Sequential Solutions in Machine Scheduling Games

Paul Giessler[1], Akaki Mamageishvili[2], Matúš Mihalák[1], and Paolo Penna[2]

Department of Computer Science, ETH Zurich, Switzerland

Abstract. We consider the classical machine scheduling, where n jobs need to be scheduled on m machines, with the goal of minimizing the makespan, i.e., the maximum load of any machine in the schedule. We study inefficiency of schedules that are obtained when jobs arrive sequentially one by one, and choose themselves the machine on which they will be scheduled. Every job is only interested to be on a machine with a small load (and does not care about the loads of other machines). Every job knows the behavior and incentives of other jobs, and knows the order in which the jobs make the decision. In game theory, the described setting forms an *extensive-form game*, and the resulting schedules are known as *subgame-perfect equilibria*.

We measure the inefficiency of a schedule as the ratio of the makespan obtained in the worst-case equilibrium schedule, and of the optimum makespan. This ratio is known as the *sequential price of anarchy (SPoA)*. We also introduce alternative inefficiency measures, which allow for a favorable choice of the order in which the jobs make their decisions. We first disprove the conjecture of Hassin and Yovel (OR Letters, 2015) claiming that for unrelated machines, i.e., for the setting where every job can have a different processing time on every machine, the sequential price of anarchy for $m = 2$ machines is at most 3. We provide an answer for the setting with $m = 2$ and show that the sequential price of anarchy grows at least linearly with the number n of players, i.e., $SPoA = \Omega(n)$. Furthermore, we show that for a certain order of the jobs, the resulting makespan is at most linearly larger than the optimum makespan. To the end, we show that if an authority can change the order of the jobs adaptively to the decisions made by the jobs so far (but cannot influence the decisions of the jobs), then there exists an adaptive ordering in which the jobs end up in an optimum schedule.

1 Introduction

Machine scheduling is a classical and fundamental optimization problem. There is an enormous literature in the scientific world about machine scheduling, see, e.g., the textbook by Pinedo [9] and the survey by Chen, Potts, and Woeginger [3] for the fundamental results and further references.

We consider the classical problem of scheduling n jobs on m *unrelated* machines. In this problem, each job has a (possibly different) processing time on each of the m machines, and a feasible schedule is simply an assignment of jobs to machines. For any such schedule, the load of a machine is the sum of all processing times of the jobs assigned to that machine. In this optimization problem, the objective is to find a schedule minimizing the *makespan*, that is, the maximum load among the machines.

This problem has been also studied from a *game theoretic* prospective where jobs correspond to players who *selfishly* aim at minimizing their own cost, and choose which

machine to place their own job accordingly. This leads to some *equilibrium* in which no player has an incentive to deviate, though the resulting schedule may not necessarily be optimal in terms of makespan. The so-called *price of anarchy* [7] quantifies the efficiancy loss due to such selfish behavior by comparing the social cost of the worst equilibrium with the optimal social cost. In particular, they do so by considering the social cost as the makespan of the resulting schedule and by restricting to the case of *identical* machines.

Such an equilibrium will have some *social cost*, that is, the makespan of the corresponding schedule.[1] The *price of anarchy*, introduced by Koutsoupias and Papadimitriou [7], is a standard way to quantify the (in)efficiency of equilibria, and it is defined as the worst case ratio of the makespan of an equilibrium schedule over the makespan of a social optimum. A slightly more optimistic measure of the efficiency of equilibrium schedules is the *price of stability* [1], defined as the best case ratio of the makespan of an equilibrium schedule over the makespan of a social schedule.

Such an equilibrium will have some *social cost*, that is, the makespan of the corresponding schedule.[2] The *price of anarchy*, introduced by Koutsoupias and Papadimitriou [7], is a standard way to quantify the (in)efficiency of equilibria, and it is defined as the worst case ratio of the makespan of an equilibrium schedule over the makespan of a social optimum. A slightly more optimistic measure of the efficiency of equilibrium schedules is the *price of stability* [1], defined as the best case ratio of the makespan of an equilibrium schedule over the makespan of a social schedule.

Much of the previous research has considered the algorithmic problem of finding a schedule of the minimum makespan. Only recently, this scheduling setting was considered from a game theoretic perspective. There are two main approaches: in the first approach, machines are viewed as players, while in the second approach, the jobs are viewed as players. In this paper we consider the latter case. Every job is owned by a player. Every player decides the machine on which the job will be processed. Every player aims to minimize her cost, which is equal to the load of the machine to which it assigns her job.

In game theory, the result of the players' decisions is an equilibrium schedule, i.e., a schedule in which no player regrets her decision (choice of a machine). Since every player is only interested in the load of the machine containing her job, an equilibrium schedule may not minimize the makespan, which is also called the *social cost* in game theory. It is thus an interesting question to quantify the efficiency of equilibrium schedules, compared to a *social optimum*, which is a schedule minimizing the social cost (makespan).

Price of anarchy, introduced by Koutsoupias and Papadimitriou [7], is a standard way to quantify the (in)efficiency of equilibrium schedules, and is defined as the worst case ratio of the makespan of an equilibrium schedule over the makespan of a social

[1] When each player chooses deterministically one machine, this definition is obvious. When equilibria are *mixed* or randomized, each player chooses one machine according to some probability distribution, and the social cost is the expected makespan of the resulting schedule.

[2] When each player chooses deterministically one machine, this definition is obvious. When equilibria are *mixed* or randomized, each player chooses one machine according to some probability distribution, and the social cost is the expected makespan of the resulting schedule.

schedule. A slightly more optimistic measure of the efficiency of equilibrium schedules is the *price of stability* [1], defined as the best case ratio of the makespan of an equilibrium schedule over the makespan of a social schedule.

Typically, in the literature, simultaneous games are considered, i.e., settings in which all players take their decisions at the same time in the beginning of the game. This approach is debatable, because in some real situations, players do not make their decisions necessarily at the same time. Even more, in some settings, players arrive one by one, and thus make their decisions sequentially.

In this paper, we study exactly the setting when players arrive one by one, and make their decision at the moment they arrive, knowing the order of the players in which they arrived and will arrive, and knowing the decisions of the previous players. All players have the same goal (having the job on a machine with small load), and players are aware of others' goal. In such a setting, the rational agents end up in an equilibrium schedule known as the *subgame perfect Nash equilibrium*.

The efficiency of solutions obtained in this sequential setting was first studied in [8]. Initial results showed that naturally defined *sequential price of anarchy* is at most $m \cdot 2^n$, while it is at least n. Later, the upper bound has been slightly improved to 2^n, and the lower bound considerably to $2^{O(\sqrt{n})}$ [2]. Hassin and Yovel [6] studied the case of identical machines, and by establishing a connection to the schedules obtained by the greedy algorithm of Graham [5] proved an upper bound of $2 - \frac{1}{m}$ for the sequential price of anarchy, while for a restricted version of a sequential price of anarchy, where agents are sorted by decreasing their processing times, Hassin and Yovel [6] proved an upper bound of $\frac{4}{3} - \frac{1}{3m}$.

The sequential price of anarchy has been studied in few other settings as well. See [8] and [4] for other examples.

1.1 Our contributions

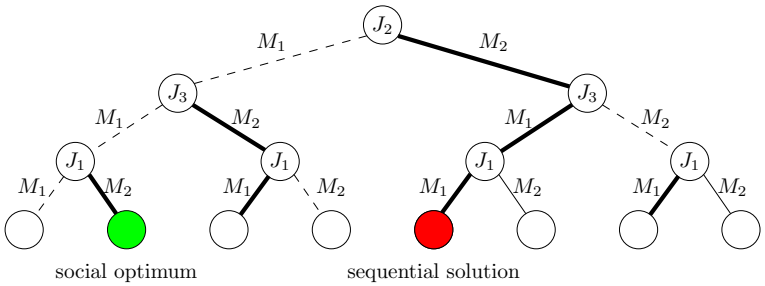
Our work is motivated by the following conjecture made by Hassin and Yovel [6]:

*The **SPoA** for $m = 2$ unrelated machines and any number of players is* 3.

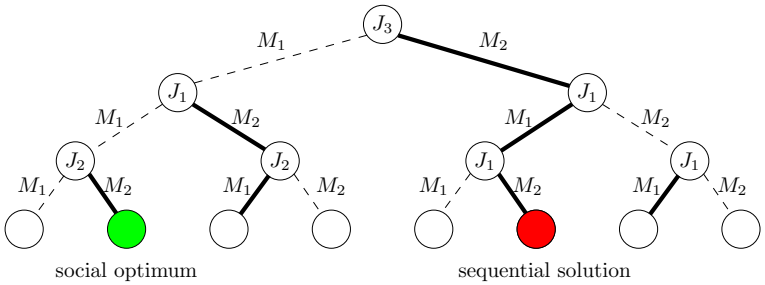
We disprove this conjecture by showing a lower bound where the sequential price of anarchy grows *linearly in the number of players* n. The result is also interesting in the sense that there is no upper bound on the sequential price of anarchy as a function of the number of machines m. On the other hand, if the initial order can be fixed by some authority, efficient schedules can be achieved. In particular, we show that a large class of sequences result in a schedule which is at most linear factor worse than the optimum. Moreover, if we assume that there is an authority that has a full control of the order in which the players make their decisions, in particular, if it does not only fix the initial sequence but can change the sequence adaptively depending on the choices of previous players, then the players can achieve an optimum schedule for the case $m = 2$. We complement this result with a negative example which shows that even for $m = 3$ machines obtaining optimum solution is impossible even with an adaptive order of players.

2 Preliminaries

In unrelated machine scheduling there are n jobs $N = (J_1, J_2, \cdots, J_n)$ and m machines $M = (M_1, M_2, \cdots, M_m)$. For each pair of machine and job (M_i, J_j) it is given how much time does machine M_i need to process job J_j, we denote this number by p_{ij}. All jobs have to be processed by assigning them to machines. Therefore, in a given schedule, for each job j we have an index of a machine it is assigned to, we denote this by s_j. When a machine M_i gets several jobs to process, it takes time which is equal to the sum $\Sigma_{s_j=i} p_{ij}$, which is also called a load of machine i and is denoted by l_i. We view the jobs as players. Players try to minimize their own costs. A cost of a player J_j is the load of a machine it is assigned to. More formally $cost_j(s) = l_{s_j}$, where s is an assignment of jobs to machines $s = (s_1, \cdots, s_n)$, for each job it is known the machine it is assigned to. s is also called a state or a strategy profile, while each component s_i is called a strategy of player i.



(a) A decision tree with fixed sequence.



(b) A decision tree with non fixed sequence.

Fig. 1: Decision trees with fixed and non fixed sequences and the corresponding sequential solutions. Dashed edges represent players' decision which are not an equilibrium, while the unique equilibrium (sequential solution) is given by the path of solid edges from the root to a leaf.

We assume that all players are rational and they know that all the other players are rational. In other words, it is a complete information game. If players enter the game sequentially, they can calculate their optimal moves by backwards induction.

More formally, it is clear that the last player makes her move greedily, the player before the last makes the move also greedily depending on the moves of the last, and so on. Any game of this type can be modelled as a decision tree, where vertices correspond to players in certain positions and edges are strategies. In the left of Figure 1 there is a decision tree for 2 machines and 3 players entering the game in a fixed order (J_2, J_3, J_1). Each leaf corresponds to a state. For given set of processing times $P = \{p_{ij}, 1 \leq i \leq m, 1 \leq j \leq n\}$ players can calculate loads on the machines in the leaf nodes (states) and then solve their own decisions by dynamic programming from the leaves to the root. It is assumed that in case of ties, all players know tie-breaking rules of all the other players. Decisions obtained this way constitute what is called the subgame perfect equilibrium, for each subtree we know what is the outcome achieved by the players in this subtree playing rationally with the given initial loads. Therefore, P and the sequence σ together with the tie-breaking rules define the game completely. Strategies (edges) that are chosen by players in the subgame perfect equilibrium are bold, and the other strategies are dashed.

Figure 1b shows an example of a non-fixed sequence game and its corresponding decision tree. Note that each path starting from the root and ending in a leaf contains all the players exactly once. The green leaf denotes the state which minimizes the makespan, while the red leaf denotes the final solution of a sequential game, which we call the sequential equilibrium. The path which connects the root node to this solution state is called subgame perfect equilibrium path.

Note that if we assume σ permutation is fixed, then P alone defines the game. We denote the makespan of the sequential equilibrium for P by $SPE(P)$ and the optimum solution makespan by $OPT(P)$. The sequential price of anarchy introduced in [8] is the worst case ratio of the sequential solution makespan over the social optimum. More formally, the sequential price of anarchy $\textbf{SPoA} = \sup_{\mathrm{P}} \frac{SPE(P)}{OPT(P)}$.

In this paper we introduce two more definitions. The first one is very intuitive, and serves as a counterpart for **SPoA**, for a given P, consider all permutations σ and choose the one which gives the best solution, i.e. the sequential equilibrium with the lowest makespan. We call the ratio of the best solution over the social optimum the sequential price of stability of game P, more formally $\textbf{SPoS}(P) = \min_{\sigma} \frac{SPE(P,\sigma)}{OPT(P)}$. The global sequential price of stability is defined as $\textbf{SPoS} = \sup_P \textbf{SPoS}(P)$.

The next definition quantifies a very optimistic scenario, where we can assume that there exists some authority which can schedule jobs in the order that gives the best outcome. For a given P, we denote by $T(P)$ the set of all possible depth n, complete $m-$ary decision trees where each path contains all the jobs (players) exactly once. For a fixed tree t, (see Figure 1b) players are still able to calculate the subgame perfect equilibrium and its makespan, denoted by $SPE(t, P)$. Note that the subgame perfect equilibrium structure may change depending on the tree nodes, but in the leaves we still have all the possible states. The adaptive sequential price of stability for a given P is defined as $\textbf{adaptive SPoS}(P) = \min_{t \in T(P)} \frac{SPE(t,P)}{OPT(P)}$, while the global adaptive sequential price of stability is equal to $\textbf{adaptive SPoS} = \sup_P \{\textbf{adaptive SPoS}(P)\}$. In this model the authority has an option to punish players for undesirable deviations, and therefore players achieve much better outcomes than in all the previous cases. We study this concept in the section Section 5.

We denote the nodes of t by H and the nodes in the level i by H_i. Clearly in the case of adaptive sequences, H_i may contain different players, while in the case of fixed sequencies, in H_i all the nodes correspond to exactly one player.

3 Linear lower bound for SPoA

In this section we consider the sequential price of anarchy for $m = 2$ unrelated machines. In [6] the authors proved a lower bound **SPoA** ≥ 3 for this case, and they conjectured that **SPoA** $= 3$. Here we disprove this conjecture by showing that **SPoA** $= \Omega(n)$, and that the conjecture does not hold already for $n = 5$ players.

3.1 A lower bound for $n = 5$ players

Theorem 1. *For two machines and at least five jobs, the **SPoA** is at least* 4.

Proof. Consider the following instance whose subgame perfect equilibrium is shown as gray boxes:

	J_1	J_2	J_3	J_4	J_5
M_1	$3-11\varepsilon$	ε	ε	$1-2\varepsilon$	$2-8\varepsilon$
M_2	ε	$2-9\varepsilon$	$2-8\varepsilon$	$1-2\varepsilon$	$1-2\varepsilon$

(1)

Note that the optimum has cost 1, while the subgame perfect equilibrium costs $4-13\varepsilon$. By letting ε tend to 0 we get the desired result.

To see why this is indeed a subgame perfect equilibrium, we assume without loss of generality that *players break ties in favor of machine* M_1. Then we note the following:

1. If the first three jobs follow the optimum, then J_4 prefers to deviate to M_2, which causes J_5 to switch to machine M_1:

	J_1	J_2	J_3
M_1	$3-11\varepsilon$	ε	ε
M_2	ε	$2-9\varepsilon$	$2-8\varepsilon$

$\Rightarrow$

J_4	J_5
$1-2\varepsilon$	$2-8\varepsilon$
$1-2\varepsilon$	$1-2\varepsilon$

   Now the cost for J_3 would be $2-6\varepsilon$.
2. Given the previous situation, J_3 prefers to deviate to M_2 because in this way J_4 and J_5 choose M_1, and her cost will be $2-7\varepsilon$:

	J_1	J_2
M_1	$3-11\varepsilon$	ε
M_2	ε	$2-9\varepsilon$

$\Rightarrow$

J_3	J_4	J_5
ε	$1-2\varepsilon$	$2-8\varepsilon$
$2-8\varepsilon$	$1-2\varepsilon$	$1-2\varepsilon$

   Now the cost for J_2 would be $3-9\varepsilon$.

3. Given the previous situation, J_2 prefers to deviate to M_2 because in this way J_3 and J_4 choose M_1, J_5 chooses M_2, and the cost for J_2 is $3 - 10\varepsilon$:

	J_1
M_1	$3-11\varepsilon$
M_2	ε

$\Rightarrow$

J_2	J_3	J_4	J_5
ε	ε	$1-2\varepsilon$	$2-8\varepsilon$
$2-9\varepsilon$	$2-8\varepsilon$	$1-2\varepsilon$	$1-2\varepsilon$

   Now the cost for J_1 would be $3 - 10\varepsilon$.

We have thus shown that, if J_1 chooses M_2 then her cost will be $3 - 10\varepsilon$. To conclude the proof, observe that if J_1 chooses M_1, then by backwards induction all players will choose the allocation in (1), and the cost for J_1 is this case is also $3-10\varepsilon$. Since players break ties in favor of M_1, we conclude that the subgame perfect equilibrium is the one in (1). □

Remark 1. In the construction above, we have used the fact that players break ties in favor of M_1. These assumptions can be removed by using slightly more involved coefficients for the ε terms, so that ties never occur.

How the instance has been obtained. It is worth mentioning that in [6] the authors discussed a linear program for finding lower bounds on the **SPoA**. In this approach the variables are the processing times $\{p_{ij}, 1 \leq i \leq m, 1 \leq j \leq n\}$, and the approach essentially goes as follows:

- Fix the subgame perfect equilibrium structure, that is, the sequence of players and all the decisions in the internal nodes, this also gives the sequential equilibrium;
- Fix the leaf which is the optimum, and impose that the optimum makespan is at most 1;

For every fixed subgame perfect equilibrium tree structure, we have one constraint for each internal node (decision of a player). The optimum state (leaf) should also be fixed and both numbers have to be assumed to be at most 1 by adding two additional contstraints to the linear program. By maximizing the maximum value of loads on the machines in the leaf which corresponds to the sequential equilibrium, we get the worst case example for this particular tree structure. There are $2^n - 1$ internal nodes in the decision tree with n players. Therefore, this approach requires exploring 2^{2^n-1} many possible subgame perfect equilibria tree structures, and for each of them we have to decide where is the optimum among 2^n leaves and solve a linear program of size $2^n \times O(n)$.

We managed to solve the case $n = 5$ players with the aid of a computer program completely by learning the tree structure of the subgame perfect equilibrium closely and breaking the symmetries. It is clear that the extremely fast growing number of possible tree structures makes the program very time-consuming even for small values. Consequently, we tried to exclude combinations from the computation, i.e. we avoid to start the linear programming solver for certain tree structures. There are some trivial cases that we present for the intuition. The first is that not all leaves of the tree should be tested for the position of the optimum. Both left and rightmost leaf nodes can be excluded from the possible optimum position due to the property that SPoA is 1 in

both cases. For the same reason, all tree structures where the equilibrium is located at those extreme leaves can also be ignored. Additionally, the leaf where the equilibrium is achieved should be avoided for the optimum position. The next idea is that trees that are mirror images of other trees with regard to the vertical axis will lead to an equal SPoA.

A crucial achievement for the speedup of the program is the discovery of the property that we describe later. It allows to exclude a high number of combinations for the last layer of the tree. As the last layer of the tree represents more than half of the digits in the tree structure vector, the number of combinations that has to be generated can be reduced drastically. For example, if $n = 5$, the improvement is from $2 \cdot 10^9$ to $6 \cdot 10^6$.

During the experimental investigations of possible outcomes based on the structure of the game tree, we found out that a relatively big part of game tree structures always leads to an infeasible linear program, regardless of the position of the optimum. Consider the very simple tree structure depicted in Figure 2. Solid line represent the best responses in each node. It is obvious that if the best response of player 2 in the left node is to choose machine 1 then the best response in the right node cannot be to choose the machine 2. We generalized this observasion to arbitrary values of players n and machines m. The observation is limited to the nodes in the second lowest level of the tree, the best responses of the last player J_n.

Let $p(h)$ denote the parent node of node h in tree t. We define the child index $c(h) = i \in M$ of node h if h is the i-th child of it parent $p(h)$, this mean that the edge from $p(h)$ to h represents that the agent corresponding to $p(h)$ selects M_i. Every node h on the j-th layer of t is defined by the choices of the agents playing before agent J_j. These decisions create a unique path from the root of t to h. The nodes on the path from the root r to h are $P(h) = \{r, ..., p(p(h)), p(h), h\} = \{h' \in H | h \in t_{h'}\}$ where $t_{h'}$ is the subtree of t rooted at h'. These definitions allow us to define the set of agents that selected a certain machine i before node h: $B(i, h) = \{j \in N | p(h') \in H_j \wedge h' \in P(h) \wedge c(h') = i\}$. Then the observation is the following:

Observation 1 *For every pair of nodes in the second lowest layer $h, h' \in H_n$ if the best response of player J_n in h is i, then it implies that the best response in h' is also i if $\forall i' \in M \setminus i : B(i', h) \subseteq B(i', h')$ holds.*

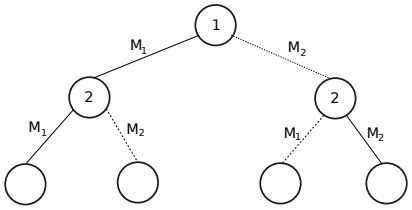


Fig. 2: Conflicting configuration

3.2 Linear lower bound

Extending the construction for $n = 5$ players is non trivial as this seems to require rather involved constants that multiply the ε terms. However, we notice that these terms only help to induce more involved tie breaking rules of the following form:

Definition 1 (arbitrary tie breaking rule). *We say that the tie breaking rule is arbitrary if each player uses a tie breaking rule between machines which possibly depends on the strategies played by the previous players in the tree.*

The following theorem gives a general result:

Theorem 2. *Even for two machines, the **SPoA** is at least linear in the number n of jobs, in the case of arbitrary tie breaking rule.*

Proof. We consider the following instance with $n = 3k - 1$ jobs arriving in this order

	J_1	J_2	J_3	J_4	J_5	J_6		J_{3k-5}	J_{3k-4}	J_{3k-3}	J_{3k-2}	J_{3k-1}
M_1	$k+1$	0	0	k	0	0	$\cdots$	3	0	0	1	2
M_2	0	k	k	0	$k-1$	$k-1$		0	2	2	1	1

and prove that the subgame perfect equilibrium is the gray allocation which results in a makespan $k + 2$, while the optimal makespan is 1. This requires players to use the following tie breaking rules in the first part: if player J_1 chooses machine M_1, then J_2 and J_3 prefer to avoid player J_1, that is, they choose the other machine in case of ties in their final cost.

We prove this by induction on k. The base case is $k = 2$ and it follows directly from the example in Equation (1) with $\varepsilon = 0$. As for the inductive step, the proof consists of the following two steps (claims below).

Claim. If the first three jobs choose their zero-cost machines, then all subsequent jobs implement the subgame perfect equilibrium on the same instance with $k' = k - 1$. The cost of J_1 in this case is $k' + 2 = k + 1$.

Proof (Proof of Claim). Note that the sequence starting from J_4 is the same sequence for $k' = k - 1$. Since the first three jobs did not put any cost on the machines, we can apply the inductive hypothesis and assume that all subsequent players play the subgame perfect equilibrium. The resulting cost on machine M_2 will be $k' + 2 = k + 1$, and this is the machine chosen by J_1.

Claim. If the first job J_1 chooses M_2, then both J_2 and J_3 choose M_1.

Proof (Proof of Claim). Choosing M_2 costs J_2 and J_3 at least k, no matter what the subsequent players do. If they instead choose M_1, by the previous claim, their cost is $k' + 1 = k$ which they both prefer given their tie breaking rule.

The above two claims show that if J_1 chooses machine M_2 then she is paying a cost $k + 1$, while in the following we show that if she chooses machine M_1, she will pay a

cost $k+1$ in this scenario as well. The tie-breaking rule is that player J_1 prefers the cost $k+1$ on the first machine M_1.

In the second part of the proof we assume different tie-breaking rules for the last two players $J_{3\cdot k-2}$ and $J_{3\cdot k-1}$, depending on which of the two players J_2 and J_3 choose machine M_2.

Now we consider the case where job J_1 chooses machine M_1, and show that she pays exactly $k+1$. If player J_2 chooses machine M_1, then we assume that player J_3 breaks ties in favor of M_2: choosing M_2 results in a cost of $k+1$ in the end, instead of some cost on machine M_1, which is at least $k+1$. If job J_3 gets assigned to machine M_2 then by backwards induction we can show that no player among $J_4, \cdots, J_{3\cdot k-3}$ gets assigned to machine where they have non-zero cost. This way only the last two players are left and we can assume that player $J_{3\cdot k-2}$ prefers to get assigned to machine M_1 and pay $k+2$ instead of getting assigned to machine M_2 and pay $k+2$ there, while the last player $J_{3\cdot k-1}$ prefers to get assigned to machine M_2 and pay the cost $k+1$ there. Therefore, job J_2 gets cost $k+2$, but here we can assume that she prefers cost $k+2$ that she gets on machine M_2. Therefore, job J_2 gets assigned to machine M_2 in the subgame perfect equilibrium state. Then again, by the same backwards induction we conclude that all players $J_3, \cdots, J_{3\cdot k-3}$ get assigned to the machine where they have cost 0, and the last two jobs $J_{3\cdot k-2}$ and $J_{3\cdot k-1}$ choose machine M_2, here again we assume that player $J_{3\cdot k-2}$ prefers to pay $k+2$ on machine M_2 than to pay the same cost on machine M_1. This way the load on machine M_1 is $k+1$, while the load on machine M_2 is $k+2$, which finishes the proof of the theorem.

Remark 2. It is worth noting that in [4] authors consider a different game (routing) and construct an unbounded lower bound example for the sequential price of anarchy, but their analysis heavily uses a carefully chosen tie-breaking rule. We solved linear programs with strict inequalities obtained from the subgame perfect equilibria tree structure given in the example from the above proof. There are solutions for $n=8$ and $n=11$, that is linear programs are feasible. Therefore, at least for small n's we can drop the assumption about tie-breaking rules completely, but the solution involves complicated coefficients for ε's and for the sake of exposition we do not present the exact solutions here.

Remark 3. We could not find any example that would give an (even locally) better lower bound on sequential price of anarchy than it is in the proof of Theorem 2. For $n \leq 7$, our computer program searched the whole space and the results obtained above are the best. We believe that the construction from the proof of the theorem gives the best lower bound example.

4 Linear upper bound on the SPoS

In this section we give a *linear upper bound* on the sequential price of stability for two machines (Theorem 3 below). Unlike in the case of the sequential price of anarchy, here we have the freedom to choose the order of the players (and suggest a particular tie breaking rule). Though finding the best order can be difficult, we found that a large set of permutations already gives a linear upper bound on **SPoS**. In particular, it is enough

that the "authority" divides the players into *two groups* and put players in the first group first, followed by the players from the second group. Inside each group players can form *any order*. The main result of this section is the following theorem:

Theorem 3. *For two machines, the **SPoS** is at most* $\frac{n}{2} + 1$.

Proof. Consider an optimal assignment and denote the corresponding makespan by OPT. By renaming jobs and machines, we can assume without loss of generality that in this optimal assignment machine M_1 gets jobs the first $k \leq \frac{n}{2}$ jobs, and machine M_2 gets all the other jobs:

$$\{J_1, J_2, \ldots, J_k\} \to M_1 \ , \qquad \{J_{k+1}, \ldots, J_n\} \to M_2 \ .$$

Take the sequence given by the jobs allocated to M_1 followed by the jobs allocated to M_2,

$$J_1, J_2, \ldots, J_k, J_{k+1}, \ldots, J_n.$$

We prove that for this sequence there is a subgame perfect equilibrium whose makespan is at most $(k+1) \cdot OPT$.

In the proof we consider the *first player who deviates*. We distinguish two cases.

Claim. If the first player J_d who deviates is in $\{J_{k+1}, \ldots, J_n\}$, then she does not improve.

Proof (of Claim). Observe that all players in $\{J_1, J_2, \ldots, J_k\}$, that come before player J_d, did not deviate. Machine M_1 has all jobs that it gets in the optimum. If J_d stays in M_2, her cost will be at most OPT. Moving to M_1 will in the end produce a feasible allocation with fewer jobs on M_2 and more jobs on M_1, compared to the optimum. The cost on M_1 is therefore at least OPT, which is the cost of J_d when deviating.

The remaining case is the following one.

Claim. If the first player J_d who deviates is in $\{J_1, \ldots, J_k\}$, then any subgame perfect equilibrium has the makespan at most $(t+1) \cdot OPT$ where $d = k + 1 - t$.

Proof (of Claim). The proof is by induction on t. For $t = 1$, the deviating player is the last in $\{J_1, \ldots, J_k\}$. If the first deviating player is J_k, then the reason she deviated is because in the final solution she is paying at most OPT. We argue that in that case M_1 will have load at most $2 \cdot OPT$. This is because if no player among $J_{k+1}, \ldots, J_n$ deviates to machine M_1, then the final load on this machine will be at most OPT. If any player deviated to M_1 this is because she is paying at most $2 \cdot OPT$ in the final solution, otherwise she would stay on the second machine and pay at most $2 \cdot OPT$.

If the first player who deviates from the optimal assignment is J_{k-t}, then we argue similarly that the makespan is at most $(t+1) \cdot OPT$. By induction we can assume that if player J_{k-t} stays on the first machine then she is guaranteed to pay at most $t \cdot OPT$, while if she deviates then we know she is paying at most $t \cdot OPT$ on the second machine. In the latter case, if any player $J_{k+1}, \ldots, J_n$ deviates from the second machine to the first machine then she is paying at most $(t+1) \cdot OPT$, because otherwise she would stay on the second machine and pay at most $(t+1) \cdot OPT$. Therefore, by induction we get that this sequence results into a solution which has a makespan at most $(k+1) \cdot OPT$, which completes the proof.

The two claims above imply the theorem □

Remark 4. This result cannot be extended to more than 2 machines, because the third machine would change the logic of the proof. We can no longer assume that players on the second machine in the optimum assignment can guarantee low costs for them just by staying on the second machine.

5 Achieving the optimum: the adaptive SPoS

In this section we study the adaptive sequential price of stability. Unlike the previous models, here we assume that there is some authority who is very strong, meaning that it has a full power on the order of players. It cannot only fix the complete order, but can also change the order depending what decision previous players get. On the other hand players still have a complete freedom into choosing any strategy and therefore, the best outcome for them. We also assume that all players know the whole structure of a decision tree defined by the authority. This model is the closest instantiation of a general extensive form game compared to the previous models in this paper. This way the authority has an option to punish players for deviating from the optimum path by placing different players after the deviating player in case of different decisions. Therefore, rational players may achieve much better solutions in the end. The following theorem shows that achieving the optimum solution is possible for 2 machines:

Theorem 4. *For two machines, the **adaptive SPoS** is* 1.

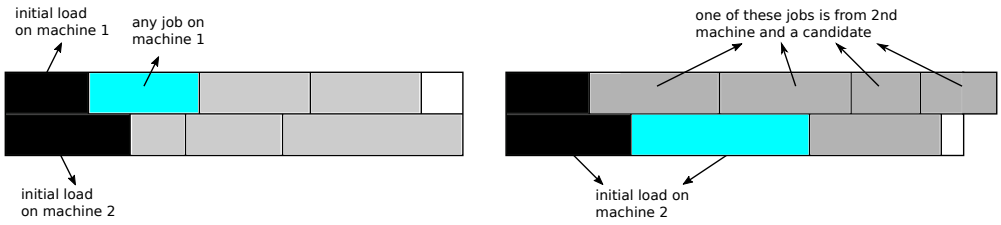


Fig. 3: Initial constrained optimum on left, new constrained optimum on right

Proof. The main idea of the proof is backwards induction. In each subtree of a decision tree we aim to implement constrained optimum. Under constrained optimum in each subtree we mean the optimum makespan solution that is possible to achieve by the remaining players in the subtree given the initial loads on both machines created by the players arriving before this point. The root of each subtree is chosen in the following way: choose the player who gets worse outcome in the case of a deviation from the optimum strategy. More formally, for each subtree consider the constrained optimum solution, we prove that there is at least one player so that if this player is assigned to

a different machine from that it is assigned in the constrained optimum, then in a new constrained optimum she gets a larger cost and therefore this deviation is undesirable for the player. Using backwards induction we can assume that in each subtree new constrained optima are implemented. Therefore, if we show the claim then it finishes the whole proof of the theorem. The proof of the claim is the following: without loss of generality consider any player from machine M_1 and put it on the machine M_2, we know that in the new constrained optimum, the makespan is at least the same as the original makespan by definition, which means that either this player or some other player from the machine M_2 in the original constrained optimum assignment gets worse outcome. We choose the player who gets a worse (or the same) outcome and this finishes the proof of the claim. See Figure 3 for the illustration.

Unfortunately, this result cannot be extended to more than 2 machines. We show this in the following theorem:

Theorem 5. *For three or more machines, the **adaptive SPoS** is at least* $\frac{3}{2}$.

Proof. Consider the following instance with three machines and three jobs, where the optimum is shown as gray boxes:

	J_1	J_2	J_3
M_1	$4-\varepsilon$	2	2
M_2	4	3	3
M_3	6	$6-\varepsilon$	$6-\varepsilon$

Note that M_1 is the cheapest machine for each job, M_2 is the second-cheapest, and M_3 is the most expensive one. We claim that, for any adaptive sequence (tree), the resulting subgame perfect equilibrium will have the following structure:

– The first job *to move* will choose M_1, the second one M_2, and the third one M_3.

This implies that the cost is at least $6-\varepsilon$, and thus the lower bound holds by taking ε small enough. To prove the claim above, we observe the following. Suppose the first two jobs to move choose M_1 and M_2. Then, the third job will choose M_3 as in some among the first two jobs chooses M_3 (otherwise we are done). Assume the third job will always choose the machine left empty. Then, for the second job it is always better to not join the same machine of the first one

Remark 5. The analysis in the proof of Theorem 4 cannot be extended to more than 2 machines even if we assume that the machines are identical. The following example shows this. Assume that we have $m = 3$ machines, the initial loads on these machines are $(0, 2, 6)$ and there are 3 jobs left to be assigned with processing times $7, 5$ and 5. Note that the constrained optimum here is $(10, 9, 6)$, that is the first job with processing time 7 gets assigned to the second machine M_2, while both jobs with processing times 5 and 5 get assigned to machine M_1. On the other hand, if any of these players chooses different machine their cost is strictly decreasing in the subgame perfect equilibrium solution. We did not find any example where the claim of Theorem 4 is wrong for more than 2 identical machines, unlike the case of unrelated machines.

6 Conclusions

In this paper we disproved a conjecture from [6] and gave a linear lower bound construction. On the other hand, for the best sequence of players we proved a linear upper bound, moreover we proved an existense of a sequential extensive game which gives an optimum solution. One possible direction for a future research is to prove or disprove that sequential price of stability is 1 for identical machines. Up to this point, we were unable to prove it or find a counterexample.

Acknowledgments. We thank Paul Dütting for valuable discussions.

References

1. Elliot Anshelevich, Anirban Dasgupta, Jon M. Kleinberg, Éva Tardos, Tom Wexler, and Tim Roughgarden. The price of stability for network design with fair cost allocation. *SIAM J. Comput.*, 38(4):1602–1623, 2008.
2. Vittorio Bilò, Michele Flammini, Gianpiero Monaco, and Luca Moscardelli. Some anomalies of farsighted strategic behavior. *Theory Comput. Syst.*, 56(1):156–180, 2015.
3. Bo Chen, Chris N. Potts, and Gerhard J. Woeginger. *Handbook of Combinatorial Optimization*, chapter A Review of Machine Scheduling: Complexity, Algorithms and Approximability, pages 1493–1641. Springer, 1999.
4. Jasper de Jong and Marc Uetz. The sequential price of anarchy for atomic congestion games. In *Proc. of the 10th International Conference on Web and Internet Economics (WINE)*, volume 8877 of *LNCS*, pages 429–434, 2014.
5. Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.
6. Refael Hassin and Uri Yovel. Sequential scheduling on identical machines. *Oper. Res. Lett.*, 43(5):530–533, 2015.
7. Elias Koutsoupias and Christos H. Papadimitriou. Worst-case equilibria. In *Proc. of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1563 of *LNCS*, pages 404–413, 1999.
8. Renato Paes Leme, Vasilis Syrgkanis, and Éva Tardos. The curse of simultaneity. In *Proc. of Innovations in Theoretical Computer Science (ITCS)*, pages 60–67, 2012.
9. Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 2012.