Sequential Solutions in Machine Scheduling Games

Paul Giessler¹, Akaki Mamageishvili², Matúš Mihalák⁴, and Paolo Penna²

Department of Data Science and Knowledge Engineering, Maastricht University, Netherlands
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. 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 show that the sequential price of anarchy grows at least linearly with the number \(n\) of players, i.e., \(\text{SPoA} = \Omega(n)\). 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

We consider the classical optimization problem of scheduling jobs on 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.

In its game-theoretic version, 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. Such an equilibrium might have a rather high social cost, that is, the makespan of the corresponding schedule\(^1\) is not guaranteed to be the optimal one (see Example\(^1\) below).

\(^1\) 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.
The inefficiency of equilibria in games is a central concept in algorithmic game theory, as it quantifies the efficiency loss resulting from a selfish behavior of the players. The two most popular notions are probably the price of anarchy (PoA) \[8\] and the price of stability (PoS) \[2\], which intuitively consider the most pessimistic and most optimistic scenario:

- The price of anarchy compares the worst equilibrium with the optimal social cost;
- The price of stability compares the best equilibrium with the optimal social cost.

The price of anarchy corresponds to the situation (anarchy) in which there is no authority, and players converge to some equilibrium by themselves. In the price of stability, one can imagine that (somehow) one can suggest the players how to play, and if that is an equilibrium then they will indeed follow the suggestion. This notion quantifies the minimal efficiency loss if, for example, no equilibrium is actually a social optimum.

One issue with these notions is that, for some games, the bounds are either too pessimistic or too optimistic, and that certain equilibria are perhaps difficult to justify:

**How did the players find such “strange” equilibria?**

Motivated by these issues, and observing that in some games “abnormal” equilibria result in unbounded price of anarchy, in \[9\] authors introduce the sequential price of anarchy (SPoA). This is essentially the variant in which players, instead of choosing their strategies simultaneously, play sequentially taking their decisions based on the previous choices and also knowing the order of players that will make play. Formally, this corresponds to an extensive-form game, and the corresponding equilibrium concept is called a subgame-perfect equilibrium. Players always choose their strategy deterministically.

**Example 1 (two jobs on two unrelated machines).** Consider two jobs and two unrelated machines, where the processing times are given by the following table \[9\]:

|       | job 1 | job 2 |
|-------|-------|-------|
| machine 1 | 1     | \(\ell\) |
| machine 2 | \(\ell\) | 1     |

The allocation represented by the gray box is a (pure Nash) equilibrium in the resulting strategic game (if a job moves to the other machine, its own cost increases from \(\ell\) to \(\ell + 1\)). The optimum makespan in 1 (swap the allocation), this example shows that the PoA for two unrelated machines is unbounded.

**1.1 Prior results (SPoA for unrelated machines)**

The first bounds on the SPoA for unrelated machines have been obtained in \[9\]:

\[n \leq \text{SPoA} \leq m \cdot 2^n.\]

Therefore, SPoA is constant for constant number of machines and jobs, while PoA is unbounded even for two jobs and two machines. The large gap in the previous bound...
naturally suggests the question of what happens for many jobs and many machines. This was addressed by [3] which improved significantly the prior bounds:

$$2^{O(\sqrt{n})} \leq \text{SPoA} \leq 2^n.$$  

At this point one should note that these lower bounds use a non-constant number of machines. In other words, it still might be possible that for constant number of machines the SPoA is constant. For two machines, [7] proved a lower bound $\text{SPoA} \geq 3$, and in the same work the authors made the following quite plausible conjecture:

**Conjecture 1.** For two unrelated machines, the SPoA = 3 for any number of jobs [7].

### 1.2 Our contributions

We disprove Conjecture[1] by showing that in fact the SPoA on two machines is not even constant. Indeed, it must grow linearly and the conjecture fails already for few jobs:

- For five jobs we have $\text{SPoA} \geq 4$ (Theorem[1]);
- In general, it holds that $\text{SPoA} \geq \Omega(n)$ (Theorem[2]).

The result is also interesting because it says that the SPoA is non-constant already for two machines (as the number of jobs grows). Moreover, it implies a strong separation with the identical machines case where $\text{SPoA} \leq 2 - \frac{1}{m}$, for any number $m$ of machines [7].

The idea that an “authority” can suggest the order in which players move so to induce a good equilibrium, can be viewed as the price of stability (PoS) for these sequential games. We introduce this notion in two variants (a weaker and a stronger):

- **Sequential Price of Stability (SPoS).** The authority can choose the order of the players moves. This order determines the tree structure of the corresponding game.
- **Adaptive Sequential Price of Stability (adaptive SPoS).** The authority decides the players moves adaptively according to the choices made at each step.

The study of these two notions for two unrelated machines is also motivated by our lower bound, and by the lack of any good upper bound on this problem. We prove the following upper bounds for two unrelated machines (Theorems[3] and [4]):

$$\text{SPoS} \leq \frac{n}{2} + 1 \quad \text{adaptive SPoS} = 1$$

The next natural question is to consider three or more machines. Here we show an impossibility result, namely adaptive SPoS $\geq 3/2$ already for three machines (Theorem[5]). That is, even with the strongest type of adaptive “authority”, it is not possible to achieve the optimum. This shows a possible “disadvantage” of having players capable of complex reasoning, like in extensive form games. In the classical strategic-games setting, where we consider pure Nash equilibrium, the optimum is an equilibrium, that is, PoS = 1 for any number of machines and jobs.
Further related work. The sequential price of anarchy has been studied for other games [6,1,5]. There is an enormous amount of literature on the classical algorithm-theoretic research on machine scheduling, see, e.g., the textbook by Pinedo [10] and the survey by Chen, Potts, and Woeginger [4] for the fundamental results and further references.

2 Preliminaries

In unrelated machine scheduling there are $n$ jobs and $m$ machines, and the processing time of job $j$ on machine $i$ is denoted by $p_{ij}$. A solution (or schedule) consists of an assignment of each job to one of the machines, that is, a vector $s = (s_1, \ldots, s_n)$ where $s_j$ is the machine to which job $j$ is assigned to. The load $l_i(s)$ of a machine $i$ in schedule $s$ is the sum of the processing times of all jobs allocated to it, that is, $l_i(s) = \sum_{j: s_j=i} p_{ij}$. The social cost of a solution $s$ is the makespan, that is, the maximum load among all machines.

Each job $j$ is a player who attempts to minimize her own cost $cost_j(s)$, that is, the load of the machine she chooses: $cost_j(s) = l_{s_j}$. Every player $j$ decides $s_j$, the assignment of job $j$ to a machine. The combination of all players strategies gives a schedule $s = (s_1, \ldots, s_n)$.

In the extensive-form version of these games, players play sequentially, and they decide their strategies based on the choices of the previous players and knowing that the other players are (also) rational. The game is a complete information game. As players enter the game sequentially, they can compute their optimal moves by the so-called backwards induction: the last player makes her move greedily, the player before the last makes the move also greedily (taking into account what the last player will do) depending on the moves of the last, and so on. Any game of this type can be modeled by a decision tree, which is a rooted tree where the non-leaf vertices are labeled with players, and edges correspond to the strategies (see appendix for examples of decision trees). Each leaf corresponds to a solution (schedule), which is simply the strategies on the unique leaf-to-root path. Given the processing times $P = (p_{ij})$, the players can compute the loads on the machines in each of the leaves. In case of ties, all players know the deterministic tie-breaking rules of all the other players. A player can then decide what the final outcome would be for each of her choices. 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, given the previous choices up to this point. We usually represent the strategies (edges) that are chosen by players in the subgame perfect equilibrium in bold, and the other strategies as dashed edges.

Notation and formal definitions. We consider $n$ jobs and $m$ machines, denoted by $J = (J_1, J_2, \ldots, J_n)$ and $M = (M_1, M_2, \ldots, M_m)$ respectively. The processing times are given by a matrix $P = (p_{ij})$, with $p_{ij}$ being the processing time of job $J_j$ on machine $M_i$. The set of all such nonnegative $n \times m$ matrices is denoted by $P_{nm}$ and it represents the possible instances of the game. For any $P \in P_{nm}$ as above, we denote by $T_{n,m}$ the set of all possible depth-$n$, complete $m$-ary decision trees where each path from the root to a leaf contains every job (player) exactly once. The whole game (and the resulting
subgame perfect equilibrium) is fully specified by $P$, $T$, and the tie-breaking rule used by the players. The most general – worst case – scenario is that ties are arbitrary (see Definition 1). In the following we do not specify the dependency on the ties, and simply denote by $SPE(P, T)$ the cost (makespan) of the subgame perfect equilibrium of the game. One type of worst-case analysis is to assume the players’ order to be adversarial, and the tree $T$ being chosen accordingly. This is the same as saying that players arrive in a fixed order (say $J_1, J_2, \ldots, J_n$) and their costs $P$ is chosen in adversarial fashion. In this case we simply write $SPE(P)$ as the tree structure is fixed. For a fixed order $\sigma$ (a permutation) of the players, and costs $P$, we also write $SPE(P, \sigma)$ to denote the quantity $SPE(P, T)$ where $T$ is the tree resulting from this order $\sigma$ of the players (see Figure 2a). The optimal social cost (makespan) is denoted by $OPT(P)$.

We next introduce formal definitions to quantify the inefficiency of subgame perfect equilibria in various scenarios (from the most pessimistic to the most optimistic). The sequential price of anarchy ($SPoA$) compares the worst subgame perfect equilibrium with the optimal social cost,

$$SPoA = \sup_{P \in P_{nm}} \frac{SPE(P)}{OPT(P)}.$$ 

In the sequential price of stability ($SPoS$), we can choose the order $\sigma$ in which players play depending on the instance $P$. The resulting subgame perfect equilibrium has cost $SPE(P, \sigma)$, which is then compared to the optimum,

$$SPoS = \sup_{P \in P_{nm}} \min_{\sigma \in S_n} \frac{SPE(P, \sigma)}{OPT(P)},$$

where $\sigma$ ranges over all permutations $S_n$ of the $n$ players. Adaptive Sequential Price of Stability (adaptive $SPoS$). In this case, we can choose the whole structure of the tree, meaning that for each choice of a player, we can adaptively choose which player will play next. The adaptive price of stability is then defined as

$$\text{adaptive } SPoS = \sup_{P \in P_{nm}} \min_{T \in T_{nm}} \frac{SPE(P, T)}{OPT(P)}.$$ 

Note that by definition $\text{adaptive } SPoS \leq SPoS \leq SPoA$.

3 Linear lower bound for $SPoA$

In this section we consider the sequential price of anarchy for two unrelated machines. In [2] the authors proved a lower bound $SPoA \geq 3$ for this case, and they conjectured that this was also a tight bound. We show that unfortunately this is not the case: Already for five jobs, $SPoA \geq 4$, and with more jobs the lower bound grows linearly, i.e., $SPoA = \Omega(n)$.

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:

\begin{equation}
\begin{array}{c|ccccc}
 & J_1 & J_2 & J_3 & J_4 & J_5 \\
\hline
M_1 & 3 - 11\varepsilon & \varepsilon & \varepsilon & 1 - 2\varepsilon & 2 - 8\varepsilon \\
M_2 & \varepsilon & 2 - 9\varepsilon & 2 - 8\varepsilon & 1 - 2\varepsilon & 1 - 2\varepsilon \\
\end{array}
\end{equation}

(1)

Note that the optimum has cost 1, while the subgame perfect equilibrium costs $4 - 13\varepsilon$. By letting $\varepsilon$ 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 $M_1$:

\begin{equation}
\begin{array}{c|cc}
 & J_1 & J_2 \\
\hline
M_1 & 3 - 11\varepsilon & \varepsilon \\
M_2 & \varepsilon & 2 - 9\varepsilon \\
\end{array}
\end{equation}

\Rightarrow

\begin{equation}
\begin{array}{c|cc}
 & J_4 & J_5 \\
\hline
 & 1 - 2\varepsilon & 2 - 8\varepsilon \\
 & 1 - 2\varepsilon & 1 - 2\varepsilon \\
\end{array}
\end{equation}

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 $2 - 7\varepsilon$:

\begin{equation}
\begin{array}{c|c}
 & J_1 & J_2 \\
\hline
M_1 & 3 - 11\varepsilon \\
M_2 & \varepsilon & 2 - 9\varepsilon \\
\end{array}
\end{equation}

\Rightarrow

\begin{equation}
\begin{array}{c|ccc}
 & J_3 & J_4 & J_5 \\
\hline
 & \varepsilon & 1 - 2\varepsilon & 2 - 8\varepsilon \\
 & 2 - 8\varepsilon & 1 - 2\varepsilon & 1 - 2\varepsilon \\
\end{array}
\end{equation}

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 $3 - 10\varepsilon$:

\begin{equation}
\begin{array}{c|c}
 & J_1 \\
\hline
M_1 & 3 - 11\varepsilon \\
M_2 & \varepsilon \\
\end{array}
\end{equation}

\Rightarrow

\begin{equation}
\begin{array}{c|ccc}
 & J_2 & J_3 & J_4 \\
\hline
 & \varepsilon & 1 - 2\varepsilon & 2 - 8\varepsilon \\
 & 2 - 9\varepsilon & 2 - 8\varepsilon & 1 - 2\varepsilon \\
\end{array}
\end{equation}

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 similar arguments as above job $J_2$ prefers to choose machine $M_2$ and all players will choose the allocation in (1), the cost for $J_1$ in 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). \hfill \Box

Remark 1 (tie-breaking rule). 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 $\varepsilon$ terms, so that ties never occur.
Remark 2 *(faster linear program formulation).* Our first lower bound for \( n = 5 \) players has been obtained by solving a linear program, suggested in \([7]\). A crucial achievement for the speedup of the program is the discovery of the property that we describe in Appendix B. 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 internal nodes in the tree, the number of combinations that has to be generated can be reduced drastically. For example, for \( n = 5 \), the improvement is from \( 2 \cdot 10^9 \) to \( 6 \cdot 10^6 \).

### 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 \( \varepsilon \) 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 all the other players.

The following theorem gives our general lower bound:

**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       | \( k-1 \) | 0         | 0         | 0         | 1         |
| \( M_2 \) | 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 (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 (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 $J_{3k-2}$ and $J_{3k-1}$, 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 $J_4, \cdots, J_{3k-3}$ 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 $J_{3k-2}$ 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 $J_{3k-1}$ 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 $J_3, \cdots, J_{3k-3}$ get assigned to the machine where they have cost 0, and the last two jobs $J_{3k-2}$ and $J_{3k-1}$ choose machine $M_2$, here again we assume that player $J_{3k-2}$ 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.

See appendix for the discussion of the above tie-breaking rule.

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 $\text{SPoS}$. In particular, it is enough that the “authority” divides the players into two groups and puts 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 the first $k \leq \frac{n}{2}$ jobs, and machine $M_2$ gets all the other jobs:

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

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

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

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, corresponding to the next two claims.

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 $\{J_1, J_2, \ldots, J_k\}$, which came before player $J_d$, did not deviate. Machine $M_1$ has thus exactly the jobs that it gets in the optimum. If $J_d$ stays on $M_2$, her cost will be at most $OPT$ (in the worst-case, all subsequent jobs choose to stay on $M_2$). Moving to $M_1$ will in the end produce a schedule 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 $OPT$ (otherwise the new schedule has smaller makespan than $OPT$, a contradiction with the optimality), 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 $t = k + 1 - d$.

Proof (of Claim). The proof is by induction on $t$. For $t = 1$, the deviating player is the last in $\{J_1, \ldots, J_k\}$, i.e., $J_k$. Recall that if $J_k$ does not deviate, then by the previous claim, $J_k$ guarantees her cost to be at most $OPT$. Thus, if $J_k$ deviates to $M_2$, then in the resulting equilibrium schedule she cannot pay more, i.e., she pays at most $OPT$. We now argue that if $J_k$ deviates, $M_1$ will have, in the resulting equilibrium, load at most $2 \cdot OPT$. Clearly, in the resulting equilibrium, load on $M_3$ is at most $OPT$. Some of the jobs among $J_{k+1}, \ldots, J_n$ may be assigned to $M_2$ in this equilibrium. Moving them all to machine $M_2$ will result in load on $M_2$ being at most $2 \cdot OPT$ (since all these jobs are, in the optimum solution, on $M_2$). Hence, each of the jobs among $J_{k+1}, \ldots, J_n$ that decided to move to $M_1$ in the resulting equilibrium cannot have a worse cost than that of staying on $M_2$, which guarantees cost 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 $(t+1) \cdot \text{OPT}$, because otherwise she would stay on the second machine and pay at most $(t + 1) \cdot \text{OPT}$. Therefore, by induction we get that this sequence results into a solution which has a makespan at most $(k + 1) \cdot \text{OPT}$, which completes the proof.

The two claims above imply the theorem.

Remark 3. This result cannot be extended to three or more machines, because the third machine would change the logic of the proof. We can no longer assume that the players on the second machine in the optimum assignment can guarantee low costs for them by simply staying on that 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, which has a full control over the order of the players. It cannot only fix the complete order, but can also change the order depending on the decision that previous players made. At the same time, the players still have a complete freedom about choosing any strategy, and therefore, the best outcome for them. Yet, the players also know the whole decision tree, and thus the strategy with which the authority chooses the order. This model is the closest instantiation of a general extensive form game compared to the previously studied models in this paper. This way the authority has an option to punish players for deviating from the optimum path (path leading to social optimum) by placing different players after the deviating decisions of the deviating player. As a result, 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.

Proof (Main Idea). The main idea of the proof is as follows. Each internal node of a tree corresponds to a choice of some player, and the path (edges) to that node correspond to an allocation of a subset of players (the nodes on the node-to-root path). We consider the corresponding constrained optimum, that is, an allocation of all remaining jobs that minimizes the makespan, given the fixed allocation of the previous players. Among
these remaining players, we then find a particular one for which the constrained optimum is better than any constrained optimum if she deviates. If this player deviates, we can punish such deviation by letting the others implement the more expensive constrained optimum (by adaptively fixing their order).

Proof. For any node $h$ of the tree, let $S_h$ be the subset of players that appeared on the previous nodes (i.e., from the parent of $h$ up to the root), and let $A_h$ be the resulting allocation (described by the path). Let $opt_h$ be the constrained optimum, that is, the allocation of all remaining jobs $R_h$ which, combined with $A_h$, minimizes the resulting makespan. We now choose a suitable player $J^*(h)$ to put on node $h$, according to the following:

Claim. There exists a player $J^*(h)$ among the remaining players $R_h$ such that the following holds. If $J^*(h)$ deviates from the constrained optimum $opt_h$, then the new constrained optimum (if implemented) is more costly for $J^*(h)$.

Proof. Let $OPT_h$ be the makespan of $opt_h$. Without loss of generality, suppose that $OPT_h$ is determined by machine $M_2$. Consider any job $J' \in R_h$ that $opt_h$ allocates to the other machine $M_1$. Note that if no such job exists, then the constrained optimum can be implemented by any order of the remaining jobs $R_h$, since they prefer machine $M_2$, because of the definition of constrained optimum. Suppose we allocate $J'$ to machine $M_2$, and consider the new constrained optimum $opt_h'$. If the load on $M_2$ is now larger than $OPT_h$, then we choose $J^*(h) = J'$ as the player to move at node $h$ in the tree; if this player deviates to machine $M_2$, she gets a worse cost because the new constrained optimum will be implemented (backwards induction shows that the new constrained optimum will be implemented). Otherwise, if in the new constrained optimum the load on $M_2$ is smaller than $OPT_h$, then we choose $J^*(h)$ as follows. Since $OPT_h' \geq OPT_h$, the machine $M_1$ is the bottleneck, that is, its load is $OPT_h'$. This means that there is a job $J''$ which in this new constrained optimum is assigned to $M_1$, but in the original constrained optimum was on the other machine $M_2$. Then we choose $J^*(h) = J''$ because this job satisfies the condition of the claim: if $J''$ deviates to $M_1$, the resulting constrained optimum will be as expensive as $OPT_h' \geq OPT_h$, while staying on $M_2$ gives the cost $OPT_h$ to her.

At each node $h$, the chosen player $J^*(h)$ can either follow the constrained optimum, or deviate. Backwards induction guarantees that in either case the remaining players implement the resulting constrained optimum. The above claim implies that $J^*(h)$ does not deviate from the constrained optimum $opt_h$.

Surprisingly, the previous result cannot be extended to more than 2 machines:

**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:

\[
\begin{array}{c|c|c|c}
 & J_1 & J_2 & J_3 \\
---&---&---&---
M_1 & 4 - \varepsilon & 2 & 2 \\
M_2 & 4 & 3 & 3 \\
M_3 & 6 & 6 - \varepsilon & 6 - \varepsilon
\end{array}
\]
We distinguish two cases for the first player to move (the root of the tree), and show that in neither case the player will implement the optimum:

1. The first to move is $J_1$. This player will choose the cheapest machine $M_1$, because none will join this machine. Indeed, the second player to move will choose $M_2$ knowing that the last one will then choose $M_3$.

2. The first to move is $J_2$ or $J_3$. This player will choose $M_2$ and not $M_1$. Indeed, if the first player to move, say $J_2$, chooses $M_1$, then either (I) the other two follow also the optimum (which costs 4 to $J_2$) or (II) they choose another solution, whose cost is at least $6 - \varepsilon$. In the later case we have the lower bound. In case (I), we argue that choosing $M_2$ is better for $J_2$, because no other player will join: for the following players, being both on machine $M_1$ is already cheaper than being on $M_2$ with $J_2$.

In the first case, given that $J_1$ is allocated to $M_1$, the cheapest solution costs $6 - \varepsilon$. In the second case, one among $J_2$ or $J_3$ is allocated to $M_2$. The best solution in this case costs again $6 - \varepsilon$. This completes the proof.

Remark 4. 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, see the appendix.

6 Conclusions

In this paper we disproved a conjecture from [7] 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 existence 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. Anna Angelucci, Vittorio Bilò, Michele Flammini, and Luca Moscardelli. On the sequential price of anarchy of isolation games. *J. Comb. Optim.*, 29(1):165–181, 2015.
2. Elliot Anshelevich, Anirban Dasgupta, Jon M. Kleinberg, Eva 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.
3. Vittorio Bilò, Michele Flammini, Gianpiero Monaco, and Luca Moscardelli. Some anomalies of farsighted strategic behavior. *Theory Comput. Syst.*, 56(1):156–180, 2015.
4. 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.
5. José Correa, Jasper De Jong, Bart De Keijzer, and Marc Uetz. The curse of sequentiality in routing games. In *Proc. of the 11th International Conference on Web and Internet Economics (WINE)*, volume 9470 of *LMCS*, pages 258–271, 2015.
6. 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.
7. Refael Hassin and Uri Yovel. Sequential scheduling on identical machines. Oper. Res. Lett., 43(5):530–533, 2015.
8. 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.
9. 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.
10. Michael L. Pinedo. Scheduling: Theory, Algorithms, and Systems. Springer, 2012.
A Decision trees

Fig. 2: 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.

B Enhanced linear programming

It is worth mentioning that in [7] 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 constraints 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.

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 3. 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 observation 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 $J_n$.

Additional notation. We denote the nodes of a $T \in T_{nm}$ 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 sequences, in $H_i$ all the nodes correspond to exactly one player.

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 $P(h) = \{r, \ldots, p(p(h)), p(h), h\} = \{h' \in H | h \in t_{h'}\}$ 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 $h$: $B(i, h) = \{j \in J | p(h') \in H_j \land h' \in P(h) \land c(h') = i\}$. 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. 3: Conflicting configuration](image)

**Remark 5.** 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.

**C Tie breaking rule**

It is worth noting that in [6] 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. In contrast, for our case we solved linear programs with strict inequalities obtained from the subgame perfect equilibria tree structure given in the example from the proof of theorem 2 by introducing small $\varepsilon$ for strict inequalities. 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 $\varepsilon$’s and for the sake of exposition we do not present the exact solutions here.

**D Example for identical machines**

The following example shows that the analysis of Theorem 4 can not be extended to 3 machines even in the case of identical machines. 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.