Sequential Scheduling on Identical Machines

Refael Hassin, Uri Yovel*

June 30, 2014

Abstract

We study a sequential version of the well-known KP-model: Each of n agents has a job that needs to be processed on any of m machines. Agents *sequentially* select a machine for processing their jobs. The goal of each agent is to minimize the finish time of his machine. We study the corresponding *sequential price of anarchy* for m identical machines under arbitrary and LPT orders, and suggest insights into the case of two unrelated machines. **Keywords:** sequential price of anarchy, machine scheduling, congestion games, load balancing, subgame-perfect equilibrium, makespan minimization.

1 Introduction

In this paper we study the following dynamic game. There are n agents, denoted $A_1, \ldots, A_n$, and m machines (or processors). Agent A_j has a job that takes $p_j > 0$ time units if processed by any of the machines. Agents *sequentially* select one of the machines for processing their jobs, starting with A_1 and ending with A_n. While choosing a machine, an agent knows the choices made by his predecessors. Once a machine completes processing all the jobs assigned to it, they are (instantaneously) delivered to their agents. The goal of each agent is to have his job delivered at the earliest possible time. We study the corresponding *sequential price of anarchy*, denoted **SPoA**, which is the cost-ratio of the worst subgame-perfect equilibria of such games to the solution that minimizes the overall makespan of the system. The formal definitions are given below.

The above model is a *sequential* version of the well-known KP-model introduced by Koutsoupias & Papadimitriou [9]. They consider a network consisting of m equal-capacity parallel links. There are n agents, each of whom selects a link that will send his own amount of flow; all agents select their links *simultaneously*. The delay suffered by an agent is proportional to the total flow through the link. The goal of each agent is to minimize the expected delay of his flow, ignoring the effect of his choice on the other players, and the solution concept is *Nash Equilibrium*. Our scheduling setting is completely analogous to the network setting described above, i.e., jobs stand for traffic flows and machines stand for links; indeed, a large part of the literature following [9] (some of which we describe below) uses that scheduling framework.

Despite its simplicity, this abstract model captures essential features of flow in the Internet – common resources are shared by interacting agents who are assumed to act selfishly. However, it is far from clear what the actual "game" played by the Internet users is; as phrased by Scott Shenker (cited, e.g., in [14]) "The Internet is an equilibrium; we just have to identify the game." While the KP-model has been widely studied and extended, it seems that in many cases, a more appropriate model should include some form of *sequentiality*, since agents (or their jobs) may *arrive at different times*. Consequently, a new line of algorithmic research has been initiated recently, which studies sequential versions of games whose simultaneous counterparts are well-studied. Specifically, in their paper [11], Leme et al. define the notion of *sequential price of anarchy* (**SPoA**) and analyze it in games of Machine Cost Sharing, of Unrelated Machine Scheduling, and of Consensus and Cut. As in our model, in all of their settings the agents are indexed by their "order of arrival" and they choose their actions *sequentially*, knowing only the choices made by their predecessors. In this paper we focus on the common setting of *identical* machines and provide exact bounds.

*Department of Statistics and Operations Research, School of Mathematical Sciences, Tel-Aviv University, Tel-Aviv 69978, Israel. Email: {hassin,uyovel}@post.tau.ac.il

Overview of our results. In §3, we analyze the price of anarchy for m identical machines. Specifically, we prove that **SPoA** is at most $2 - \frac{1}{m}$, and when $m = 2$ this bound is tight. We also prove that if the agents are ordered in nonincreasing order of their job's processing times (this is the well-known *LPT rule*), then this bound on **SPoA** is reduced to $\frac{4}{3} - \frac{1}{3m}$. These bounds coincide with the approximation ratios of the Greedy algorithm (i.e., each agent chooses a least loaded machine) in the classical *List Scheduling model* of Graham [8]; however, their proofs are inherently different. Essentially, it is because in our model, the agents are selfish, so they need not choose a least loaded machine (except for the last agent, who will always do so). In fact, we demonstrate that the greedy strategy may be bad for an agent. In §4, we discuss **SPoA** for two unrelated machines. We conjecture that it is bounded by 3, and provide some examples (one of which achieves this bound). In Appendix A, we show how to compute an optimal solution of the game by a dynamic programming algorithm, and by a reaching algorithm.

Finally, we strongly believe that our sequential framework can be developed for other combinatorial optimization problems; at the end of the paper we present similar sequential versions for the classical Set Cover and Bin Packing problems.

Related work. The KP-model was introduced in 1999. As mentioned, this model is analogous to ours, except that agents make *simultaneous* decisions. Subsequent work improving their bounds on the price of anarchy includes Mavronicolas & Spirakis [12] and Czumaj & Vöcking [4].

The sequential price of anarchy, **SPoA**, was recently introduced by Leme et al. [11]. They prove that for *unrelated* machines, the worst **SPoA** is bounded between $\Omega(n)$ and $O(m \cdot 2^n)$; they also study **SPoA** for other games, and in [10], they study **SPoA** of sequential auctions. In [2], Biló et al. improve the above bounds to $2^{\Omega(\sqrt{n})}$ and 2^n, respectively. In [1], Angelucci et al. **SPoA** of Isolation Games, and in [5] de Jong et al. study a sequential decision variation of their main model where each player controls a set of machines and wishes to maximize the value of jobs that can be feasibly scheduled on its machines.

Another sequential model that resembles our model is that of *crowding games*, which was introduced by Milchtaich [13]. In this model, players share a common set of actions and a payoff function which is nonincreasing in the number of players who play the same action. Milchtaich shows that the perfect information sequential game formed by letting the players act in an arbitrary order has a subgame-perfect equilibrium in pure strategies. Chakrabarty et al. [3] discuss the complexity of computing solutions in such games.

The last model that we mention is not sequential but resembles sequential models. Fiat et al. [7] consider another situation where selfish agents choose their time of transmission in a shared communication media. Transmission is successful only if there is no simultaneous transmission at this time. The main difference between this model and ours is that decisions are made simultaneously and when transmission fails the agent can repeat trying in a later time, whereas in our case decisions are made sequentially and no regret is possible.

2 Notation

Let $N \equiv \{1, \ldots, n\}$, $M \equiv \{1, \ldots, m\}$. Thus, as we introduced above, the n agents are A_j, $j \in N$. We denote the m machines (or processors) by M_i, $i \in M$. Agent A_j has a job, denoted J_j, that takes $p_j > 0$ time units on any of the machines (i.e., they are identical). We denote the list of processing times of all the agents by $p := (p_1, \ldots, p_n)$. Each agent selects one of the machines for processing his job, thus, the action set for each A_j is M. In step j of the game, A_j observes the current loads on all the machines, i.e., he knows the actions chosen by $A_1, \ldots, A_{j-1}$, and chooses a machine for processing his job. Hence, the strategy for A_j is a function $s_j : M^{j-1} \to M$. We denote $J_j \in M_i$ if $s_j = i$, i.e., A_j chooses M_i for his job, and we say that *M_i is A_j's machine.* After all agents chose their machines, i.e., the *strategy profile* $s \equiv (s_1, \ldots, s_n) \in M^n$ has been determined, a complete job schedule is obtained. For a given such schedule, we denote by S_j and C_j the start time and the completion time of J_j, respectively ($j \in N$), and by L_i the (final) load on M_i, i.e., $L_i \equiv \sum_{j: J_j \in M_i} p_j$; our schedule will always be clear from the context, so we do not write $S_j(s), C_j(s)$ etc. We denote the *average load* by $\bar{L} := \frac{1}{m} \sum_{i=1}^{m} L_i = \frac{1}{m} \sum_{j=1}^{n} p_j$. We denote by C_{max} the *makespan* of the schedule, i.e.,

$$C_{max} = \max_{j \in N} C_j = \max_{i \in M} L_i.$$

Once a machine completes processing all the jobs assigned to it, they are (instantaneously) delivered to their agents, hence the cost of A_j is the (final) load of his machine, i.e., it is L_i satisfying $J_j \in M_i$. The goal of each agent is to minimize this cost, i.e., to have his job delivered at the earliest possible time. This is an *extensive form game*, and so it always posses (pure) *subgame perfect equilibria*, which can easily be calculated by *backward induction*; see Osborne and Rubinstein [15] (or any other standard textbook on Game Theory) for a comprehensive treatment of these notions. Consequently, we refer to any schedule of the jobs which was obtained by the sequential decision process as a *subgame perfect equilibrium*, or *equilibrium* for short. We denote by SPE the set of these equilibria (corresponding to the given game in context).

We denote by C^*_{max} the makespan of an *optimal schedule*, i.e., it is the minimum possible value of the makespan of the system, ideally achieved if a central authority were to schedule the entire set of jobs.

We study the corresponding *sequential price of anarchy* of the game, denoted **SPoA**, which is the cost-ratio of the worst subgame-perfect equilibrium to the optimal makespan, that is:

Definition 2.1 (Sequential price of anarchy [11])**.**

$$\boldsymbol{SPoA} \equiv \max_{s \in SPE} \frac{C_{max}(s)}{C^*_{max}}.$$

($C_{max}(s)$ is C_{max} in the schedule corresponding to the strategy profile $s \in M^n$.)

3 SPoA for identical machines

We start with three simple examples with $n = 3$ agents and $m = 2$ machines which illustrate some interesting properties.

3.1 Motivating examples

In all three examples we use a sufficiently small $\epsilon > 0$ in one of the processing times, so no ties ever occur.

Remark 3.1. *We emphasize that the conclusions below are also valid when $\epsilon = 0$, and the bounds obtained are exact and not asymptotic; if $\epsilon = 0$ the obtained solution is just one of several possibilities, so any implementation should include a tie-breaking rule. Thus, we use ϵ in the examples for convenience, but when we refer to **SPoA**, we substitute $\epsilon = 0$, since this always yields the highest possible value.*

Example 3.1. There are $n = 3$ agents and $m = 2$ machines. The processing times are $p = (p_1, p_2, p_3) = (1, 1+\epsilon, 2)$. Without loss of generality, assume that A_1 chooses M_1 for his job. Then A_2 chooses M_2 for his job, since he realizes that in this case A_3 will choose M_1, so A_2's cost will be $L_2 = p_2 = 1+\epsilon$, which is best possible. Thus, in the resulting equilibrium, $J_1, J_3 \in M_1$ and $J_2 \in M_2$, hence the loads are $L_1 = 3, L_2 = 1+\epsilon$, so $C_{max} = 3$. However, the optimal schedule (with minimum possible makespan) is $J_1, J_2 \in M_1$ and $J_3 \in M_2$, achieving $C^*_{max} = 2 + \epsilon$. Consequently, the sequential price of anarchy is **SPoA** $= \frac{3}{2}$ (this is obtained by taking $\epsilon = 0$; see Remark 3.1).

Figure 1 depicts the game-tree associated with Example 3.1. Vectors at the leaves are the cost vectors (i.e., the loads). The solid lines show the subgame-perfect strategies, and the (unique) path from the root to the leaf corresponding to the black circle is the equilibrium solution. The bold circle corresponds to an optimal solution. The values of C_{max} and C^*_{max} are underlined in their corresponding cost vectors.

Note that in the above example, each agent acts *greedily*, i.e., chooses a least loaded machine. However, the next example shows that this strategy need not yield a minimal makespan for that agent.

Example 3.2. Consider the previous example with a small change to the processing times so that $p = (1, 1-\epsilon, 2)$. Let A_1 choose M_1. Now A_2 reasons as follows: if he chooses M_2 for his job (which is currently

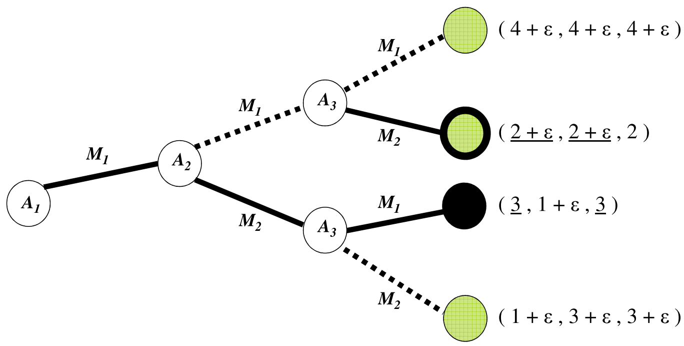

Figure 1: The game-tree associated with Example 3.1 ($n = 3$, $m = 2$)

empty), then A_3 will choose M_2 as well, hence A_2's cost will be $3 - \epsilon$. However, if he chooses M_1, A_3 will choose M_2, and so A_2's cost will be $2 - \epsilon$. Thus, in the resulting equilibrium, $J_1, J_2 \in M_1$ and $J_3 \in M_2$, hence the loads are $L_1 = 2 - \epsilon, L_2 = 2$, so $C_{max} = 2$. It is easily verified that this is also an optimal schedule, i.e., $C^*_{max} = 2$. Consequently, **SPoA** $= 1$.

Examples 3.1, 3.2 also demonstrate that an agent may prefer to have his job longer, *ceteris paribus*: the fact that p_2 is $1 + \epsilon$ (in 3.1) rather than $1 - \epsilon$ (in 3.2) gives A_2 considerable advantage.

Observe that the order of agents crucially affects the outcome:

Example 3.3. Consider again Example 3.1 but change the order of p to $p = (2, 1, 1 + \epsilon)$. It is easily verified that the resulting equilibrium is $J_1 \in M_1$, $J_2, J_3 \in M_2$, and it forms an optimal schedule as well, yielding $C^*_{max} = C_{max} = 2 + \epsilon$; consequently, **SPoA** $= 1$.

In Example 3.1, **SPoA** $= \frac{3}{2}$. We will show that this is the worst possible case for $m = 2$ (identical) machines.

3.2 Analysis

In the following, we show that for $m \geq 2$ identical machines, the sequential price of anarchy is at most $2 - \frac{1}{m}$. The argument for the case of two machines is similar to that of Graham for his classical *List Scheduling* algorithm [8] (see also chapter 2 in [16]); however, as we will show, *this argument* does not hold for $m > 2$ machines.

Lemma 3.1. *Let $m = 2$. Without loss of generality, suppose that the makespan is attained by J_j on M_1, that is, $C_{max} = L_1 = S_j + p_j$ Then, $L_2 \geq C_{max} - p_j = S_j$.*

Proof. Suppose to the contrary that $L_2 < C_{max} - p_j$. Note that by our choice of A_j, all the subsequent agents $A_{j+1}, \ldots, A_n$ choose M_2. If agent A_j changes his choice to M_2, then even if all of $A_{j+1}, \ldots, A_n$ also select M_2, still he will incur a lower cost, namely, $L_2 + p_j < C_{max} - p_j + p_j = C_{max}$. Therefore, A_j would profit by changing his selection, which is a contradiction. ∎

Theorem 3.1. *For $m = 2$ identical machines, **SPoA** $\leq \frac{3}{2}$, and this bound is tight.*

Proof. The bound follows from Lemma 3.1 in the same way as the analogous bound for the makespan of any solution obtained in the List Scheduling setting. Specifically, the optimal makespan, C^*_{max}, is bounded below both by the maximum processing time $p_{max} = \max_{k=1,\ldots,n}\{p_k\}$, and by the average load $\bar{L} = \frac{1}{2}\sum_{k=1}^{n} p_k$.

Suppose, as in Lemma 3.1, that the makespan is attained by J_j on M_1, so $C_{max} = L_1 = S_j + p_j$. By Lemma 3.1, $S_j \leq L_2$. Combined with $S_j = L_1 - p_j$, we obtain that $S_j \leq \frac{1}{2}(L_1 + L_2 - p_j) = \frac{1}{2}\sum_{k\neq j} p_k$. Therefore,

$$C_{max} = S_j + p_j \leq \frac{1}{2}\sum_{k\neq j} p_k + p_j = \bar{L} + \frac{1}{2}p_j \leq \bar{L} + \frac{1}{2}p_{max} \leq \frac{3}{2}C^*_{max}.$$

Consequently, **SPoA** $\leq \frac{3}{2}$.

As for the tightness, Example 3.1 demonstrates that **SPoA** can be $\frac{3}{2}$. ■

As we claimed in the paragraph preceding Lemma 3.1, the argument used in that lemma does not hold for $m > 2$. The precise meaning of this claim is the following: suppose that the makespan is attained by J_j, i.e., $C_{max} = S_j + p_j$. Then in our model, as opposed to List Scheduling, there may exist machines that complete processing their jobs prior to S_j. *Moreover*, J_j may start after the average load $\bar{L}$. The following example demonstrates this fact:

Example 3.4. Consider $m \geq 3$ machines and $n = m+1$ jobs, whose processing times are $(p_1, \ldots, p_{m+1}) = (K, 1-\epsilon, 1, 1, \ldots, 1, K+\epsilon)$. Let A_1 choose M_1. We claim that in the resulting schedule, A_2 will schedule his job (with $p_2 = 1-\epsilon$) on M_1 (starting at $S_1 = p_1 = K$), and each of the remaining machines $M_2, \ldots, M_m$ will be assigned a single job (which belongs to $A_3, \ldots, A_n$); see Figure 2. This is true since if A_2 chooses a machine other than M_1, say M_2, then each of $A_3, \ldots, A_{n-1}$ will send his (unit-time) job to an empty machine, and then A_n, being the last agent, will send his job (with $p_n = K+\epsilon$) to the least loaded machine, which is M_2. Thus, by choosing M_1, A_2's cost is $K+1-\epsilon$ rather than $K+1$. Note that $\bar{L} = \frac{1}{m}(2K+m-1) < K$, implying that A_2's job, which attains the makespan $C_{max} = L_1 = K+1-\epsilon$, starts after $\bar{L}$ while all other machines except the one with the last (and longest) job, complete at 1, which is earlier than $\bar{L}$.

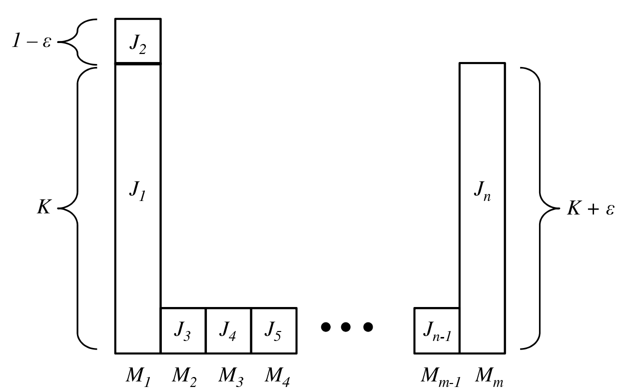

Figure 2: The schedule associated with Example 3.4

However, as we show in the sequel, the result stated in Theorem 3.1 *can* be generalized to any number of machines. We describe the key proof technique in a more general form. It claims that any upper bound on the cost of the last agent is immediately a bound on the cost of each other agent as well:

Lemma 3.2. *Let there be $m \geq 2$ identical machines. Suppose that there exists U such that for each $j \in N$, agent A_j can choose a machine so that his job's completion time is at most U, regardless of the decisions made by $A_1, \ldots, A_{j-1}$. Then, for any $j \in N$, A_j can guarantee himself a cost of at most U as well.*

Proof. The intuition behind the lemma is that once A_j chooses a machine with $C_j \leq U$, subsequent agents with large jobs will not choose his machine, because if they choose it, and hence make A_j pay a high cost, they immediately incur that same cost as well. Formally, let U be as stated in the lemma. We will show that for each $j \in N$, A_j can guarantee himself a cost of at most U, by backward induction on the agent's index. For the last agent, A_n, this is immediate since his cost is C_n, and $C_n \leq U$ by assumption.

Suppose the claim holds for all agents $A_{j+1}, \ldots, A_n$ and consider A_j. He reasons as follows. By assumption, there is a machine M_i which he can choose so that his *job*'s completion time, C_j, will not exceed U; for example, by *acting greedily*, i.e., choosing a (currently) least loaded machine, this condition must hold.

Now, if there are no subsequent agents who choose M_i, then A_j's cost will be his job's completion time, C_j, and we are done. So assume that some of $A_{j+1}, \ldots, A_n$ choose M_i as well, so A_j's cost will exceed C_j. However, by the induction hypothesis, each of them can choose a machine with (final) load at most U. They choose M_i, so their cost will be L_i, the (final) load on M_i, which satisfies $L_i \leq U$. Of course, this is A_j's cost as well. Thus, by choosing M_i, agent A_j can guarantee himself a cost of at most U. ■

Theorem 3.2. *For $m \geq 2$ identical machines, **SPoA** $\leq 2 - \frac{1}{m}$.*

Proof. We will first show that for each $j \in N$, if A_j acts greedily, then $C_j \leq \left(2 - \frac{1}{m}\right) C^*_{max}$. (for any decisions of $A_1, \ldots, A_{j-1}$). The theorem then follows by Lemma 3.2.

As in Graham's proof for List Scheduling, C^*_{max} is bounded below by $LB := \max\{p_{max}, \bar{L}\}$. We will show that if A_j chooses a least loaded machine, then $C_j \leq \Big(2 - \frac{1}{m}\Big) LB$. For each $j \in N$, this least load is bounded from above by the average load of the first $n-1$ scheduled jobs, namely $J_1, \ldots, J_{n-1}$. Hence $S_j \leq \frac{1}{m} \sum_{k=1}^{n-1} p_k$. Thus, A_j's job's completion time is bounded from above:

$$C_j = S_j + p_j \leq \frac{1}{m} \sum_{k=1}^{n-1} p_k + p_n = \bar{L} + \Big(1 - \frac{1}{m}\Big) p_n \leq \Big(2 - \frac{1}{m}\Big) LB,$$

hence $C_j \leq \Big(2 - \frac{1}{m}\Big) C^*_{max}$. This completes the proof. ■

Note that the $2 - \frac{1}{m}$ bound matches the bound on the approximation ratio of the greedy algorithm in Graham's *List Scheduling*, and the proof resembles his proof. However, the additional step of Lemma 3.2 is required because the agents are selfish, so they may apply a strategy other than the greedy one, that is, *a-priori they need not choose a least loaded machine* (except for A_n, who will always do so). This remark also applies to Theorem 3.3 below.

3.3 Imposing order on agents

Suppose that we impose order on the agents, so that $p_1 \geq p_2 \geq \cdots \geq p_n$. The justification for imposing such order is the intuition that since agents are selfish, "long jobs that arrive last" may create imbalance in the loads, and hence increase the sequential price of anarchy. To this end, we denote the setting in which this order is assumed by (the well-known) *LPT rule*, i.e., Longest Processing Time first. As we show below, it turns out that imposing such order reduces **SPoA** from $2 - 1/m$ to $4/3 - 1/3m$; note that this decrease is identical to that of the approximation ratio in Graham's List Scheduling [8].

Theorem 3.3. *For $m \geq 2$ identical machines, under the LPT rule, **SPoA** $\leq \frac{4}{3} - \frac{1}{3m}$.*

Proof. The proof idea is similar to that of Theorem 3.2; we will first show that for each $j \in N$, if A_j acts greedily, then $C_j \leq \left(\frac{4}{3} - \frac{1}{3m}\right) C^*_{max}$ (for any decisions of $A_1, \ldots, A_{j-1}$). The theorem then follows by Lemma 3.2. Thus, suppose that A_j chooses a least loaded machine.

- Case 1: $p_j \le \frac{1}{3}C^*_{max}$.
  Let $L^{(j-1)}_{min}$ denote the load on a least loaded machine immediately after A_{j-1} chose the machine for his job. Then, $L^{(j-1)}_{min} \le \frac{1}{m}\sum_{k=1}^{n-1} p_k$. Consequently, as A_j chooses a least loaded machine:

$$\begin{aligned} C_j &\le L^{(j-1)}_{min} + p_j \le \frac{1}{m}\sum_{k=1}^{n-1} p_k + p_j = \bar{L} + \left(1-\frac{1}{m}\right)p_j \\ &\le C^*_{max} + \left(1-\frac{1}{m}\right)\frac{1}{3}C^*_{max} = \left(\frac{4}{3}-\frac{1}{3m}\right)C^*_{max}. \end{aligned}$$

- Case 2: $p_j > \frac{1}{3}C^*_{max}$.
  We will prove that in this case, $C_j \le C^*_{max}$. If $j \le m$ then this is trivial (because there is an empty machine available for A_j), so assume that $j \ge m+1$. Denote by OPT any fixed optimal schedule (achieving C^*_{max}), and denote by OPT_j the corresponding sub-schedule of $J_1, \ldots, J_j$. The LPT rule implies that $p_k > \frac{1}{3}C^*_{max}$ for each $k = 1, \ldots, j$. Therefore, in OPT_j, each machine contains *at most two jobs*. Call J_k a *long* job if $p_k > C^*_{max} - p_j$ and *short* otherwise, and let r be such that $J_1, \ldots, J_r$ are long, i.e.,

$$p_1 \ge \cdots \ge p_r > C^*_{max} - p_j \ge p_{r+1} \ge \cdots \ge p_j > \frac{1}{3}C^*_{max};$$

  note that the assumption $j \ge m+1$ and the LPT rule imply that $p_j \le \frac{1}{2}C^*_{max}$, so such $r \le j-1$ exists. It follows that in OPT_j, each long job is processed on a distinct machine that does not process any other job (otherwise, there exists J_i, J_q with $i \le r, q \le j$ on the same machine, so its load is $p_i + p_q \ge p_r + p_j > C^*_{max}$). Thus, the (remaining) short jobs are processed on at most $m-r$ machines. Since each machine contains at most two jobs, we have:

$$n - r = |\{J_{r+1}, \ldots, J_j\}| \le 2(m-r). \tag{1}$$

  Now consider our schedule immediately before agent A_j chooses the machine for his job. Without loss of generality, we may assume that none of $M_1, \ldots, M_{m-r}$ contains any long job (there may exist more such machines – this happens if and only if some machine processes at least two long jobs). By (1), the number of short jobs other than J_j is

$$n - r - 1 = |\{J_{r+1}, \ldots, J_{j-1}\}| \le 2(m-r) - 1.$$

  Consequently, there exists a machine among $M_1, \ldots, M_{m-r}$ with at most one (short) job, say J_l, $r+1 \le l \le j$, so by choosing this machine, A_j's job's completion time will be $C_j = p_l + p_j \le C^*_{max}$. This completes the proof.

■

4 Two unrelated machines and concluding remarks

As mentioned in the introduction, the best bounds on the worst **SPoA** for unrelated machines are given by Biló et al. [2]. Specifically, they obtained lower and upper bounds of $2^{\Omega(\sqrt{n})}$ and 2^n, respectively. We believe that these bounds can be further improved.

Denote by p_{ij} the processing time of J_j on M_i, and we denote the processing times on M_i (of all agents) by $p_{i*} := (p_{i1}, p_{i2}, \ldots, p_{in})$, for $i \in M$.

Example 4.1. Let $n = 3, m = 2$, and the processing times are $p_{1*} = (2\epsilon, 1, 2-\epsilon)$ and $p_{2*} = (2-\epsilon, 1+\epsilon, 1+\epsilon)$. The equilibrium is $J_1 \in M_2$ and $J_2, J_3 \in M_1$; see Figure 3. Thus, $C_{max} = 3 - \epsilon$. However, the optimal solution is $J_1, J_2 \in M_1$ and $J_3 \in M_2$, $C^*_{max} = 1 + 2\epsilon$. Consequently, **SPoA** $= 3$.

We believe that Example 4.1 demonstrates the worst possible case for **SPoA** for two machines. In fact, this example corresponds to a solution of a linear program whose variables are $\{p_{ij} : i = 1,2,\ j = 1,2,3\}$ and whose objective is to maximize **SPoA**, for the *specific tree structure* depicted in Figure 3. We omit the details. We also considered alternative tree structures and the corresponding **SPoA** never exceeded 3. This motivates us to propose:

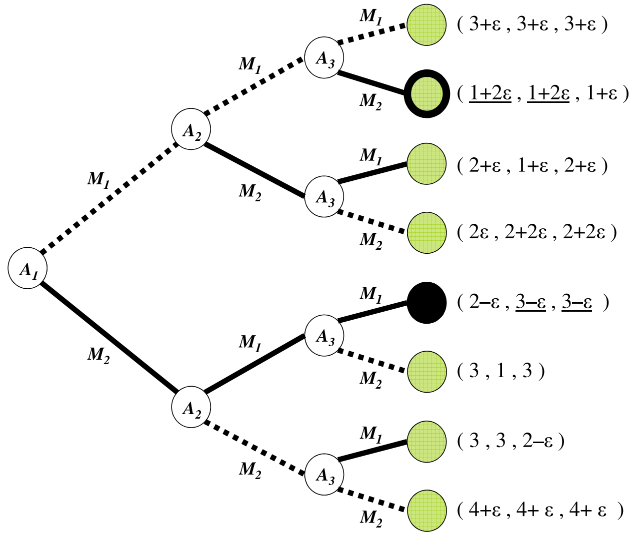

Figure 3: The game-tree associated with Example 4.1 ($n = 3$, $m = 2$)

Conjecture 4.1. *For $m = 2$ unrelated machines, **SPoA** ≤ 3.*

Sequential price of anarchy is a new concept and and the current knowledge on related models is limited. There are many interesting open research problems. For example, consider a version in which the information or the computational power of the agents is limited. Note that both "extreme cases" achieve the same sequential price of anarchy of $2 - \frac{1}{m}$: In our model, the agents actually have *full information* and *unlimited computational power*, as they may compute the complete game-tree. As we proved, **SPoA** $= 2 - \frac{1}{m}$. The other extreme is when agents act greedily, as they are completely ignorant of the remaining (future) agents' decisions; in this case, **SPoA** $= 2 - \frac{1}{m}$ by an argument similar to Graham's. However, we can consider intermediate settings, e.g., agents know the processing times of a fraction of the others, or only the minimum/maximum processing time, or they know all processing times but can only compute the optimal makespan in a fixed number of steps ahead. It is an interesting question whether **SPoA** will change in these settings. We conjecture that the answer is affirmative.

We believe that the study of the sequential price of anarchy is fruitful for other sequential games motivated from combinatorial optimization problems. We describe two suggestions:

Sequential Set Cover. The input consists of a set system, where the agents correspond to items. Agents sequentially select sets – each agent chooses a set that covers his item. The goal of each agent is to choose a set such that in the final solution, his item is covered by the least number of sets (chosen by other agents). In contrast, an optimal solution is defined as one in which the average number of *sets per item* is minimum. Note that this objective is equivalent to a minimum number of sets, which is the objective of the ordinary Set Cover problem.

Sequential Bin Packing. The input consists of items, and their volumes, each volume is at most 1. Again, agents correspond to items. Agents sequentially select bins – each agent chooses a bin in which his item can fit (this may be an existing bin or a new one). The goal of each agent is to choose a bin such that in the final solution, the total number (or volume) of items in his bin is maximum. In contrast, an optimal solution is defined as one in which the average number of *items per bin* is maximum. Note that this objective is equivalent to a minimum number of bins, which is the objective of the ordinary Bin Packing problem.

References

[1] Angelucci, A., V. Biló, M. Flammini, L. Moscardelli, "On the Sequential Price of Anarchy of Isolation Games," *COCOON* 2013, LNCS 7936, Springer, 2013.

[2] Biló, V., M. Flammini, G. Monaco, and L. Moscardelli, "Some Anomalies of Farsighted Strategic Behavior," *WAOA* 2012.

[3] Chakrabarty, D., A. Mehta, V. Nagarajan, and V. Vazirani, "Fairness and optimality in congestion games," EC 2005, 52-57.

[4] Czumaj, A. and B. Vöcking, "Tight bounds for worst-case equilibria," In Proc. 13rd ACM-SIAM Symp. on Discrete Algorithms, (2002), 413-420.

[5] De Jong, J., M. Uetz, and A. Wombacher, "Decentralized throughput scheduling," *LNCS* **7878** (2013) 134-145.

[6] Denardo, E. V., *Dynamic Programming: Models and Applications,* Prentice Hall, 1982.

[7] Fiat, A., Y. Mansour, and U. Nadav, "Efficient contention resolution protocols for selfish agents," SODA 2007, 179-188.

[8] Graham, R. L., "Bounds on multiprocessing timing anomalies," SIAM J. Appl. Math. **17** (1969), 263-269.

[9] Koutsoupias, E. and C. Papadimitriou, "Worst-case equilibria," *Proceedings of STACS'99*, LNCS **1563** (1999), 404-413.

[10] Leme, R. P., V. Syrgkanis, and É. Tardos, "Sequential auctions and externalities," *SODA* 2012, 869-886.

[11] Leme, R. P., V. Syrgkanis, and É. Tardos, "The curse of simultaneity," *ITCS* 2012, 60-67.

[12] Mavronicolas, M. and P. Spirakis, "The price of selfish routing," In Proc. 33rd ACM Symp. on Theory of Computing (2001), 510-519.

[13] Milchtaich, I., "Crowding games are sequentially solvable," *International Journal of Game Theory* **27** (1998), 501-509.

[14] Nisan, N., T. Roughgarden, É. Tardos, and V. Vazirani (Eds.), *Algorithmic Game Theory,* Cambridge University Press, 2007.

[15] Osborne, M. J. and A. Rubinstein, *A course in game theory,* MIT Press, 1994.

[16] Williamson, D. P. and D. B. Shmoys *The Design and Analysis of Approximation Algorithms,* Cambridge University Press, 2011.

A Exact Algorithms

In this appendix, we show how to compute an optimal solution, i.e., achieving C^*_{max}, for the setting of *unrelated* machines. Let p_{ij} denote the processing time of J_j on M_i, $j \in N, i \in M$.

A.1 Dynamic programming algorithm

Let the state vector be the load vector $L = (L_1, L_2, \ldots, L_m)$, meaning that upon arrival of an agent, the load already assigned to machine M_i is L_i, $i \in M$. Denote by $F_j^i(L)$ the final load of M_i when A_j arrives at state L ($i \in M, j \in N$). Note that $L_i \in \{0, \ldots, \sum_{k=1}^{j-1} p_{ik}\}$. Let $x_j(L)$ be a variable that takes value i when M_i is selected by A_j given state L.

For given L and p_{ij}, denote the vector $(L_1, \ldots, L_{i-1}, L_i + p_{ij}, L_{i+1}, \ldots, L_m)$ by $L \oplus p_{ij}$. The following is a dynamic programming algorithm:

- Define $F_{n+1}^i(L) = L_i$ for all $i \in M$, for any $L \in \mathbb{R}^m$.
- for $j = n, n-1, \ldots, 1$:
  - let $r := \operatorname{argmin}_{q \in M} F_{j+1}^q(L \oplus p_{qj})$ (ties broken arbitrarily).
  - $x_j(L) := r$.
  - for $i = 1, 2, \ldots, m$: $F_j^i(L) := F_{j+1}^i(L \oplus p_{rj})$.

Note that $C^*_{max} = \max_{i \in M} F_1^i(0, \ldots, 0)$.

The solution is constructed recursively. It can be computed in pseudopolynomial time for fixed m. It is a little simpler for identical machines. For example, consider the special case with two identical machines. In this case, $p_{1j} = p_{2j} = p_j$ for every $j \in N$. Denote by $F_j^i(h)$ the final load of M_i, $i = 1, 2$, if upon arrival of A_j, the load currently assigned to M_1 is h. Then the algorithm is:

- Define $F_{n+1}^1(h) = h$, $F_{n+1}^2(h) = \sum_{j=1}^n p_j - h$, for any $h \in \mathbb{R}$.
- for $j = n, n-1, \ldots, 1$:

  if $F_{j+1}^1(h + p_j) \le F_{j+1}^2(h)$:

  $$F_j^1(h) := F_{j+1}^1(h + p_j), \quad F_j^2(h) := F_{j+1}^2(h + p_j), \quad x_j(h) := 1.$$

  otherwise:

  $$F_j^1(h) := F_{j+1}^1(h), \quad F_j^2(h) := F_{j+1}^2(h), \quad x_j(h) := 2.$$

A.2 A reaching algorithm

An alternative *reaching algorithm* (see, e.g., the book [6]) is more efficient when n is small. First define for every vertex v of the game-tree the n-dimensional vector $L(v)$ of costs (final loads) of the agents. Let $V^{(1)} \subseteq V$ denote the set of leaves in this tree. Note that the minimum makespan is

$$C^*_{max} = \min_{v \in V^{(1)}} \max_{j \in N} L(v)_j.$$

Now recursively, in decreasing levels of the tree, compute the vector L corresponding to every node in level j (corresponding to a decision made by agent A_j). Specifically, let L' and L'' be the cost vectors of its two sons in level $j+1$. Then, $L = L'$ if $L'_j \ge L''_j$, and $L = L''$ otherwise. Note that the makespan (obtained by the sequential decision process) is $C_{max} = \max_{j \in N} L(r)_j$, where $L(r)$ is the cost vector at the root r.