspective based on game theory. Each job is regarded as a player and we propose an algorithm $\boldsymbol{A}$. We prove that the proposed algorithm is a potential game and converges to a *Nash equilibrium* in a linear number of rounds. The price of anarchy of the algorithm is $(1+b_{\max})^{\frac{m-1}{m}}$, where $b_{\max}$ is the maximum deteriorating rate of all jobs. We further show that the price of anarchy obtained by $\boldsymbol{A}$ is tight. Finally, we analyze the time complexity of the algorithm, which is $\Theta(n\log(n))$ where n is the number of jobs to be scheduled.

The rest of the paper is organized as follows. In Section 2, we formally define the problem and give some necessary notations for the discussion. In Section 3, we introduce the concept of non-cooperative game theory and consider the scheduling problem in a game way. In Section 4, we present an approximation algorithm and analyze its properties. We conclude the paper and suggest some interesting topics for future research in the last section.

2. Problem statement and notations

There are a set of n simple linear deteriorating jobs $\ell = \{J_1, J_2, \ldots, J_n\}$, which are simultaneously available at time t_0, to be scheduled on m (>0) identical parallel machines. Once a job has been processed, it cannot be interrupted by any other job until it is finished. For each job J_i, its actual processing time is $p_i = b_i t_i$, where b_i (>0) and t_i are its deteriorating rate and starting time respectively. We assume that $t_0 > 0$, since otherwise if $t_0 = 0$, it is trivial that the makespan is equal to 0 due to $p_i = 0$ $(i \in \{1, \ldots, n\})$. The objective is to minimize the makespan, i.e., the completion time of the last finished job. We denote the problem as $P_m|p_i = b_i t_i|C_{\max}$.

Denote by M_j $(j \in \{1, \ldots, m\})$ the j-th machine in the system. Let $C_{[i]}$ be the completion time of the i-th job on a machine. It is noted that $C_{[i+1]} = t_{[i+1]} + p_{[i+1]} = t_{[i+1]}(1 + b_{[i+1]}) = C_{[i]}(1 + b_{[i+1]})$. Thus, by induction, we have

$$C_{[i]} = t_0 \prod_{j=1}^{i} (1 + b_{[j]}),$$

for every $i \geq 1$ on this machine.

Given a schedule, let n_j $(j \in \{1, \ldots, m\})$ be the number of jobs scheduled on machine M_j. Then $n_j \in \{1, \ldots, n\}$ for $j \in \{1, \ldots, m\}$ and $\sum n_j = n$. Denote C_j the completion time of the last finished job on M_j. We obtain

$$C_j = t_0 \prod_{i=1}^{n_j} (1 + b_{[i]}).$$

Our goal is to minimize the makespan, i.e., $C_{\max} = \max_{j=1,\ldots,m}(C_j)$. Since $t_0 > 0$ is an extraneously given constant, we assume without loss of generality that $t_0 = 1$.

3. Non-cooperative game theory

Game theory studies the problems in which players try to maximize their returns. In this section, we formulate the problem $P_m|p_i = b_i t_i|C_{\max}$ as a non-cooperative game among the players. As described in [15], a non-cooperative game consists of a set of players, a set of strategies, and preferences over the set of strategies. In this paper, each job in ℓ is regarded as a player, i.e., the set of players is the n non-preemptive jobs. The strategy set S_i of player i $(i \in \{1, \ldots, n\})$ is the m identical parallel machines, i.e., $S_i = \{1, \ldots, m\}$. We define $S = \bigtimes_{i=1}^{n} S_i$.

The strategy of player i is represented by s_i $(s_i \in \{1, \ldots, m\})$. Thus, for all players we obtain a strategy vector $\boldsymbol{s} = (s_1, \ldots, s_n)$ $(\boldsymbol{s} \in S)$. $\boldsymbol{s}$ is called a job scheduling strategy vector. Each player's preference is represented by its utility u_i $(i \in \{1, \ldots, n\})$ and the player tries to maximize it. We denote $u_i = u_i(\boldsymbol{s})$ which means the utility of player i when the strategy vector of all players is $\boldsymbol{s}$. A player i prefers the strategy s_i^* to the strategy s_i' if and only if $u_i(s_i^*, \boldsymbol{s}_{-i}) > u_i(s_i', \boldsymbol{s}_{-i})$, where $\boldsymbol{s}_{-i}$ is used to denote what remains from $\boldsymbol{s}$ when its i-th element s_i is dropped. $(s_i', \boldsymbol{s}_{-i})$ denotes the strategy vector after replacing s_i by s_i'.

Let $C_j(\boldsymbol{s})$ $(j \in \{1, \ldots, m\})$ be the completion time of last finished job on M_j when the strategy vector is $\boldsymbol{s}$. $C_{\max}(\boldsymbol{s})$ denotes the maximal completion time of all machines under $\boldsymbol{s}$. Similarly, let $C_{\min}(\boldsymbol{s})$ be the minimal completion time. Then the scheduling problem stated in Section 2 is to find a strategy vector $\boldsymbol{s}$ such that $C_{\max}(\boldsymbol{s})$ is minimized, i.e.,

$$\textit{minimize} \quad \max_{j=1,\ldots,m} C_j(\boldsymbol{s}), \quad \boldsymbol{s} \in S \qquad \text{(OPT)}$$

It is NP-hard to find an optimal strategy vector for the above scheduling problem. However, there exists an effective solution to a non-cooperative game. We define $u_i(\boldsymbol{s}) = 1/C_{s_i}(\boldsymbol{s})$ $(i \in \{1, \ldots, n\})$, each player i tries to find a machine M_{s_i} such that its utility is maximized.

Definition 3.1 *(Nash equilibrium).* A *Nash equilibrium* of the non-cooperative game defined above is a strategy vector $\boldsymbol{s}^*$ such that for every player i $(i \in \{1, \ldots, n\})$:

$$s_i^* \in \arg\max_{s_i \in S_i} u_i\left(s_i, \boldsymbol{s}_{-i}^*\right), \quad \boldsymbol{s}^* \in S$$

At the *Nash equilibrium*, every player cannot further increase its utility by choosing a different strategy while the strategies of other players are fixed. The equilibrium strategy vector can be found when each player's strategy is the best response to the strategies of other players. Given complete knowledge of the system, the best response for player i $(i \in \{1, \ldots, n\})$ is a solution to the following optimization problem (BR_i):

$$\mathit{minimize} \quad C_{s_i}(s_i, \boldsymbol{s}_{-i}), \quad s_i \in S_i$$

Remark 3.1. In finding the solution to BR_i, the strategies of all other players are kept fixed, so the variable in BR_i is the strategy of player i, i.e., s_i.

In this paper, we use the price of anarchy [16], which is the ratio between the worst possible *Nash equilibrium* makespan and the optimal makespan, as a measure of the effectiveness of our proposed algorithm.

4. Algorithm *A*

This section presents a polynomial time algorithm $\boldsymbol{A}$, which constructs a solution for the $P_m|p_i = b_i t_i|C_{\max}$ problem based on game theory. We first show the convergence of $\boldsymbol{A}$ and then analyze the solution obtained by $\boldsymbol{A}$. The approximation and time complexity guarantee of the algorithm are also analyzed.

Let us think of jobs as players. The game is assumed to be played sequentially with players taking turns in some order to make a decision. We emphasize that in each turn exactly one player tries to change strategy to increase its utility while the others keep their strategies fixed. Let $\boldsymbol{s}$ denote the strategy vector of all players at the beginning of the k-th iteration. Given that it is the turn of player i to make a decision in this iteration, it selects a machine M_{s_i} so as to maximize its utility, that is, player i solves the problem BR_i mentioned in Section 3.

We next present the algorithm $\boldsymbol{A}$. The main idea is to construct a strategy vector in which every player cannot further increase its utility while the strategies of other players are kept fixed. We propose to do this with a greedy scheme. The player with a larger deteriorating rate is earlier to make a decision.

Algorithm *A*.
Step 1. Re-index jobs in non-increasing order of their deteriorating rates such that $b_1 \geq b_2 \geq \ldots \geq b_n$. Randomly assign each job J_i $(i \in \{1, \ldots, n\})$ to a machine and set $k = 0$.
Step 2. Record the initial selection vector $\boldsymbol{s}^0$ and store the value $C_j(\boldsymbol{s}^0)$ $(j \in \{1, \ldots, m\})$ in a min-heap.
Step 3. In the k-th iteration, get the machine M_j with the smallest current load from the min-heap and it is the turn of player i $(i = k \bmod n + 1)$ to make a decision. Assuming that the strategy vector is $\boldsymbol{s}$ at the beginning of this iteration. If $C_{s_i}(\boldsymbol{s}) > C_j(\boldsymbol{s})(1 + b_i)$, then set $s_i = j$.
Step 4. Set $k = k + 1$ and update values in the min-heap, go to Step 3 until the strategy vector $\boldsymbol{s}$ in the k-th iteration is equal to the one in the $(k - n)$-th iteration.

4.1. Convergence to pure-strategy Nash equilibrium

It is catastrophic if the job to machine mapping keeps changing. A good scheduling mechanism is one with a small price of anarchy and fast convergence to a *Nash equilibrium*. In this section, we analyze the convergence of our proposed algorithm $\boldsymbol{A}$. The result shows that $\boldsymbol{A}$ converges to a *Nash equilibrium* in n number of rounds where n is the number of jobs.

Before addressing the problem mentioned above, we first introduce the notation of a state graph and a potential function which can be found in [14]. As described in [14], a state graph $G = (V, E)$ is a directed graph where the set V of nodes equals to S $(S = \bigtimes_{i=1}^{n} S_i)$, and an edge exists from state $\boldsymbol{s}$ to $\boldsymbol{s}'$ if the difference between these two states is the strategy of a player i $(i \in \{1, \ldots, n\})$ and i's payoff is strictly less in $\boldsymbol{s}'$. A *Nash equilibrium* corresponds to a node with no outgoing edges. A potential function is a function f mapping the set of states to a totally ordered set such that $f(\boldsymbol{s}')$ is strictly less than $f(\boldsymbol{s})$ for each edge from $\boldsymbol{s}$ to $\boldsymbol{s}'$ in E. In other words, whenever a player in state $\boldsymbol{s}$ changes its strategy to improve its payoff, the resulting state $\boldsymbol{s}'$ satisfies $f(\boldsymbol{s}) > f(\boldsymbol{s}')$.

The authors in [14] also mentioned that the existence of a potential function implies that the state graph is acyclic, which establishes the existence of a *Nash equilibrium*. A game that has a potential function is also called a potential game.

Theorem 4.1. *The proposed algorithm* $\boldsymbol{A}$ *for* $P_m|p_i = b_it_i|C_{\max}$ *is a potential game.*

Proof. For any state $\boldsymbol{s}$, let $C(\boldsymbol{s})$ be the vector of total completion times of all machines sorted in non-increasing order. We show that as a job switches from one machine to another machine to increase its utility, it decreases the corresponding vector C lexicographically.

Suppose the system is in state $\boldsymbol{s}$ and $C(\boldsymbol{s}) = (C_1, C_2, \dots, C_m)$. Assuming that job i with utility $1/C_j$ $(s_i = j)$ switches its machine under the proposed policy $\boldsymbol{A}$. Call the new state $\boldsymbol{s}'$, and let $C(\boldsymbol{s}') = (C'_1, C'_2, \dots, C'_m)$. Assuming that i's utility in $\boldsymbol{s}'$ is $1/C'_{j'}$ $(j' \in \{1, \dots, m\})$. We obtain $C'_{j'} < C_j$ (see Step 3 of algorithm $\boldsymbol{A}$). Even though the change of i's strategy causes an increase on the total completion time of $M_{j'}$, the value $C'_{j'}$ is still less than C_j. Furthermore, the action obviously decreases the total completion time of M_j while keeping the total completion times of other machines (except for M_j and $M_{j'}$) unchanged. Thus this switch decreases the corresponding vector lexicographically, i.e., $C(\boldsymbol{s}') < C(\boldsymbol{s})$, and so C is a potential function. □

Corollary 4.2. *The selfish behavior of players will converge to a* Nash equilibrium *under the proposed scheduling policy* $\boldsymbol{A}$ *for* $P_m|p_i = b_it_i|C_{\max}$.

Knowing that the selfish behavior of players converges to a *Nash equilibrium* does not indicate fast convergence to a solution. The speed of convergence to a *Nash equilibrium* is also a key point. We next prove the convergence to a *Nash equilibrium* for the proposed policy $\boldsymbol{A}$. Before addressing the problem, we show two important properties which are presented in Theorem 4.3 and Theorem 4.4.

Theorem 4.3. *Suppose the algorithm* $\boldsymbol{A}$ *is in a state* $\boldsymbol{s}$ *and the subsequent state is* $\boldsymbol{s}'$. *Then we have* $C_{\min}(\boldsymbol{s}) \le C_{\min}(\boldsymbol{s}')$.

Proof. Assuming that it is the turn of player i to switch machine. Then we know that J_i moves from machine M_{s_i} to $M_{s'_i}$ and we have $C_{s'_i}(\boldsymbol{s}) = C_{\min}(\boldsymbol{s})$. Furthermore, $C_{\min}(\boldsymbol{s}') = \min\{C_{s_i}(\boldsymbol{s})/(1+b_i), C_{s'_i}(\boldsymbol{s})(1+b_i), C_j(\boldsymbol{s})\}$ $(j \in \{1, \dots, m\}$ and $j \ne s_i, s'_i)$. From the proposed algorithm, we know that it is the condition $C_{s_i}(\boldsymbol{s}) > C_{\min}(\boldsymbol{s})(1+b_i)$ which motivates the move of job i from machine M_{s_i} to $M_{s'_i}$. Therefore, we obtain $C_{s_i}(\boldsymbol{s})/(1+b_i) > C_{\min}(\boldsymbol{s})$. Clearly, $C_j(\boldsymbol{s}) \ge C_{\min}(\boldsymbol{s})$ $(j \in \{1, \dots, m\}$ and $j \ne s_i, s'_i)$. Thus, we have $C_{\min}(\boldsymbol{s}) \le C_{\min}(\boldsymbol{s}')$. □

Theorem 4.4. *Assuming that player i* $(i \in \{1, \dots, n\})$ *has updated its strategy and the corresponding state of algorithm* $\boldsymbol{A}$ *is* $\boldsymbol{s}$. *For all* $1 \le j \le i$, *we have* $s_j \in \arg\max_{s'_j \in S_j} u_j(s'_j, \boldsymbol{s}_{-j})$.

Proof. It is equivalent to prove that $s_j \in \arg\min_{s'_j \in S_j} C_{s'_j}(s'_j, \boldsymbol{s}_{-j})$, i.e., $C_{s_j}(\boldsymbol{s}) \le C_{\min}(\boldsymbol{s})(1+b_j)$. We prove the above claim by induction on i. For $i = 1$, the statement becomes $C_{s_1}(\boldsymbol{s}) \le C_{\min}(\boldsymbol{s})(1+b_1)$. Clearly, it is true and we have our induction basis.

Assuming that the result is true for player i $(i \in \{1, \dots, n-1\})$, i.e., we have for all $1 \le j \le i$, $C_{s_j}(\boldsymbol{s}) \le C_{\min}(\boldsymbol{s})(1+b_j)$. Then we need to show that for player $i+1$, for all $1 \le j \le i+1$, $C_{s_j}(\boldsymbol{s}') \le C_{\min}(\boldsymbol{s}')(1+b_j)$ where $\boldsymbol{s}'$ is the state after player $i+1$ make a decision. We proceed as follows.

In state $\boldsymbol{s}$, if the condition $C_{s_{i+1}}(\boldsymbol{s}) \le C_{\min}(\boldsymbol{s})(1+b_{i+1})$ is satisfied, then the strategy vector keeps unchanged, i.e., $\boldsymbol{s}' = \boldsymbol{s}$. Under this situation, we have $C_{s_j}(\boldsymbol{s}') \le C_{\min}(\boldsymbol{s}')(1+b_j)$ for all $1 \le j \le i+1$. Otherwise, $C_{s_{i+1}}(\boldsymbol{s}) > C_{\min}(\boldsymbol{s})(1+b_{i+1})$ and J_{i+1} moves from machine $M_{s_{i+1}}$ to $M_{s'_{i+1}}$. This action changes the total completion times of these two machines while keeping the total completion times of all others unchanged. We have $C_{s_j}(\boldsymbol{s}') = C_{s_j}(\boldsymbol{s})$ $(1 \le j \le i+1$ and $s_j \ne s_{i+1}, s'_{i+1})$. By Theorem 4.3, we know that $C_{\min}(\boldsymbol{s}) \le C_{\min}(\boldsymbol{s}')$. Therefore, $C_{s_j}(\boldsymbol{s}') \le C_{\min}(\boldsymbol{s})(1+b_j) \le C_{\min}(\boldsymbol{s}')(1+b_j)$ $(1 \le j \le i+1$ and $s_j \ne s_{i+1}, s'_{i+1})$. The completion time of $M_{s_{i+1}}$ decreases and we have $C_{s_j}(\boldsymbol{s}') \le C_{\min}(\boldsymbol{s}')(1+b_j)$ $(1 \le j \le i+1$ and $s_j = s_{i+1})$. As for all $1 \le j \le i+1$ and $s_j = s'_{i+1}$, it is obvious that $C_{s'_{i+1}}(\boldsymbol{s}') \le C_{\min}(\boldsymbol{s}')(1+b_{i+1})$. Furthermore, $b_j \ge b_{i+1}$ and $s_j = s'_{i+1}$. We also have $C_{s_j}(\boldsymbol{s}') \le C_{\min}(\boldsymbol{s}')(1+b_j)$.

Hence, $s_j \in \arg\max_{s'_j \in S_j} u_j(s'_j, \boldsymbol{s}_j)$ $(1 \le j \le i$ and $i \in \{1, \dots, n\})$, and the result follows. □

Theorem 4.5. *The proposed algorithm* $\boldsymbol{A}$ *takes n rounds to converge to a* Nash equilibrium *for* $P_m|p_i = b_it_i|C_{\max}$.

Proof. We know that a player with a larger deteriorating rate is earlier to make a decision, i.e., the players make decisions according to the sequence $b_1 \ge b_2 \ge \dots \ge b_n$. By Theorem 4.4, we know that if player i $(i \in \{1, \dots, n\})$ has updated its strategy, the players before i keep their strategies fixed. Therefore, if player n has updated its strategy, the strategy vector keeps unchanged. Thus, algorithm $\boldsymbol{A}$ takes n rounds to converge to a *Nash equilibrium* for $P_m|p_i = b_it_i|C_{\max}$. □

4.2. Analysis of the solution

We make an analysis on the solution obtained by algorithm $\boldsymbol{A}$. We prove that there exists at least one *Nash equilibrium* that is also an optimal solution and analyze the price of anarchy of the proposed algorithm.

Theorem 4.6. *There exists at least one Nash equilibrium that is also a solution of* (OPT).

Proof. Denote $\boldsymbol{s}^*$ a solution of (OPT) and assume that $\boldsymbol{s}^0 = \boldsymbol{s}^*$. After algorithm $\boldsymbol{A}$ is executed, we obtain a solution $\boldsymbol{s}$ and have $C_{\max}(\boldsymbol{s}) \le C_{\max}(\boldsymbol{s}^*)$. Since $\boldsymbol{s}^*$ has been assumed to be an optimal strategy vector, $C_{\max}(\boldsymbol{s}^*)$ cannot be further reduced. Thus, $C_{\max}(\boldsymbol{s}) = C_{\max}(\boldsymbol{s}^*)$ and consequently, $\boldsymbol{s}$ is also an optimal strategy vector. Hence, we can conclude that there is at least one *Nash equilibrium* that is also a solution of (OPT). □

Theorem 4.7. *For $P_m|p_i = b_i t_i|C_{\max}$, the price of anarchy of the game presented in $\boldsymbol{A}$ is at most $(1+b_{\max})^{\frac{m-1}{m}}$, where $b_{\max}$ is the maximum deteriorating rate.*

Proof. Even though the proof is similar to list scheduling and the bound is the same, the result obtained by our algorithm can be better in practice, because we consider the job with minimal deteriorating rate among all jobs on the machine with makespan. Denote $C^*_{\max}$ the optimal makespan and $C^{\boldsymbol{A}}_{\max}$ the makespan obtained by $\boldsymbol{A}$ respectively. Suppose M_j is the machine with $C^{\boldsymbol{A}}_{\max}$ and J_i is the job with minimal deteriorating rate among all jobs on M_j. By Theorem 4.5, we know that the strategy vector $\boldsymbol{s}$ obtained by $\boldsymbol{A}$ is a *Nash equilibrium* which satisfies $s_i \in \arg\max_{s'_i \in S_i} u_i(s'_i, \boldsymbol{s}_{-i})$ $(i \in \{1, \dots, n\})$. Let $C^{\boldsymbol{A}}_{\min} = C_{\min}(\boldsymbol{s})$. Then

$$\frac{C^{\boldsymbol{A}}_{\max}}{1+b_i} \le C^{\boldsymbol{A}}_{\min} \le C_{j'}(\boldsymbol{s}),$$

where $j' \in \{1, \dots, m\}$ and $j' \ne j$. We obtain

$$C^{\boldsymbol{A}}_{\max} \le (1+b_i)\left(\frac{\prod_{j'=1}^{m} C_{j'}(\boldsymbol{s})}{1+b_i}\right)^{\frac{1}{m}} = (1+b_i)\left(\frac{\prod_{l=1}^{n}(1+b_l)}{1+b_i}\right)^{\frac{1}{m}} = (1+b_i)^{\frac{m-1}{m}}\left(\prod_{l=1}^{n}(1+b_l)\right)^{\frac{1}{m}}.$$

Because of $C^*_{\max} \ge (\prod_{l=1}^{n}(1+b_l))^{\frac{1}{m}}$, we have

$$C^{\boldsymbol{A}}_{\max} \le (1+b_i)^{\frac{m-1}{m}} C^*_{\max} \le (1+b_{\max})^{\frac{m-1}{m}} C^*_{\max}.$$

Furthermore, since $(1+b_i)^{\frac{m-1}{m}} \le (1+b_{\max})^{\frac{m-1}{m}}$ is satisfied for all $1 \le i \le n$, the worst NE is also bounded by $(1+b_{\max})^{\frac{m-1}{m}}$. The result follows. □

Theorem 4.8. *The upper bound $(1+b_{\max})^{\frac{m-1}{m}}$ is tight for algorithm $\boldsymbol{A}$.*

Proof. It suffices to present a specific job instance in which the ratio $(1+b_{\max})^{\frac{m-1}{m}}$ is satisfied. Given an instance $\ell = \{J_1, J_2, \dots, J_n\}$ where $n = m(m-1)+2$, $b_1 = b_2 = 2^m - 1$ and $b_3 = b_4 = \dots = b_n = 1$. Then $b_{\max} = b_1$ and $(1+b_{\max})^{\frac{m-1}{m}} = (1+2^m-1)^{\frac{m-1}{m}} = 2^{m-1}$. Assuming that a solution $\boldsymbol{s}$ is obtained after $\boldsymbol{A}$ is executed, in which J_1, J_2 are assigned to M_1 and all the remaining $n-2$ jobs are equally assigned to the other $m-1$ machines. Clearly, the solution $\boldsymbol{s}$ corresponds to a *Nash equilibrium* and the makespan $C^{\boldsymbol{A}}_{\max}$ obtained by $\boldsymbol{A}$ is 2^{2m}. In an optimal schedule, J_1 and J_3 are assigned to M_1, J_2 and J_4 are assigned to M_2, and the remaining jobs are equally assigned to the other $m-2$ machines. This yields to optimal makespan $C^*_{\max} = 2^{m+1}$. Thus, $C^{\boldsymbol{A}}_{\max}/C^*_{\max} = 2^{m-1}$. The result follows. □

4.3. Time complexity analysis

Theorem 4.9. *For $P_m|p_i = b_i t_i|C_{\max}$, the time complexity of $\boldsymbol{A}$ is $\Theta(n\log(n))$.*

Proof. By algorithm $\boldsymbol{A}$, we note that Step 1 requires the time of $\Theta(n\log(n))$ to sort the jobs and $\Theta(n)$ to assign the jobs to machines. In Step 2, it takes $\Theta(m)$ to construct a min-heap. In Step 3 and Step 4, each iteration requires update operations with time $\Theta(\log(m))$ on the min-heap. There are at most $2n$ iterations in all. Hence, the time complexity of algorithm $\boldsymbol{A}$ is $\Theta(n\log(n) + n + m + 2n\log(m)) = \Theta(n\log(n))$. The result follows. □

5. Conclusion

This work presents an approximation algorithm for $P_m|p_i = b_i t_i|C_{\max}$ with price of anarchy $(1+b_{\max})^{\frac{m-1}{m}}$, where $b_{\max}$ is the maximum deteriorating rate. We tackle the problem with a game theoretic approach, in which each job is regarded as player and tries to make a decision to optimize its own payoff while ignoring the others. We further prove that the algorithm converges to a *Nash equilibrium* in n number of rounds, where n is the number of jobs. Finally, we analyze the time complexity of the proposed algorithm, which is $\Theta(n\log(n))$. A future direction is to consider more general deterioration such as $p_i = a_i + b_i t_i$, non-linear deterioration, or other time-dependent functions. It would be interesting and more challenging to study such problems.

Acknowledgements

We are very grateful to the anonymous reviewers for their comments and suggestions which significantly improved the quality of the manuscript. This research was partially funded by the Key Program of National Natural Science Foundation of China (Grant No. 61133005), and the National Natural Science Foundation of China (Grant Nos. 61070057, 61103047, 61370095), the National Science Foundation for Distinguished Young Scholars of Hunan (12JJ1011), Key Projects in the National Science & Technology Pillar Program (2012BAH09B02).

References

[1] A.S. Kunnathur, S.K. Gupta, Minimizing the makespan with late start penalties added to processing times in a single facility scheduling problem, European J. Oper. Res. 47 (1) (1990) 56–64.
[2] G. Mosheiov, Scheduling jobs under simple linear deterioration, Comput. Oper. Res. 21 (6) (1994) 653–659.
[3] G. Mosheiov, Multi-machine scheduling with linear deterioration, INFOR 36 (4) (1998) 205–214.
[4] S. Browne, U. Yechiali, Scheduling deteriorating jobs on a single processor, Oper. Res. 38 (3) (1990) 495–498.
[5] J.N. Gupta, S.K. Gupta, Single facility scheduling with nonlinear processing times, Comput. Ind. Eng. 14 (4) (1988) 387–393.
[6] S. Yu, P.W. Wong, Online scheduling of simple linear deteriorating jobs to minimize the total general completion time, Theoret. Comput. Sci. 487 (2013) 95–102.
[7] M. Ji, T.E. Cheng, Parallel-machine scheduling with simple linear deterioration to minimize total completion time, European J. Oper. Res. 188 (2) (2008) 342–347.
[8] M. Ji, T.E. Cheng, Parallel-machine scheduling of simple linear deteriorating jobs, Theoret. Comput. Sci. 410 (38) (2009) 3761–3768.
[9] C. Miao, Y. Zhang, Z. Cao, Bounded parallel-batch scheduling on single and multi machines for deteriorating jobs, Inform. Process. Lett. 111 (16) (2011) 798–803.
[10] M. Liu, F. Zheng, S. Wang, J. Huo, Optimal algorithms for online single machine scheduling with deteriorating jobs, Theoret. Comput. Sci. 445 (2012) 75–81.
[11] S. Li, J. Yuan, Parallel-machine scheduling with deteriorating jobs and rejection, Theoret. Comput. Sci. 411 (40) (2010) 3642–3650.
[12] C. Ng, S. Li, T.E. Cheng, J. Yuan, Preemptive scheduling with simple linear deterioration on a single machine, Theoret. Comput. Sci. 411 (40) (2010) 3578–3586.
[13] X.-R. Wang, X. Huang, J.-B. Wang, Single-machine scheduling with linear decreasing deterioration to minimize earliness penalties, Appl. Math. Model. 35 (7) (2011) 3509–3515.
[14] N. Immorlica, L.E. Li, V.S. Mirrokni, A.S. Schulz, Coordination mechanisms for selfish scheduling, Theoret. Comput. Sci. 410 (17) (2009) 1589–1598.
[15] S. Penmatsa, A.T. Chronopoulos, Game-theoretic static load balancing for distributed systems, J. Parallel Distrib. Comput. 71 (4) (2011) 537–555.
[16] E. Koutsoupias, C. Papadimitriou, Worst-case equilibria, Comput. Sci. Rev. 3 (2) (2009) 65–69.