Strategic Scheduling Games: Equilibria and Efficiency*†

Laurent Gourvès ‡ Jérôme Monnot ‡ Orestis A. Telelis §

1 Introduction

The growth and popularity of large-scale distributed systems and applications has spawned in recent years novel theoretical models for their design and performance optimization. Such systems include computational grids for computationally intensive scientific applications, peer-to-peer file sharing communities, distributed replication and content dissemination systems and, last but not least, the World Wide Web itself as an application deployed over the internet. The underlying infrastructure consists of a vast amount of interconnected resources, including processors, distributed storage, routers, network links, to name a few basic. Users of modern distributed environments are driven by individual constraints and are self-interested, in that they aim at optimizing their own objectives. They are completely autonomous and, to this end, their individual decisions and actions in utilizing the wealth of system resources may harm the global system performance. Taking into account the users' autonomous and self-interested behavior is therefore of crucial importance in designing an environment of distributed resources. This concern has greatly contributed in shaping the fast growing field of *Algorithmic Game Theory* (AGT) within the computer science community. AGT lies at the intersection of Theoretical Computer Science and Game Theory as a field concerned with computational considerations on Game Theoretical grounds. We refer the reader to [41] for the multiple facets of AGT. In this short contribution we examine an algorithmic game-theoretic approach to optimizing the performance of distributed systems utilized by autonomous self-interested users.

Koutsoupias and Papadimitriou proposed in their seminal work [36] that, in contexts modeled as strategic games [43] with autonomous self-interested agents, performance evaluation is carried in terms of the overall efficiency of the system at *Nash equilibrium* configuration. In this text, although some preliminaries are provided, we assume a certain level of familiarity with strategic games and the concept of Nash equilibrium; for an excellent introduction the reader is referred to [43, 41]. The overall efficiency of a Nash equilibrium configuration is naturally measured as an aggregate function over the agents' (players') individual objectives. This overall cost is often also referred to as *social cost*. In [36] the authors introduce the *worst-case coordination ratio* as the ratio of the cost of the socially most expensive Nash equilibrium over the socially optimum cost. This ratio has come to be widely referred to as the *Price of Anarchy*. The Price of Anarchy was largely exemplified in [36] on games emerging from the context of multi-processor scheduling. Each agent owns a job of a certain load and chooses one out of m machines to assign his job to.

*This work is supported by French National Agency (ANR), project COCA ANR-09-JCJC-0066-01.

†This work was carried out during the tenure of an ERCIM "Alain Benssousan" Fellowship Programme of the third author.

‡LAMSADE, CNRS FRE 3234, Université de Paris-Dauphine, 75775 Paris, France
(email: {laurent.gourves, monnot}@lamsade.dauphine.fr)

§Center for Mathematics and Computer Science (CWI) Amsterdam, The Netherlands (email: telelis@cwi.nl)

The cost of each agent is the total load assigned on the machine that he chooses, as a measure of the congestion incurred on that machine. The social cost is defined as the *maximum* cost over all players. If we view the players' costs as completion times then the social cost coincides with the *makespan* objective function, widely used in the scheduling literature. The makespan is simply the completion time of the machine that finishes last. The first bounds on the Price of Anarchy in strategic games were hence exemplified in [36] on a selfish scheduling game.

Strategic multiprocessor scheduling games are of importance in the wider context of managing distributed resources. The vast number of resources and users in modern distributed environments renders centralized deployment of global resource management policies expensive and inefficient. It is impossible for a centralized authority to optimize the assignment of loads to resources across the system. Such loads may of course be jobs to be processed, traffic to be forwarded across network links, data to be stored, replicated and disseminated etc. One reason is the scale of the system which makes information gathering and processing impossible; another is the autonomic nature of the loads' owners; they are expected to behave strategically anyway, even if they agree to comply with a suggestion, because they may misreport data associated with their loads. How can then a central authority manipulate the congestion caused on resources towards a cost efficient equilibrium? The idea is to specify local operational rules per resource, so as to *coordinate* users towards globally efficient system utilization. The deployment of such local rules was formalized in [19], under the notion of *coordination mechanisms* and demonstrated for scheduling and network congestion games. In scheduling, local policies are scheduling algorithms deployed on the machines; their set is referred to as a coordination mechanism. A policy decides the way that jobs are executed on each machine. This can be thought of as indirectly defining the completion time that agents experience on the machine they choose. As an example, in the game model studied in [36] the agents' completion times equal the total load assigned to the machines they choose. This cost can be interpreted as completion time under a scheduling policy that breaks down the jobs into infinitesimally small pieces and executes them in a round robin fashion, allocating each job execution time (processing power) proportionally to its size. This way all jobs complete at the same time, which coincides with the total load assigned to the machine.

The policies may be preemptive or non-preemptive, deterministic or randomized; a policy decides the order of execution of assigned jobs on each machine and may also introduce delays in a systematic manner. A coordination mechanism induces a strategic game, by affecting the players' completion times. Apart from achievement of cost-efficient Nash equilibria, there are some other natural requirements imposed on the choice of appropriate coordination mechanisms. In particular, we are mostly interested for games where Nash equilibria can be found efficiently (i.e. in polynomial time) or can be reached efficiently by the agents, when they follow a simple natural protocol of assigning their loads to resources. In their most general form, Nash equilibria are configurations under which agents may play a probability distribution over their assignment choices (machines), so as to minimize the expectation of their cost (completion time). In practice though, it is unlikely that any such agent will randomize over his strategy space, as his regret for a choice is directly observable on the actual immediate outcome. Furthermore, we are not aware of some simple protocol under which agents may reach a mixed Nash equilibrium. Thus we restrict ourselves to *pure* Nash equilibria, where every agent plays deterministically one action from his strategy space. The drawback with pure Nash equilibria is that they do not always exist in strategic games. Therefore, in choosing a coordination mechanism, we always impose the constraint that the induced game has pure Nash equilibria. A protocol by which these equilibria may be reached in particular classes of games is the *iterative best response* procedure, where in each iteration a player is chosen to improve his strategy given the strategy of all other players; when no player may perform an improvement any more, the configuration reached is a pure Nash equilibrium. Monderer and Shapley showed in

their influential work [38] that this procedure converges when the underlying strategic game belongs to the class of *potential games*. We are interested in coordination mechanisms that have pure Nash equilibria that can be found efficiently by algorithms/protocols that resemble closely the iterative best response algorithm.

Contents. In what follows we provide a brief account of selfish scheduling games, that have emerged by application of the notion of coordination mechanisms. We start by providing a few basics on strategic games in section 2 and definitions with respect to cost-efficiency of stable configurations (subsection 2.1). Our preliminary discussion is concluded with definitions and general remarks on scheduling games and coordination mechanisms in subsection 2.2. In section 3 we discuss several results regarding the most studied coordination mechanisms from the literature. These include the scheduling model (preemptive mechanism) introduced in [36] and further studied in [23, 20, 28, 9, 24, 46] (subsection 3.1), non-preemptive ordering mechanisms studied in [5, 19, 34, 22] (subsection 3.2) and a brief account of mechanisms for the most general scheduling environment of unrelated machines (subsection 3.5) [34, 10, 14]. In section 4 we revisit non-preemptive ordering mechanisms in the context of truthfulness [19, 18], where agents may misreport their processing loads to the mechanism so as to minimize their completion time in the outcome of the game.

Subsequently, we illustrate the design of coordination mechanisms and concerns raised by the introduction of *setup overheads*, additionally to the loads placed by autonomous users on the resources (machines) of the system [30]. The model is introduced in section 5. In section 5.1 we provide an analysis of the mechanism of [36] on this model. Then we introduce a class of mechanisms in section 6 for this model, for which we prove existence of pure Nash equilibria and a lower bound on their performance on identical machines. In section 6.2 we identify and analyze an optimum mechanism out of this class. We conclude our presentation by mentioning important open research problems for coordination mechanisms.

2 Preliminaries and Notation

Strategic Games and Equilibria A *strategic game* Γ is a tuple $\langle N, (\Sigma_j)_{j\in N}, (c_j)_{j\in N}\rangle$, where N is the set of players and Σ_j is the *set of actions* for player j. In general, the *strategy space* of player j consists of all possible probability distributions over his set of actions Σ_j [43]. A probability distribution is then referred to as a *mixed strategy*. In this text we will be studying deterministic players, which play *pure* strategies, i.e. an element of Σ_j with probability 1. Then Σ_i coincides with the strategy space of each player i and will be referred to as such. A strategy for player j will be an element of Σ_j, denoted by s_j. A *strategy profile* or *configuration* - denoted by $s = (s_1, \ldots, s_n)$ - is a vector of strategies, one for each player. The set of all strategy profiles is $\Sigma = \times_{j\in N}\Sigma_j$. Every player j has an individual *cost function* $c_j : \Sigma \to \mathbb{R}$ which depends on the strategy profile. We use the standard notation s_{-j} for $(s_1, \ldots, s_{j-1}, s_{j+1}, \ldots, s_n)$. For any subset of players $J \subseteq N$ we generalize this notation to s_{-J}, to refer to the part of s restricted to strategies of players in $N \setminus J$. We use s_J to refer to the part of s restricted to strategies of players in J. For any subset of players $J \subseteq N$, a strategy profile s, and s'_J such that $s'_j \neq s_j$ for all $j \in J$, let $s' = (s_{-J}, s'_J)$ refer to a new strategy profile under which $s'_j = s_j$ for all $j \in N \setminus J$.

The most prominent notion for describing the configurations that are *outcomes* (solutions) of a strategic game is the *Nash* equilibrium [43, 40]. In its generality it concerns players that may randomize over their action spaces Σ_i; we will restrict here to configuration outcomes that are *pure* Nash equilibria. These concern players that play deterministically a single element out of Σ_j. The Nash equilibrium expresses rationality of players, in that they select the strategy (action), that

induces the lowest individual cost for each of them, given the strategies of the rest. Formally:

Definition 1 *A strategy profile s is a pure Nash equilibrium if for every player $j \in N$ and for every strategy $s'_j \in \Sigma_j \setminus \{s_j\}$, $c_j(s) \leq c_j(s_{-j}, s'_j)$.*

Thus s is stable since no unilateral deviation from s is rational. We will use the abbreviation PNE for Pure Nash Equilibria. Let us note that PNE are not guaranteed to exist in strategic games; in contrast, John Nash's celebrated theorem proves that *mixed* strategy Nash equilibria always exist [40]. The notion of *Strong Equilibrium* (SE), due to Aumann [7] extends the notion of PNE and prescribes stability of configurations against pure joint/coalitional deviation of any subset of agents:

Definition 2 [7, 1] *A strategy profile s is a strong equilibrium if for every subset of players $J \subseteq N$ and every configuration s', where $s'_j \in \Sigma_j \setminus \{s_j\}$ for all $j \in J$, there is at least one player $j_0 \in J$ with $c_{j_0}(s) \leq c_{j_0}(s_{-J}, s'_J)$.*

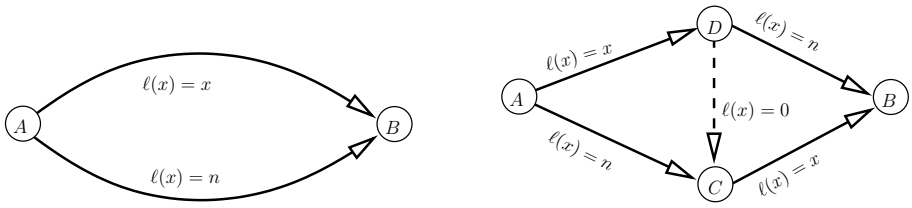
Every SE is also a PNE, because it is by definition resilient to deviations of trivial *singleton* coalitions, i.e. unilateral pure deviations; to this end it is also a Nash equilibrium, since every mixed (randomized) deviation of any single player is a linear combination of pure deviations. No such unilateral deviation may be profitable for the player under a PNE or SE, since there is no pure unilateral deviation that is profitable for him. For the sake of completeness however, we note that a strong equilibrium is *not* generally resilient to mixed deviations of non-trivial coalitions [1]; the concept only extends PNE with respect to *pure* deviations of coalitions. See [45] for further extensions that may handle also mixed deviations.

2.1 Social Cost-Efficiency of Equilibria

We will be interested in quantifying the overall cost efficiency of stable configurations for strategic games, i.e. configurations that are *Pure Nash equilibria* or *Strong Equilibria*. The *social cost* of a configuration s for a strategic game Γ characterizes how costly s is, aggregatively over the set of players. We use $C(s)$ to denote the social cost. Most commonly $C(s)$ is defined as $\sum_j c_j(s)$ or by $\max_j c_j(s)$. The definition of the social cost is not fixed because it depends crucially on the strategic situation that the game captures. The social cost is minimized for some configurations that are referred to as *social optima*. These are not necessarily PNE or SE.

Let us give two famous examples which motivate the need for quantification of the social cost-efficiency of equilibria. Consider a network $G = (V, E)$ (e.g. a road network), where V contains a source A and a destination B, and E is the set of arcs (roads). A large *even* number of n players (e.g. cars), each incurring *unit* traffic volume, need to travel from A to B. Every arc e has a latency function ℓ_e whose argument is the total traffic volume traversing e, and which expresses how long this volume takes to pass through arc e. The strategy space of each player is the set of all paths from A to B and his unique goal is to minimize his total transit time. Given a configuration s, let $x_e(s)$ denote the number of players using arc e, i.e. $x_e(s) = |\{j | e \in s_j\}|$. A player's cost for a source to destination path P is $c_j(s) = \sum_{e \in P} \ell_e(x_e(s))$. Define the social cost to be $C(s) = \sum_j c_j(s) = \sum_{e \in E} x_e(s) \ell_e(x_e(s))$.

The Pigou Example. Pigou's example [44] shows how the players, guided by their individual cost functions, can ruin the social cost. The network is such that $V = \{A, B\}$ and two parallel arcs link A and B (see Figure 1a for an illustration). The top link is a narrow – hence sensitive to traffic variation – road with latency function $\ell(x) = x$, where x denotes the amount of traffic on this link.



(a) Pigou Example. (b) Braess Paradox.

Figure 1: Two classical examples that motivate quantification of efficiency of equilibria.

The bottom link is a large enough road that can accommodate any traffic volume in n time units. Observe that the social cost is minimized when $\frac{n}{2}$ players play the bottom link and $\frac{n}{2}$ players play the upper link. Let s^* denote this configuration and then, $C(s^*) = \left(\frac{n}{2}\right)^2 + \frac{n^2}{2} = \frac{3n^2}{4}$. Notice that s^* is not a PNE; any player playing the bottom link has individual cost n. Hence the player prefers to switch to playing the upper link at an individual cost $\frac{n}{2} + 1$. Now consider a configuration s where all players play the upper link. In this case the individual cost of every player n and the social cost $C(s)$ is equal to n^2, that is, $\frac{4}{3}$ times greater than $C(s^*)$. Configuration s is a PNE, because any player that may deviate to the bottom link, will not decrease his individual latency below n.

Braess' Paradox. In an effort to mitigate the impact of selfish behavior on the social cost, one may be tempted to add some efficient facilities. Braess' paradox [12, 13] goes against intuition since it shows that adding some facilities may damage the social welfare. The network is as depicted on Figure 1b, without the dashed arc. There are two paths and the configuration which splits evenly the traffic on those paths is the only equilibrium, inducing a social cost of $\frac{3n^2}{2}$. Now suppose that a very fast road is built from node D to C, such that every player using that arc at D joins immediately C, ignoring the congestion on the arc (see the dashed arc on Figure 1b). Hence the previous solution is not an equilibrium anymore since the path (A, D, C, B) is more profitable than (A, D, B) or (A, C, B). However the only equilibrium in the augmented network is when all players adopt (A, D, C, B), inducing a social cost of $2n^2$.

To quantify the inefficiency of stable configurations in comparison to socially optimum ones, Koutsoupias and Papadimitriou introduced in [36] the notion of the *worst-case coordination ratio*, that became widely known as the *Price of Anarchy*; this is the worst-case ratio of the social cost of the most expensive Nash equilibrium over the socially optimum cost.

Definition 3 *Let Γ denote the set of all game instances for particular strategic game, as defined by its parameters. Let $I \in \Gamma$ refer to any particular instance of the game, when the parameters are instantiated. Let $\mathcal{E}(I)$ denote the set of all stable configurations and for every configuration $s \in \mathcal{E}(I)$ its social cost is denoted by $C(s)$. Finally, let s^* be any configuration that minimizes the social cost C, i.e. $C(s^*) \leq C(s)$ for every configuration s. The Price of Anarchy of Γ, referred to as $PoA(\Gamma)$ is:*

$$PoA(\Gamma) = \sup_{I \in \Gamma} \max_{s \in \mathcal{E}(I)} \frac{C(s)}{C(s^*)}$$

When the stability concept used is PNE, we simply refer to the Price of Anarchy. When we are interested in SE, we refer to the *Strong* Price of anarchy, abbreviated as $SPoA$. We will drop the argument Γ, since the set of game instances we refer to will be clear from the context. The *Price of*

Stability (PoS) was introduced in [4] as the worst-case ratio of the social cost of the least expensive Nash equilibrium over the socially optimum cost:

$$PoS(\Gamma) = \sup_{I \in \Gamma} \min_{s \in \mathcal{E}(I)} \frac{C(s)}{C(s^*)}$$

This quantifies the efficiency that selfish players may achieve if they coordinate. Coordination is expressed also in terms of strong equilibria, where instability is caused by *joint* coordinated deviations of agents. Ultimately, because every strong equilibrium is a PNE, we have that $PoS(\Gamma) \leq SPoA(\Gamma) \leq PoA(\Gamma)$.

2.2 Selfish Scheduling and Coordination Mechanisms

We consider an environment of m resources that we refer to as machines, indexed by $i \in \mathcal{M} = \{1, \ldots, m\}$ and n jobs $j \in \mathcal{J} = \{1, \ldots n\}$. Each job is owned by a self-interested player, thus our set of players will be $N = \mathcal{J}$. We use interchangeably the terms *job* and *player*. Every job j is associated with a processing load $\ell_j > 0$. Each player chooses a machine to assign his job to; thus, the strategy space of any player j will be $\Sigma_j = \mathcal{M}$. Execution of a job $j \in \mathcal{J}$ on a machine $i \in \mathcal{M}$ incurs *processing time* p_{ji} (e.g. processing cycles, or total CPU time). The processing time is decided by the characteristics of the machines environment as follows:

- *Identical Machines:* For every pair of distinct machines $i_1, i_2 \in \mathcal{M}$ we have $p_{j,i_1} = p_{j,i_2} = \ell_j$.
- *Uniformly Related Machines:* Each machine $i \in \mathcal{M}$ is associated with a speed factor $v_i \geq 1$ and $p_{ji} = \ell_j / v_i$.
- *Restricted Assignment Machines:* Every job $j \in \mathcal{J}$ is associated with a subset of machines $M_j \subseteq \mathcal{M}$ such that $p_{ji} = \ell_j$ for every machine $i \in M_j$ and $p_{ji} = \infty$ for $i \in \mathcal{M} \setminus M_j$.
- *Unrelated Machines:* The processing time of job $j \in \mathcal{J}$ on machine $i \in \mathcal{M}$ is p_{ji}.

For any player $j \in \mathcal{J}$ denote by s_j the assignment of job j to machine $s_j \in \mathcal{M}$. We refer to $s = (s_1, \ldots, s_n)$ as an *assignment*, *configuration* or *strategy profile*. Given an assignment s, player $j \in \mathcal{J}$ experiences an individual cost $c_j(s)$, which denotes the *completion time* of his job on his chosen machine $s_j \in \mathcal{M}$. We denote by $C_i(s)$ the completion time of machine $i \in \mathcal{M}$ under assignment s. Next we define the social cost function $C(s)$ that will quantify the cost efficiency of an assignment s; this is the maximum completion time of any player and coincides with the maximum completion time of any machine;

$$C(s) = \max_{j \in \mathcal{J}} c_j(s) = \max_{i \in \mathcal{M}} C_i(s)$$

$C(s)$ is commonly referred to in the scheduling literature as the *makespan* of the assignment s. Following the notation of Graham [31, 32] we will refer to the makespan minimization problems that correspond to the mentioned machine environments of identical, uniformly related, restricted assignment and unrelated machines as $P, Q, B, R||C_{\max}$ respectively. For the rest of our discussion, the socially optimum cost $C(s^*)$ will be the optimum value of the optimization problems $P, Q, B, R||C_{max}$.

Following [19] we define a coordination mechanism as the set of scheduling policies (algorithms), that are deployed on the machines; these indirectly define the individual completion times functions $c_j : \mathcal{M}^{|\mathcal{J}|} \to \Re_+$ of the players. Thus choosing a coordination mechanism is equivalent to choosing

completion time functions for players in $\mathcal{J}$. A coordination mechanism induces a strategic game for the players set $\mathcal{J}$. The design and study of coordination mechanisms concerns indirectly *choosing* a strategic game, that has cost-efficient - preferably pure - Nash equilibria (or even strong equilibria) which can be found efficiently.

A coordination mechanism can be characterized as *strongly local* or simply *local*, depending on the amount of information its constituting policies use to schedule the jobs assigned on the machines that the policies operate on. Under any assignment configuration s, if $J_i = \{j \in \mathcal{J} | s_j = i\}$ denotes the subset of jobs assigned to machine $i \in \mathcal{M}$, a policy deployed on i is *strongly local* if it uses only data of jobs in $j \in J_i$ with respect to machine i. This is better illustrated in the case of unrelated machines; a strongly local policy may use e.g. the index of any job $j \in J_i$ and its processing time p_{ji} on i. A policy that uses information of jobs in J_i that is relevant to machines other than i is referred to as simply *local*. Such a policy may e.g. use the processing time $p_{ji'}$ of $j \in J_i$, for any $i' \neq i$. A mechanism is local if it involves a local policy on at least one machine. It is strongly local if policies deployed on all machines are strongly local. Strongly local mechanisms are desirable of course, as their policies are applicable independently of the characteristics of the machines environment they are deployed upon. Few simply local mechanisms have been studied in the literature [10, 14].

3 An Overview of Coordination Mechanisms

We describe below and survey the known results with respect to the most studied mechanisms.

3.1 Load Balancing: The `Makespan` Mechanism

The `Makespan` mechanism applies on each machine a scheduling policy first studied in a game-theoretic setting in [36]. This policy breaks a job's time requirement into infinitesimally small pieces and executes pieces of all jobs assigned to the machine interchangeably, in a round-robin fashion. Each job is allocated an amount of processing cycles proportionally to its total processing time. All jobs finish at the same time on each machine i, that is $c_j(s) = C_i(s)$ for all $j \in \mathcal{J}$ such that $s_j = i$. Because every player's completion time is the completion time of the machine that he plays, scheduling games under the `Makespan` mechanism are widely mentioned also as *load balancing* games. Selfish choices in this context are motivated by and affect directly the balancing of the total load across the machines.

Pure Nash Equilibria: Existence & Convergence

`Makespan` induces a game that has PNE, because it belongs to the class of *potential games*, as shown in [23]. This class was introduced by Monderer and Shapley in [38]. Under any configuration of a potential game, if there is a player that has incentive to deviate so as to improve his individual utility (or cost), then such an improvement also incurs an improvement to a potential function that is associated with the game. As this function may only receive values out of a finite set (one per configuration of the game), its indefinite improvement is impossible. Eventually a configuration is reached where no agent has an incentive to deviate unilaterally. This configuration is a local optimum for the potential function and a PNE for the game. Using a similar potential function argument, Andelman *et al.* [1] also proved existence of *strong* equilibria under the `Makespan` mechanism for the most general environment of unrelated machines. We further generalize their proof in section 5.1 to prove the same result for even more general scheduling games (Theorem 3).

The existence of a potential function that is improved along with every selfish improvement performed by each player suggests a simple procedure for finding PNE: initialize an arbitrary con-

figuration and have iteratively an arbitrary player optimize his strategy against the fixed strategies of all other players. This is known as *Iterative Best Response*. Rate of convergence of Iterative Best Response under the `Maskespan` mechanism was studied extensively by Even-Dar, Kesselman and Mansour [23]. They considered $R||C_{\max}$ scheduling with m machines and n agents of at most K different processing times on the machines. Apart from iterative best response they studied also *iterative improvements* algorithms, where each agent is not restricted to *best* responses, but merely to strictly improving his completion time. The authors show that even these less restricted iterative algorithms converge to PNE, by introducing a total ordering of all possible configurations of the $R||C_{\max}$ game. Any improvement is shown to lead to a configuration lower in the global ordering, hence the game is an *ordinal potential game*. For $R||C_{\max}$ the space of all possible configurations is shown to have cardinality $\min\{[O(\frac{n}{Km}+1)]^{Km}, m^n\}$ which is a general upper bound on the of iterative improvements needed for a PNE to be reached. An $O(4^P)$ bound is shown when the processing times on all machines are integers and P denotes the *worst-case* sum of processing times of jobs. Even-Dar *et al.* proved several more bounds for the $P,Q||C_{\max}$ cases, by using specific rules for choosing the agent that will perform an improvement next [23].

Let us note that computation of PNE for `Makespan` can be carried out efficiently, in polynomial time, for the $P||C_{\max}$ and $Q||C_{\max}$ models. The simple algorithm which assigns the longest unassigned job next to the machine on which it will finish first, given the current partial assignment, finds a PNE. We will study this algorithm for $P||C_{\max}$ under the more general model of *setup times* in subsection 6.1 (Theorem 6). This algorithm is also referred to as `LPT` scheduling; the reader is referred to [41] (chapter 20) for the case of $Q||C_{\max}$ which has an essentially similar proof as for the case of $P||C_{\max}$, even under the model of setup times examined in subsection 6.1. Gairing *et al.* devised in [27] algorithms for computing cost-efficient PNE for the case of restricted assignment, even when the machines are also uniformly related (a combination of $B||C_{\max}$ and $Q||C_{\max}$). Their algorithms are based on techniques of flow computation. For the case of $B||C_{\max}$ alone, they provide the first polynomial-time algorithm for computing a PNE with social cost within strictly less than 2 of the socially optimum cost.

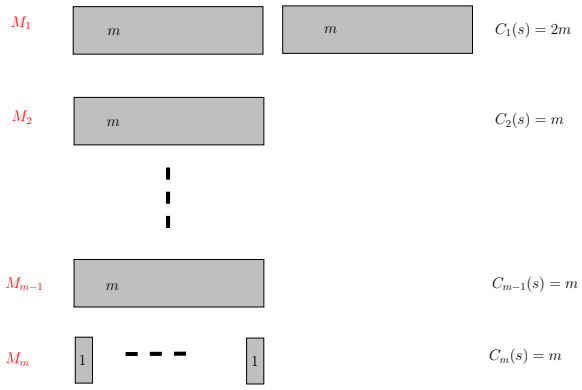
Efficiency of Equilibria

A considerable amount of literature on scheduling games and coordination mechanisms has been concerned with the study of efficiency of the `Makespan` mechanism. The results are summarized in table 1. For $P||C_{\max}$, results of Schuurman and Vredeveld [46] (and, earlier, of Finn and Horowitz [25]) imply that the PoA is $\frac{2m}{m+1}$. We provide a proof of this result, along with a tight lower bound for strong equilibria.

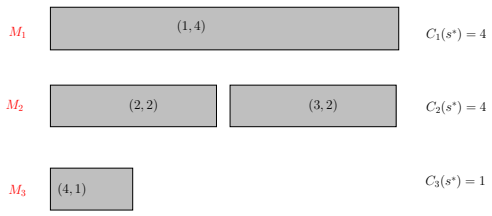
Theorem 1 *The Price of Anarchy of the* `Makespan` *mechanism for selfish scheduling on identical machines is* $2-\frac{2}{m+1}$ *and this bound is tight even for strong equilibria.*

Proof. Under an equilibrium assignment s, let i denote the machine that finishes last, i.e. $C_i(s) \geq C_{i'}(s)$ for all $i' \in \mathcal{M}$. Now consider j to be the job of the smallest processing load among the jobs assigned to i under s, i.e. $\ell_j = \min\{\ell_k : k \in \mathcal{J} : s_k = i\}$. Then, without loss of generality, there are at least two jobs assigned to machine i, i.e. there is $j' \neq j$ with $s_{j'} = i$. If this is not the case we obtain $C_i(s) = \ell_j \leq C(s^*)$, because $C(s^*) \geq \ell_j$ for every job $j \in \mathcal{J}$. Now if $s_{j'} = s_j = i$ and $\ell_j \leq \ell_{j'}$, we have $C(s) = C_i(s) \geq \ell_j + \ell_{j'} \geq 2\ell_j$. Because s is a PNE, j does not have incentive to switch to any machine $i' \neq i$, i.e. $c_j(s) = C_i(s) \leq C_{i'}(s) + \ell_j$ hence $C_{i'}(s) \geq C_i(s) - \ell_j \geq C(s) - \frac{1}{2}C(s) = \frac{1}{2}C(s)$. For the socially optimum configuration s^* we have:

$$C(s^*) \geq \frac{1}{m}\sum_j \ell_j = \frac{1}{m}\sum_{i'} C_{i'}(s) = \frac{1}{m}C_i(s) + \frac{1}{m}\sum_{i' \neq i} C_{i'}(s) \geq \frac{1}{m}C(s) + \frac{(m-1)}{2m}C(s)$$



(a) A strong equilibrium s for $P||C_{\max}$ under `Makespan`.



(b) A social optimum s^* which is not a Nash equilibrium. Every couple on the picture designates a job's index and load (e.g. $(1,4)$ for job 1 with load 4).

Figure 2: Worst-case SE and unstable social optimum for $P||C_{\max}$ under `Makespan`.

The latter yields $C(s) \leq \frac{2m}{m+1} C(s^*)$.

Now let us illustrate a lower bound that is also a strong equilibrium. For any instance with m machines, define $n = 2m$ jobs, with $\ell_1 = \cdots = \ell_m = m$ and $\ell_j = 1$, for $j = m+1, \ldots, 2m$. In the socially optimum assignment we have $s^*_j = j$ and $s_{m+j} = j$ for $j = 1, \ldots, m$. The makespan of this assignment is $C(s^*) = m + 1$. Consider now an equilibrium assignment s, under which $s_j = j$ for $j = 1, \ldots, m-1$, $s_m = 1$ and the m unit-length jobs are executed on machine m, see Figure 2a for an illustration. Then $C(s) = \ell_1 + \ell_m = 2m$. We claim that s is a *strong* equilibrium. By contradiction, assume that there are a coalition $\mathcal{J}' \subseteq \{1, \ldots, 2m\}$ and a state $s' = (s_{-\mathcal{J}'}, s'_{\mathcal{J}'})$ such that $\forall j \in \mathcal{J}'$, $c_j(s') < c_j(s)$. Notice that $c_j(s) = m = \ell_j$ for jobs $j = 2, \ldots, m-1$. Then jobs in $\{2, \ldots, m-1\}$ cannot belong to $\mathcal{J}'$. Also, jobs of $\mathcal{J}'$ must deviate to machines 1 or m since $c_j(s) \geq m$ for all jobs and jobs in $\{2, \ldots, m-1\}$ cannot move. Obviously, at least one job of 1 or m must deviate to machine m since otherwise none jobs of $\{m+1, \ldots, 2m\}$ can deviate and also the two jobs 1 and m cannot deviate both (otherwise, from previous argument they move together to machine m and then $c_1(s') \geq \ell_1 + \ell_m = 2m = c_1(s)$). Thus, we deduce that exactly one job, either job 1 or job m must deviate to machine m and also that some jobs of $\{m+1, \ldots, m\}$ must deviate to machine 1 (since $c_j(s) = m$ for $j \in \{m+1, \ldots, 2m\}$). These latter jobs of $\{m+1, \ldots, 2m\} \cap \mathcal{J}'$ increase their completion times, which is impossible. □

Andelman, Feldman and Mansour showed in [1] that there is always a socially optimum configuration that is a strong equilibrium under `Makespan`, even for $R||C_{\max}$. However, there may exist socially optimum configurations that are not even PNE. Consider the following instance with 3 identical machines and 4 jobs $\mathcal{J} = \{1, \ldots, 4\}$ where the processing times satisfy $\ell_1 = 4$, $\ell_2 = \ell_3 = 2$ and $\ell_4 = 1$. State $s^* = (1, 2, 2, 3)$ is a social optimum since $C(s^*) = C_1(s^*) = \ell_1$, see Figure 2b for an illustration. Now, if job 2 moves to machine 3 (corresponding to state $s' = (s^*_{-2}, 3)$), its completion time drops to $3 = c_2(s') < 4 = c_2(s^*)$. Hence, s^* is not a Nash equilibrium. When $m = 2$, the situation is more optimistic:

Proposition 1 *For the* `Makespan` *mechanism for selfish scheduling on $m = 2$ identical machines ($P||C_{max}$), any social optimum is a strong equilibrium.*

`Makespan` Mechanism [36]

Model	**PoA**	**SPoA**
$P\|\|C_{\max}$	$\frac{2m}{m+1}$ [25, 46]	$\frac{2m}{m+1}$ [25, 46]
$Q\|\|C_{\max}$	$\Theta\left(\frac{\log m}{\log\log m}\right)$ [20]	$\Theta\left(\frac{\log m}{(\log\log m)^2}\right)$ [24]
$B\|\|C_{\max}$	$\Theta\left(\frac{\log m}{\log\log m}\right)$ [9, 28]	$\Omega\left(\frac{\log m}{\log\log m}\right)$ [9, 28]
$R\|\|C_{\max}$	∞	m [1, 24]

Table 1: Performance of the `Makespan` Mechanism for Multiprocessor Scheduling.

Proof. Let s^* be an social optimum, and assume that machine 1 is the most loaded machine, i.e., $C(s^*) = C_1(s^*) \geq C_2(s^*)$. Moreover, let $\mathcal{J}_i = \{j \in \mathcal{J} | s^*_j = i\}$ for $i = 1, 2$. Finally, let $\mathcal{J}'$ be a coalition and consider the state $s' = (s^*_{-\mathcal{J}'}, s'_{\mathcal{J}'})$ where $s'_j = 1$ for $\mathcal{J}' \cap \mathcal{J}_2$ and $s'_j = 2$ for $\mathcal{J}' \cap \mathcal{J}_1$. We affirm that there is $j \in \mathcal{J}'$ such that $c_j(s') \geq c_j(s^*)$ leading to the conclusion that s^* is a strong equilibrium.

- If $\mathcal{J}' \subseteq \mathcal{J}_2$, then let $j_2 \in \mathcal{J}'$. We have: $c_{j_2}(s') = C_1(s^*) + \sum_{j\in\mathcal{J}'} \ell_j > C_1(s^*) \geq C_2(s^*) = c_{j_2}(s^*)$.
- If $\mathcal{J}' \subseteq \mathcal{J}_1$, then let $j_1 \in \mathcal{J}'$. By contradiction, suppose that: $\forall j \in \mathcal{J}'$, $c_j(s') < c_j(s^*)$. In particular, we obtain: $C_2(s') = c_{j_1}(s') < c_{j_1}(s^*) = C_1(s^*)$. Also we get: $C_1(s') = C_1(s^*) - \sum_{j\in\mathcal{J}'} \ell_j < C_1(s^*)$. Thus, $C(s') = \max\{C_1(s'); C_2(s')\} < C_1(s^*) = C(s^*)$, which is a contradiction since s^* is supposed to be a social optimum.
- If $\mathcal{J}' \cap \mathcal{J}_1 \neq \emptyset$ and $\mathcal{J}' \cap \mathcal{J}_2 \neq \emptyset$, then let $j_i \in \mathcal{J}' \cap \mathcal{J}_i$ for $i = 1, 2$. Finally, let $\delta = \sum_{j\in\mathcal{J}'\cap\mathcal{J}_1} \ell_j - \sum_{j\in\mathcal{J}'\cap\mathcal{J}_2} \ell_j$. We get that $c_{j_1}(s') = c_{j_2}(s^*) + \delta = C_2(s^*) + \delta$ and $c_{j_2}(s') = c_{j_1}(s^*) - \delta = C_1(s^*) - \delta$. Hence $c_{j_1}(s') + c_{j_2}(s') = C_1(s^*) + C_2(s^*)$.

  Now, if $\forall j \in \mathcal{J}'$, $c_j(s') < c_j(s^*)$, we deduce that $c_{j_1}(s') < c_{j_1}(s^*) = C_1(s^*)$ and $c_{j_2}(s') < c_{j_2}(s^*) = C_2(s^*)$. Thus, $c_{j_1}(s') + c_{j_2}(s') < C_1(s^*) + C_2(s^*)$, which is a contradiction with the previous equality.

In conclusion, for any $\mathcal{J}' \subseteq \mathcal{J}$, there exist $j \in \mathcal{J}'$ such that $c_j(s') \geq c_j(s^*)$ or equivalently s^* is a strong equilibrium. □

For uniformly related machines ($Q||C_{\max}$), Czumaj and Vöcking proved in [20] that `Makespan` achieves $PoA = \Theta\left(\frac{\log m}{\log\log m}\right)$. The same bound was shown independently by Gairing *et al.* [28] and Awerbuch *et al.* [9] for restricted assignment machines. We discuss the lower bound for both cases; for the case $B||C_{\max}$ we show that the lower bound is true for strong equilibria as well. In both cases we will use the *Gamma Function* Γ, which is defined by $\Gamma(k+1) = k!$ for any integer $k \geq 1$. Both Γ and its inverse, Γ^{-1} are monotone increasing. It is known that $\Gamma^{-1}(k) = \Theta\left(\frac{\log k}{\log\log k}\right)$ as k grows.

Proposition 2 *The Price of Anarchy of* `Makespan` *for uniformly related machines is* $\Omega\left(\frac{\log m}{\log\log m}\right)$.

Proof. Consider $k+1$ disjoint groups of machines, G_r, $r = 0, \ldots, k$, so that $\mathcal{M} = \cup_{r=0}^{k} G_r$. For $r = 0, \ldots, k$, group G_r contains $|G_r| = \frac{k!}{r!}$ machines, where $0! = 1$. Each machine in group G_r has speed 2^r. Furthermore We make r jobs of processing length 2^r, for every machine in group G_r. Consider the assignment s under which r jobs of processing length 2^r each are assigned to each of

the machines in group G_r, for $r = 1, \ldots, k$. Notice that no jobs are assigned to any of the machines in group G_0. We claim that this assignment is a PNE. Notice that for every machine $i \in G_r$ we have $C_i(s) = r$, for $r = 0, \ldots, k$. Hence the only potentially profitable deviation for a job j with $s_j \in G_r$ for $r \geq 1$, is to play $s'_j \in G_{r'}$ with $r' < r$. But then:

$$c_j(s') = r' + \frac{2^r}{v_{s'_j}} = r' + 2^{r-r'} \geq r' + (r - r' + 1) = r + 1 > c_j(s),$$

since $2^x \geq x + 1$ when $x \geq 1$.
Now the total number of machines is $m = \sum_{r=0}^{k} \frac{k!}{r!} = k! \sum_{r=0}^{k} \frac{1}{r!} \leq 3\Gamma(k+1)$. Then we obtain $k \geq \Gamma^{-1}(\frac{m}{3}) - 1 = \Omega\left(\frac{\log m}{\log\log m}\right)$ since Γ^{-1} is an increasing mapping. We show that the socially optimum configuration s^* has makespan $C(s^*) = 2$. Define s^* as the assignment of $\frac{k!}{r!} \times r$ jobs of processing length 2^r on machines of group G_{r-1}. Because $|G_{r-1}| = \frac{k!}{(r-1)!} = \frac{k!}{r!} \times r$, one job is assigned on each machine of G_{r-1} and the makespan is $\frac{2^r}{2^{r-1}} = 2$. Thus, $PoA \geq \frac{C(s)}{C(s^*)} = \frac{k}{2} = \Omega\left(\frac{\log m}{\log\log m}\right)$.□

Proposition 3 *The Strong Price of Anarchy of* `Makespan` *for restricted assignment machines is* $\Omega\left(\frac{\log m}{\log\log m}\right)$.

Proof. Now let us utilize the same grouping of machines for the case of restricted assignments, with identical machines; assume they all have unit speed. For $r = 1, \ldots k$ make up r jobs per machine in group G_r of unit processing length each. Call J_r the set of these jobs. These jobs may only be assigned to machines of groups G_{r-1} or G_r. Consider the assignment s where jobs J_r are assigned to group G_r, r jobs per machine. This assignment has makespan $C(s) = k$ (actually, for every machine $i \in G_r$ we have $C_i(s) = r$). We claim that s is a strong equilibrium. Consider any joint deviation of any coalition $J \subseteq \mathcal{J}$ of jobs, under s. We show two simple claims.

Claim 1 *If a job $j_1 \in J$ with $s_j \in G_r$ deviates to any machine $s'_{j_1} \in \mathcal{M} \setminus \{s_{j_1}\}$, at least one job $j_2 \neq j_1$, $j_2 \in J$ with $s_{j_2} = s'_{j_1}$ leaves machine $s_{j_2} = s'_{j_1}$ for some machine $s'_{j_2} \neq s_{j_2}$.*

This follows by the fact that j_1 may only switch to a machine within G_r, or to a machine within G_{r-1}. If no job leaves j_2 we have, in the first case, $c_{j_1}(s') \geq r + 1 > c_{j_1}(s)$ and in the second case it is $c_{j_1}(s') \geq (r-1) + 1 = r = c_{j_1}(s)$. Thus, in either case, j_1 would not participate in the joint deviation of J.

Claim 2 *If there is a deviating job $j \in J$ with $s_j \in G_r$, then there must exist at least one job $j' \in J$ with $s_{j'} \in G_r$, that deviates to some machine in G_{r-1}.*

This follows because jobs are optimally distributed to each group of machines under s, hence simply swapping machines among them will not lessen their completion time. Using this fact recursively down to group G_1, there is at least one job $j \in J$ with $s_j \in G_1$, that deviates to some machine of G_0. But j may not decrease its completion time this way because it was $c_j(s) = 1$ and since no jobs are assigned to machines of G_0, it will still be $c_j(s') = 1$. Thus s is strong. For the optimum configuration s^*, we assign jobs of J_r to machines G_{r-1}, for $r = 1, \ldots, k$. Thus, $C_i(s^*) = 1$ for every machine $i \in G_r$ when $r = 0, \ldots, k$ and then $C(s^*) = 1$. In conclusion, Thus, $PoA \geq \frac{C(s)}{C(s^*)} = k = \Omega\left(\frac{\log m}{\log\log m}\right)$. □

A trivial example however shows that `Makespan` has unbounded PoA for $R||C_{\max}$; take two machines and two jobs with $p_{11} = p_{22} = \epsilon$ and $p_{12} = p_{21} = W$. Then the configuration $(2, 1)$

is a PNE of makespan W while $(1, 2)$ is the socially optimum configuration (also a PNE). Thus $PoA \geq W/\epsilon$, which may be arbitrarily large for big values of W and arbitrarily small values of ϵ. Andelman et al. [1] first studied strong equilibria for selfish scheduling under the `Makespan` mechanism. They show that the $SPoA$ for $R||C_{\max}$ lies between m and $2m - 1$. Fiat et al. subsequently showed that $SPoA = \Theta\left(\frac{\log m}{(\log \log m)^2}\right)$ for $Q||C_{\max}$ and tightened the $SPoA$ for $R||C_{\max}$ to m.

3.2 Ordering Policies

An ordering policy is intuitively one that orders execution of jobs non-preemptively on a machine. The `Makespan` policy discussed above is a preemptive policy (hence, not an ordering one). To order the jobs on their assigned machine, an *ordering* policy may use the jobs' processing time on the machine, or even their distinct index, to resolve ties in a definitive manner. A characterization of *strongly local* deterministic ordering policies proposed in [10] is through the *Independence of Irrelevant Alternatives* (IIA) property; this roughly dictates that the relative ordering of any two jobs on a machine under a policy satisfying the IIA is independent of the presence of another job. More formally, let $J_i(s) \subseteq \mathcal{J}$ denote the subset of jobs that are assigned on machine i under configuration s. Suppose that for two distinct jobs $j_1, j_2 \in J_i(s)$, j_1 precedes j_2, i.e. $j_1 \prec_\Pi j_2$, according to an ordering $\prec_\Pi$ induced under s, by the strongly local policy Π deployed on $i \in \mathcal{M}$. Then under any configuration s' with $j_1, j_2 \in J_i(s) \cap J_i(s')$, j_1 we still have $j_1 \prec_{\Pi(s')} j_2$. Notice that under ordering policies the completion time of a job j is dependent on the ordering policy deployed on the machine that j is assigned and on the rest of the jobs assigned to this machine. In particular, if $\Pi(i)$ denotes the ordering policy deployed on machine $i \in \mathcal{M}$, we have:

$$c_j(s) = p_{j,s_j} + \sum_{j' \prec_{\Pi(s_j)} j : s_{j'} = s_j} p_{j',s_j}$$

Below we describe known results with respect to two well studied ordering policies satisfying the IIA property. Bounds on the PoA of these policies and on the randomized ordering policy discussed subsequently are summarized in table 2.

Shortest Processing Time (`SPT`) Under this mechanism jobs on all machines are executed by shortest processing time first. Ties are resolved consistently across all machines on the basis of jobs' indices (i.e. smaller or larger index first), hence jobs are totally ordered under `SPT`. This is an ordering non-preemptive policy, in contrast to `Makespan` which is preemptive. It was shown in the classical work of Graham [32] that `SPT` achieves $PoA = 2 - \frac{1}{m}$ for $P||C_{\max}$. The related *shortest-first greedy algorithm* was shown to exhibit approximation ratio of $O(\log m)$ for $Q||C_{\max}$ and $B||C_{\max}$, by Aspnes *et al.* [5] and Azar *et al.* [11] respectively. Cho and Sahni showed a lower bound of $\Omega(\log m)$ for $Q||C_{\max}$ [17]. Immorlica *et al.* used these results in [34] to prove a PoA of $\Theta(\log m)$ for `SPT` in $Q, B||C_{\max}$. In particular, they showed that the set of pure Nash equilibria under `SPT`, even for $R||C_{\max}$, coincides with the set solutions that can be returned by the shortest-first greedy algorithm. They also proved a general upper bound of $O(\log m)$ on the PoA of any deterministic ordering policy for $Q||C_{\max}$ and for $B||C_{\max}$. We provide the proof for the $Q||C_{\max}$ below in Theorem 2. For $R||C_{\max}$ a PoA of m for `SPT` emerges by the analysis of the shortest-first greedy algorithm, a result due to Ibarra and Kim [33]. An asymptotically matching lower bound was shown by Azar *et al.* in [10] for any deterministic ordering policy (we survey their results later in subsection 3.5). Interestingly, `SPT` induces a potential game, even on unrelated machines as shown in [34]. Furthermore, an iterative best response performed by the players converges in

exactly n rounds, when each job is given the chance for best response in every round, in arbitrary order.

Longest Processing Time (LPT) Under LPT jobs are scheduled on all machines by longest processing time first. Ties are resolved consistently across all machines on the basis of jobs' indices (i.e. smaller or larger index first), hence jobs are totally ordered under LPT. It is a non-preemptive ordering policy that achieves $PoA = \frac{4}{3} - \frac{1}{3m}$ on identical machines and at most $O(\log m)$ on uniformly related and restricted assignment machines. The first bound emerges by the analysis of the *longest-first greedy algorithm* by Graham [32] (upper bound) and by the analysis of Christodoulou *et al.* [19] (lower bound). The second comes from the analysis of [34]. Immorlica *et al.* show in [34] that the set of pure Nash equilibria obtained under LPT for $P, Q, B||C_{\max}$ is exactly the set of solutions that can be returned by the *longest-first greedy algorithm*. This yields $PoA \in [1.52, 1.59]$ for $Q||C_{\max}$, by the works of Dobson [21] and Friesen [26] on the longest-first greedy algorithm. Immorlica *et al.* proved a general $PoA = \Theta(\log m)$ for any deterministic ordering policy for $B||C_{\max}$. They also showed that under LPT players converge in PNE after n rounds of iterative best responses but only for the $P, Q, B||C_{\max}$ scheduling models. It is unknown whether iterative best response converges for $R||C_{\max}$.

We provide the proof of a general $O(\log m)$ upper bound for the PoA of any deterministic ordering policy (including SPT,LPT) for $P, Q||C_{\max}$. This appeared originally in [34]. Notice however, that both SPT and LPT perform much better for $P||C_{\max}$ (and LPT performs much better for $Q||C_{\max}$). The performance of SPT and LPT for $P||C_{\max}$ is given in a separate subsection below.

Theorem 2 *The Price of Anarchy of any deterministic ordering policy for uniformly related machines is* $O(\log m)$.

Proof. Assume that machines are ordered in such a way that $v_1 \geq v_2 \geq \cdots \geq v_m$. Let α be an integer so that $\alpha C(s^*) \leq C(s) \leq (\alpha+1)C(s^*)$, where s is a PNE and s^* is the socially optimum configuration. For $i \in \{1, \ldots, \alpha-1\}$, let μ_i be the minimum index of a machine so that $C_{\mu_i+1}(s) < (\alpha - i)C(s^*)$ and $C_\mu(s) \geq (\alpha - i)C(s^*)$ for all machines $\mu \leq \mu_i$. We show first that $C_1(s) \geq (\alpha-1)C(s^*)$, which means that $\mu_1 \geq 1$. Subsequently, we will prove that $\mu_i \geq (i-r-1)\mu_r$, for any $r \in \{1, \ldots, i-1\}$ and for $i \leq \alpha - 1$. This will establish that $\mu_{\alpha-1} \geq 2\mu_{\alpha-4} \geq 2^q \cdot \mu_{\alpha-3q-1} \geq 2^{\frac{\alpha-2}{3}} \cdot \mu_1$. Because $m \geq \mu_{\alpha-1}$ and $\mu_1 \geq 1$, we will obtain $\alpha = O(\log m)$.

For proving $\mu_1 \geq 1$ and $C_1(s) \geq (\alpha-1)C(s^*)$, let j denote the job scheduled last on the machine of maximum makespan under s; thus, we deduce $c_j(s) \geq \alpha C(s^*)$. Then, if $C_1(s) < (\alpha-1)C(s^*)$, setting $s'_j = 1$ yields $c_j(s_{-j}, s'_j) = C_1(s) + \frac{\ell_j}{v_1} < (\alpha-1)C(s^*) + C(s^*) < \alpha C(s^*) \leq c_j(s)$ since $C(s^*) \geq C_i(s^*) \geq \frac{\ell_j}{v_i} \geq \frac{\ell_j}{v_1}$ where i is the machine containing job j in assignment s^*, which contradicts s being a PNE. Therefore $C_1(s) \geq (\alpha-1)C(s^*)$ and $\mu_1 \geq 1$.

Now we prove that $\mu_i \geq (i-r-1)\mu_r$ for any $r \in \{1, \ldots, i-1\}$ and $i \leq \alpha - 1$. Let $J \subseteq \mathcal{J}$ denote the set of jobs that are scheduled on machines $1, 2, \ldots, \mu_r$ in s and have completion times $c_j(s) \geq (\alpha - i + 1)C(s^*)$, for all $j \in J$. Take any job $j \in J$; we claim that in s^* job j is assigned to one of the machines $1, 2, \ldots, \mu_i$. If j is assigned to a machine $\mu > \mu_i$ in s^*, then $\frac{\ell_j}{v_{\mu_i+1}} \leq \frac{\ell_j}{v_\mu} \leq C(s^*)$, which means that j has an incentive to move to $s'_j = \mu_i + 1$ under s, because:

$$c_j(s_{-j}, s'_j) = C_{\mu_i+1}(s) + \frac{\ell_j}{v_{\mu_i+1}} < (\alpha - i) + C(s^*) < (\alpha - i + 1)C(s^*) \leq c_j(s)$$

Thus all jobs in J are assigned to machines $1, 2, \ldots, \mu_i$ in s^*. The sum of processing lengths of jobs in J on any machine $q \leq \mu_r$ is at least $((\alpha-r)-(\alpha-i+1))C(s^*)v_q = (i-r-1)C(s^*)v_q$. The sum of

Ordering Mechanisms

Model	`SPT`	`LPT`	`RANDOM`
$P\|\|C_{\max}$	$2-\frac{1}{m}$ [31, 34]	$\frac{4}{3}-\frac{1}{3m}$ [32, 19]	$2-\frac{2}{1+m}$ [25, 46]
$Q\|\|C_{\max}$	$\Theta(\log m)$ [5, 34]	$[1.52, 1.59]$ [21, 26, 34]	$\Theta\left(\frac{\log m}{\log\log m}\right)$ [20]
$B\|\|C_{\max}$	$\Theta(\log m)$ [5, 34]	$\Theta\left(\frac{\log m}{\log\log m}\right)$ [11, 34]	$\Theta\left(\frac{\log m}{\log\log m}\right)$[9, 28]
$R\|\|C_{\max}$	m [10, 34]	∞	$\Theta(m)$ [34]

Table 2: Performance of Ordering Mechanisms for Multiprocessor Scheduling.

processing lengths of all jobs in J is then: $\sum_{j\in J}\ell_j \geq (i-r-1)C(s^*)\sum_{q=1}^{\mu_r} v_q$. This total processing time is assigned to machines $1, 2, \ldots, \mu_i$ in the socially optimum configuration s^*, thus: $\sum_{j\in J}\ell_j \leq C(s^*)\sum_{q=1}^{\mu_i} v_q$. The latter two inequalities yield $C(s^*)\sum_{q=1}^{\mu_i} v_q \geq (i-r-1)C(s^*)\sum_{q=1}^{\mu_r} v_q$, which becomes $\mu_i \geq (i-r-1)\mu_r$, because machines are indexed by non-increasing order of speeds. This completes the proof. □

3.3 Efficiency of `SPT` & `LPT` on Identical Machines

On the basis of a classical work due to R. L. Graham [32], we show that the PoA for `SPT` and `LPT` is $2-1/m$ and $4/3-\frac{1}{3m}$ respectively, for the case of $P||C_{\max}$.

Mechanism `SPT` Suppose w.l.o.g. that $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_n$. Let s be a PNE whereas s^* denotes the socially optimum configuration. First remark that in s, the completion time of job n coincides with the makespan, i.e. $C(s) = c_n(s)$. If it were not the case then the job whose completion time coincides with the makespan is not on machine s_n, and it can move to machine s_n and benefit. Denote by T the time at which the execution of job n starts in solution s:

$$C(s) = T + \ell_n. \tag{1}$$

Now observe that every machine is busy between times 0 and T, otherwise job n could move to another machine and complete earlier:

$$T \leq \frac{1}{m}\sum_{i=1}^{n-1}\ell_i. \tag{2}$$

As usual, we have that:

$$C(s^*) \geq \frac{1}{m}\sum_{i=1}^{n}\ell_i \text{ and } C(s^*) \geq \ell_n \tag{3}$$

Using Equations (1), (2) and (3), we get that $C(s) = T + \ell_n \leq \frac{1}{m}\sum_{i=1}^{n-1}\ell_i + \frac{\ell_n}{m} + (1-\frac{1}{m})\ell_n \leq (2-\frac{1}{m})C(s^*)$.

A tight example for every $m \geq 2$ can be the following: there are $(m-1)^2$ job of load 1, one job of load m and one job of load $m-1$. In a strategy profile s, the jobs of load m and $m-1$ are on the first machine, and every other machine hosts $m-1$ jobs of length 1. It is not difficult to see that s is a PNE with $C(s) = 2m-1$. In a strategy profile s^*, the job of load m is alone on the first machine, the job of load $m-1$ shares the second machine with one job of load 1, and every machine out of the $m-2$ remaining ones hosts m jobs of length 1. We get that $C(s^*) = m$ so $C(s)/C(s^*) = 2-1/m$.

Mechanism `LPT` We suppose w.l.o.g. that $\ell_1 \geq \ell_2 \geq \cdots \geq \ell_n$. Again, s denotes a PNE whereas s^* denotes a social optimum. Let k be the job whose completion time in s coincides with the makespan, i.e. $C(s) = c_k(s)$. By a minimality argument on the number of jobs in an instance achieving a given PoA, one can suppose w.l.o.g. that $k = n$.[1] Denote by T the time at which the execution of job n starts in solution s, equations (1), (2) and (3) still hold so we get that $C(s) \leq C(s^*) + (1 - \frac{1}{m})\ell_n$. Now suppose that $\ell_n \leq C(s^*)/3$, it would be $C(s) \leq (\frac{4}{3} - \frac{1}{3m})C(s^*)$. Now suppose that $\ell_n > C(s^*)/3$. By hypothesis $\ell_i > C(s^*)/3$ for all i. Hence at most two jobs share the same machine in s^*. In that case s must be optimal, i.e. up to a renaming of the machine indexes, s is such that job j is on machine j when $1 \leq j \leq m$, and job j is on machine $2m + 1 - j$ when $m + 1 \leq j \leq 2m$.

A tight example for every $m \geq 2$ can be the following: there are three jobs of load m and for every $\ell \in [m + 1..2m - 1]$ there are two jobs of load ℓ. The $2m + 1$ jobs are such that $\ell_1 \geq \ell_2 \geq \cdots \geq \ell_{2m+1}$. In a strategy profile s, the m largest jobs choose distinct machines, i.e. job j is on machine j for $1 \leq j \leq m$. For $m+1 \leq j \leq 2m$, job j is on machine $2m+1-j$. Job $2m+1$ is on machine 1. It is not difficult to see that s is a PNE with $C(s) = 4m - 1$. In a strategy profile s^*, the three jobs of load m are on the first machine, the second and third machine both host one job of load $2m - 1$ and one job of load $m + 1$, the fourth and fifth machine both host one job of load $2m - 2$ and one job of load $m + 2$, and so on. We get that $C(s^*) = 3m$ since the total load on every machine is $3m$. Hence $C(s)/C(s^*) = \frac{4}{3} - \frac{1}{3m}$.

3.4 `RANDOM`: Randomized Ordering

Let us close this section by mentioning a randomized non-preemptive (ordering) mechanism. Under the `RANDOM` mechanism each machine executes the jobs by order chosen uniformly at random (random permutation). `RANDOM` simulates in some sense `Makespan` in a randomized manner and yields a similar expression to the individual completion time of jobs. It can be verified that the completion time of a job under `RANDOM` is:

$$c_j(s) = p_{j,s_j} + \frac{1}{2} \sum_{j' \neq j : s_{j'} = s_j} p_{j',s_j}$$

Because of the technical similarity of the players' cost under `RANDOM` with that under `Makespan`, the PoA bounds of $2 - \frac{2}{1+m}$ for $P||C_{\max}$ and $O\left(\frac{\log m}{\log \log m}\right)$ for $Q, B||C_{\max}$ under `Makespan` hold also for the case of `RANDOM`. Immorlica *et al.* showed $PoA \leq 2m - 1$ and $PoA \geq m - 1$ for `RANDOM` in $R||C_{\max}$ [34]. It is not known whether `RANDOM` does induce games with PNE on environments other than identical machines ($P||C_{\max}$). For uniformly related machines Dürr and Nguyen Kim showed that it induces PNE if the speeds of the machines do not differ much [22]. They also proved that a potential game is induced by the policy on 2 unrelated machines. Perhaps the most interesting feature of `RANDOM` is that it is the only known non-preemptive *non-clairvoyant* policy; it does not need to know the processing lengths of the jobs and it is an ordering policy. However, it cannot handle anonymous jobs (since it needs to order them). Completing the study of its performance remains an important open question.

3.5 Unrelated Machines Scheduling

Scheduling unrelated machines has received considerable attention as the most general machines environment. Under the framework of coordination mechanisms it has also attracted interest from

[1] If $k < n$ then remove every job of index $i > k$. It is not difficult to see that s remains a Nash equilibrium with the same makespan. At the same time, $C(s^*)$ cannot increase.

the perspective of the performance of the iterative best response procedure; this is a simple practical local search procedure compared to the polynomial-time 2-approximation method that was developed in the seminal paper of Lenstra *et al.* [37]. The most recent developments in this context are due to Azar *et al.* [10] and Caragiannis [14].

Azar *et al.* [10] study coordination mechanisms consisting of local policies. For any set of strongly local ordering policies deployed upon a set of m unrelated machines, they prove that the PoA of the mechanism is at least $\frac{m}{2}$. This matches asymptotically the upper bound of m shown by Ibarra and Kim [33] for the performance of `SPT` scheduling; it is known that the output of `SPT` scheduling corresponds to PNE of the `SPT` mechanism. In an effort to improve over the performance of strongly local ordering policies, Azar *et al.* study a simply local ordering policy that orders the jobs on each machine according to their *inefficiency* ratio. The inefficiency of a job $j \in \mathcal{J}$ on machine $i \in \mathcal{M}$ is defined as $e_{ji} = \frac{p_{ji}}{p_j}$, where $p_j = \min_i p_{ji}$ is the smallest processing time of j over all machines. The inefficiency-based policy schedules jobs on a machine in order of non-decreasing inefficiency e_{ji}. Clearly this is not a strongly local policy, because it uses information about the jobs assigned to a machine, that is relevant to all other machines. In [10] it is shown that the inefficiency based policy achieves a PoA of $O(\log m)$; its drawback is that it does not always induce PNE and, even when it does, an iterative best response procedure may not converge to any of them. The upper bound on the performance of this policy is shown to be tight for the whole class of simply local ordering policies. The authors then propose a *preemptive* mechanism that has slightly worse $PoA = O(\log^2 m)$, but guarantees fast convergence of iterative best response to PNE in at most $O(n)$ rounds for all players, i.e. at most $O(n)$ best responses per player.

Caragiannis [14] continued the study of coordination mechanisms in the unrelated machines model, to uncover the potential of *preemptive* scheduling policies. He designed three mechanisms with different properties. They are all deterministic, preemptive and simply local. The first one induces a potential game with $PoA = O(\log m)$, in which players converge to PNE in at most n rounds of iterative best response. The second breaks through the $\Omega(\log m)$ lower bound of Azar *et al.* for simply local non-preemptive mechanisms, by exhibiting $PoA = O\left(\frac{\log m}{\log \log m}\right)$; however, iterative best response may not converge under this mechanism. The third proposed mechanism induces a potential game with $PoS = O(\log m)$, $PoA = O(\log^2 m)$ and players can converge to PNE within $O(n \log n)$ rounds of iterative best response. The latter mechanism is the first one with the properties of inducing a potential game with bounded PoA, that can also handle *anonymous* jobs, i.e. it does not need to know a specific index for each job.

4 Selfish scheduling and truthfulness

In this section we suppose that a job's load is only known to his owner. In this case every player must communicate this information. For feasibility reasons, every machine should allocate a time window to every job it has to execute and this slot must be at least as large as the processing time of the job that it accommodates. However the computation of these slots are based on the declaration of the agents. This is an opportunity for some agents to report a false load and decrease their individual cost. Then the agents are definitely not trustworthy. With false information, unreasonable solutions can emerge if the system is not designed to cope with manipulation. Therefore the problem is at the intersection of scheduling theory and *mechanism design*, a field recently explored by Nisan and Ronen in [42], in which the social cost is minimized while misreporting a private value should not be profitable. A mechanism is called *truthful* (or *strategy-proof*) if misreporting a private value is not profitable to a single player.

Nisan and Ronen's seminal article follows the well-established formalism of mechanism design in

which an output function and a payment vector is computed on the basis of the agents' declarations. Nevertheless many models were investigated in the literature (see for example [3, 2, 42, 6, 19, 18, 39] for a non exhaustive list of results), depending on several parameters: the players are either the jobs or the machines, payments are allowed or not, the strategy of an agents is vector or a singleton, pre-emption is allowed or not, etc. Here we present some results which apply for a model close to the one mainly discussed in this chapter.

4.1 The model

We consider the model introduced in Section 2 and we enrich it as follows. Every player's action, in addition to a machine number, contains a real (denoted by b_i for player i) which represents the declared length of his job. The machine environment is $P||C_{max}$ (i.e. the machines are identical and the social cost coincides with the makespan) and the mechanism must be deterministic and non-preemptive. Every machine must compute a feasible schedule, i.e. a time window for every job that chooses to be executed on it. Then a job j starts being executed when its corresponding time window is opened and it completes ℓ_j time units after. A player would not be satisfied if his job were not fully executed so we assume that he always reports a value $b_i \geq \ell_i$ and the machine allocates him a time slot of at least b_i time units. We say that a mechanism (a set of m algorithms, one per machine, which allocates time windows to the players, on the basis of their declarations) is *truthful* (without money) if a player cannot unilaterally report a false length and decrease his completion time. The next two subsections are based on [19, 18].

4.2 `LPT` is not truthful

Suppose that the mechanism is `LPT`, i.e. every machine proceeds as follows for the subset of jobs that it has to execute: it takes the player who declared the largest length (ties are broken indexwise) and allocates him a time window of size equal to the bidded length; next the player who declared the second largest length is allocated a time window of size equal to the bidded length, and so on. No idle time is inserted between two consecutive time slots. As we have seen before, the PoA is $\frac{4}{3} - \frac{1}{3m}$ if the players report their true lengths.

Consider an instance with two machines and three jobs 1, 2 and 3 with loads 1, 2 and 3 respectively. The state where job 3 is alone on the first machine while the others are on the second machine is optimal but it is not a Nash equilibrium. Indeed job 1 completes at time 3 (it is executed just after job 2) but if it declares $2 + \epsilon$ (with $1 > \epsilon > 0$) then it ends at time $2 + \epsilon$ because the mechanism executes it before job 2. Suppose that job 1 does so, the second job completes at time $4 + \epsilon$. Then player 2 can move to machine 1 and declare $3 + \epsilon$ and complete at time $3 + \epsilon$, etc. Clearly `LPT` is not truthful.

4.3 `SPT` and a matching lower bound

Suppose that the mechanism is `SPT`, i.e. every machine proceeds as follows for the subset of jobs that it has to execute: it takes the player who declared the smallest length (ties are broken indexwise) and allocates him a time window of size equal to the bidded length; next the player who declared the second smallest length is allocated a time window of size equal to the bidded length, and so on. No idle time is inserted between two consecutive time slots. As we have seen before, the PoA is $2 - \frac{1}{m}$ if the players report their true lengths.

It is not difficult to see that `SPT` is truthful. If player i declares a length b_i satisfying $b_i > \ell_i$ then the time window that the mechanism allocates him can only begin after the one the player

gets when he reports his true length. As mentioned above, no player can rationally report a value below his true length.

Now suppose that there is a deterministic and non preemptive mechanism, say `MEC`, which is truthful and its PoA is strictly less than $2 - \frac{1}{m}$. Consider a first instance $\mathcal{I}_1$ made of $m(m-1)+1$ jobs of length 1. In this case the optimal makespan is m. Since `MEC` is deterministic and non preemptive, there must be a job of $\mathcal{I}_1$, say j', which starts to be executed at time $t \geq m-1$. Consider a second instance $\mathcal{I}_2$ made of $m(m-1)$ jobs of length 1 plus one job of length m. For this second instance the optimal makespan is still m (the large job is alone on one machine). Since `MEC` has PoA strictly less than $2 - \frac{1}{m}$, any job of $\mathcal{I}_2$ completes before time $(2 - \frac{1}{m})m = 2m - 1$. In particular `MEC` gives to the job of length m a time slot which starts before time $2m-1-m = m-1$. Combining instance $\mathcal{I}_1$ and $\mathcal{I}_2$ we reach a contradiction: job j' in $\mathcal{I}_1$ starts at time $t \geq m-1$ but if it declares m instead of 1 then `MEC` allocates him a time window which starts strictly before $m-1$, as it is shown for instance $\mathcal{I}_2$. `MEC` cannot be truthful. We deduce that `SPT` is optimal among truthful mechanisms.

5 Setup Times: A Case Study

Let us exemplify the study of coordination mechanisms for a slightly more involved class of multi-processor job scheduling games; we will introduce setup times, where each job is of a certain *type*. On any machine, jobs of a given type may be executed *only* after a type-dependent preprocessing (performed once for all same-type jobs) called *setup*. Each machine schedules its assigned jobs according to a *scheduling policy* (algorithm). Given the deployed scheduling policies, players assign their jobs selfishly, to minimize their individual completion times. We examine the impact of selfish behavior on the *overall* (social) cost of *stable* assignments (PNE and SE) and how this can be alleviated, by deployment of appropriate scheduling policies on the machines.

This study is motivated by concerns in performance optimization of large-scale distributed systems (computational grids, P2P file sharing systems etc.). Setup overheads in these systems may well dominate the net processing load of tasks assigned by users; consider e.g. loading application environments, booting operating systems, establishing QoS for network connections. Scheduling with setup times also provides for modeling another very natural situation; the case where many autonomous users may benefit from the output of an identical process. In this case users may only care for the output of the setup (which takes non-negligible time) but their jobs may have virtually zero load for the machine. As an example, consider the setup corresponding to computation and writing of output to a remote file which users simply read. Then the machine may also have to deal *obliviously* of users' presence; only one of them may actually declare his presence by requesting execution of the setup, whereas the rest simply benefit from the request.

Preemptive multi-processor scheduling *with setup times* [8] requires a minimum makespan preemptive schedule, such that the setup is executed by a machine between execution of two job portions of different type. The problem is known to be **NP**-hard, see e.g. [29] [SS6]. The best known approximation algorithm has a performance guarantee of $\frac{4}{3}$ [47] (see [15] for a previous $\frac{3}{2}$ factor). For equal setup times a PTAS is given in [47] and an FPTAS for 2 machines in [48]. See [16] for a slightly different version. In our context we will only maintain the concept of setup times, to investigate the congestion effects that it introduces in a game-theoretic setting. We will study a straightforward adaptation of the `Makespan` mechanism and then introduce a class of ordering mechanisms and characterize their performance.

Definitions. We mainly consider an environment of *identical* machines. Each job $j \in \mathcal{J}$ is

associated with a *type* $t_j \in \mathcal{U}$, where $\mathcal{U}$ is the universe of all possible types [2]. The subset of $\mathcal{U}$ corresponding to the set of jobs $\mathcal{J}$ is denoted by $\mathcal{T} = \{t_j | j \in \mathcal{J}\}$ and define $k = |\mathcal{T}|$. We refer to any specific type by θ. Each job $j \in \mathcal{J}$ and each type $\theta \in \mathcal{U}$ are respectively associated to processing length $\ell_j \geq 0$ and setup time $w(\theta) \geq 0$. If $w(\theta) = 0$, then $\ell_j > 0$ for all j with $t_j = \theta$. Otherwise, we allow $\ell_j = 0$. In this model, the makespan of a socially optimum assignment s^* can be lower bounded as:

$$\begin{array}{lrcll} \textbf{(a)} & mC(s^*) & \geq & \displaystyle\sum_{\theta \in \mathcal{T}} w(\theta) + \sum_{j \in \mathcal{J}} \ell_j & \\ \textbf{(b)} & C(s^*) & \geq & w(t_j) + \ell_j & \text{for any } j \in \mathcal{J} \\ \textbf{(c)} & (k-1)C(s^*) & \geq & \displaystyle\sum_{\xi \in \mathcal{T} \setminus \{\theta\}} w(\xi) & \text{for any } \theta \in \mathcal{T} \end{array} \tag{4}$$

The only restriction that we impose on the scheduling policies is that the setup of any type θ on any machine i is executed before execution of type θ jobs on i.

5.1 On the `Makespan` Mechanism

We study an adaptation of the preemptive `Makespan` mechanism introduced in [36] and discussed in subsection 3.1; under `Makespan` completion time of j equals completion time of the machine that j is assigned to. The proof of the following result is a generalization of the proof of [1]. Their proof depends crucially on all jobs having non-zero processing length (see Lemma $A.1$ in the full version [1]). To overcome this, we used a more detailed vector potential function.

Theorem 3 *Strong Equilibria exist in the scheduling game with setup times, under the* `Makespan` *mechanism.*

Proof. We suppose that $\mathcal{J} = \{1, \ldots, n\}$. To every state s we associate a permutation π_s of the set of players such that $\pi_s(j) \leq \pi_s(j')$ if and only if $c_j(s) \geq c_{j'}(s)$. Obviously, such a permutation always exists (for instance we sort the jobs by non increasing completion time). In particular, $\pi_s^{-1}(1)$ is a job whose completion time is the makespan under s, i.e., $C(s) = c_{\pi_s^{-1}(1)}(s)$. Associate the vector $v_s = \langle c_{\pi_s^{-1}(1)}(s), c_{\pi_s^{-1}(2)}(s), \cdots, c_{\pi_s^{-1}(n)}(s) \rangle$ to every state s. In the following, v_s^i denotes the i-th coordinate of v_s. Given two assignments s and r, we say that v_r is *lexicographically smaller* than v_s iff $v_r^1 < v_s^1$ or there is an index $i^* > 1$ such that $v_r^i = v_s^i$ for $i \in \{1, \cdots, i^* - 1\}$ and $v_r^{i^*} < v_s^{i^*}$. We show by contradiction that, if s is the lexicographically smallest assignment, then it is a strong equilibrium. Let r be an assignment and $J \subseteq \mathcal{J}$ be a nonempty coalition such that $\forall j \in J\ s_j \neq r_j$ and $\forall j \notin J\ s_j = r_j$. Moreover, by construction of a coalition, we must get: $\forall j \in J\ c_j(s) > c_j(r)$. Let $c_{max} = \max_{j \in J} c_j(s)$, that is c_{max} is the maximum completion time of the jobs of the coalition J under state s. We prove the following properties:

(i) For any $j \in \mathcal{J}$ such that $c_j(s) > c_{max}$, $c_j(r) = c_j(s)$.

(ii) For any $j \in J$, $c_j(r) < c_{max}$.

(iii) If $j \in \mathcal{J} \setminus J$ and $c_j(s) = c_{max}$, then $c_j(r) \leq c_{max}$.

(iv) For $j \in \mathcal{J} \setminus J$ with $c_j(s) < c_{max}$, $c_j(r) < c_{max}$.

[2] E.g. the set of application environments installed on each machine.

For (i). Let j be a player such that $c_j(s) > c_{max}$. Since $c_{max} \geq c_p(s)$ for all $p \in J$, we deduce that $j \in \mathcal{J} \setminus J$ and $s_j = r_j$. If at least one job of J, say p, moves to machine s_j then $c_p(s) \leq c_{max} < c_j(s) \leq c_p(r)$, contradiction with $c_p(r) < c_p(s)$. Then no job quit or moves to s_j, the completion time on that machine remains unchanged.

For (ii). If $j \in J$ then $c_j(r) < c_j(s)$ and $c_j(s) \leq c_{max}$.
For (iii). Since $j \in \mathcal{J} \setminus J$, it is $s_j = r_j$. If no job of J moves to machine s_j then $c_j(r) \leq c_j(s) = c_{max}$ (the completion of s_j cannot increase). If at least one job of J, say p, moves to s_j then $c_j(r) = c_p(r)$ and $c_p(r) < c_{max}$ by item (ii).

For (iv). The proof is quite similar to item (iii). By definition we have $s_j = r_j$, i.e. j stays on machine s_j. If no job of J moves to s_j, then $c_j(r) \leq c_j(s)$ (the completion of machine s_j cannot increase). If at least one job of J, say p, moves to s_j then $c_j(r) = c_p(r)$ and $c_p(r) < c_{max}$ by item (ii).

Items (i) to (iv) lead to $|\{i : c_i(r) = c_{max}\}| \leq |\{i : c_i(s) = c_{max}\}|$. Moreover, using item (ii), we deduce that $|\{i : c_i(r) = c_{max}\}| \neq |\{i : c_i(s) = c_{max}\}|$ since by construction there exists at least one job $j \in J$ with $c_j(s) = c_{max}$ (and $c_j(r) < c_{max}$ by item (ii)). Thus globally, items (i) to (iv) imply that v_r is lexicographically smaller than v_s (contradiction with the minimality of v_s) because $v_r^i = v_s^i$ when $v_r^i > c_{max}$ and $|\{i : v_r^i = c_{max}\}| < |\{i : v_s^i = c_{max}\}|$. □

Theorem 4 *The PoA of the* `Makespan` *mechanism for the scheduling game with setup times is m when $m \leq k$, at most $k + 1 - k/m$ when $m > k$ and at least $\frac{k+1}{1+\epsilon}$ for $m \geq 3k - 2$ and $\epsilon = \frac{2k-1}{m-k+1}$.*

Proof. **Case** $m \leq k$: The most expensive PNE s has makespan $C(s) \leq \sum_\theta w(\theta) + \sum_j \ell_j$ (all jobs on one machine). By (4a), it is $PoA \leq m$. For the lower bound take $k = m$ types of jobs, each type θ having $w(\theta) = 1$ and containing m jobs of zero processing length. A job of each type is assigned to each machine in s . Then $C(s) = m$ and s is clearly a PNE. In the social optimum s^* each type is assigned to a dedicated machine, thus $C(s^*) = 1$.

Case $m > k$: Assume that for the most expensive PNE s it is $C(s) = C_1(s)$. Let x be a job of type $t_x = \theta$ executed on machine 1. x cannot decrease its cost $c_x(s) = C_1(s)$ by switching to any machine $i \neq 1$. Then $C_1(s) \leq C_i(s) + \ell_x$ if $\theta \in T_i(s)$ or $C_1(s) \leq C_i(s) + w(\theta) + \ell_x$ otherwise. We sum up the inequalities over all machines, assuming that θ does not appear on α machines, and add $C_1(s)$ to both sides to obtain $mC_1(s) \leq \sum_{i=1}^m C_i(s) + \alpha w(\theta) + (m-1)\ell_x \leq m \sum_{\xi \in \mathcal{T}} w(\xi) + (m-1)\ell_x + \sum_{j \in \mathcal{J}} \ell_j$. Divide by m and rewrite it as:

$$C_1(s) \leq \frac{m-1}{m}\Big(\sum_{\xi \in \mathcal{T} \setminus \{\theta\}} w(\xi) + w(\theta) + \ell_x\Big) + \frac{1}{m}\Big(\sum_{\xi \in \mathcal{T}} w(\xi) + \sum_{j \in \mathcal{J}} \ell_j\Big)$$

Using (4a,b,c), $C(s) \leq \frac{m-1}{m}\left((k-1)C(s^*) + C(s^*)\right) + C(s^*) = (k + 1 - \frac{k}{m})C(s^*)$.

For the lower bound take k types, $m \geq 3k-2$, and let $w(1) = 0$ and $w(\theta) = 1$ for $\theta \in \{2, \ldots, k\}$. There are $k + 1$ jobs of type 1 and length 1 and $\frac{m-1}{\epsilon}$ jobs of type 1 and length $\epsilon = \frac{2k-1}{m-k+1}$. Types $\theta \in \{2, \ldots, k\}$ have $m - 1$ jobs each, of processing length 0. A PNE s is as follows. $k + 1$ jobs of type 1 and length 1 are assigned to machine 1. One job from each type $\theta \geq 2$ is assigned to each machine $i = 2, \ldots, m$. $\frac{1}{\epsilon}$ jobs of type 1 and length ϵ are also assigned to each machine $i \geq 2$. Thus $C_i(S) = k$ for $i \geq 2$ and $C_1(s) = k + 1$. No job may decrease its completion time (equal to the makespan of the machine it is assigned to) by switching machine. In the optimum assignment s^* assign two jobs of type 1 - with lengths 1 and ϵ - to each machine $i = 1 \ldots k + 1$. Every machine $i = k + 2 \ldots 2k$, has $m - 1$ jobs of type $i - k$, each of length 0. Every machine $i = 2k + 1 \ldots m$, has $1/\epsilon + 1$ jobs of type 1, of length ϵ. The makespan of s^* is $1 + \epsilon$.See Figure 3 for an illustration of the example. □

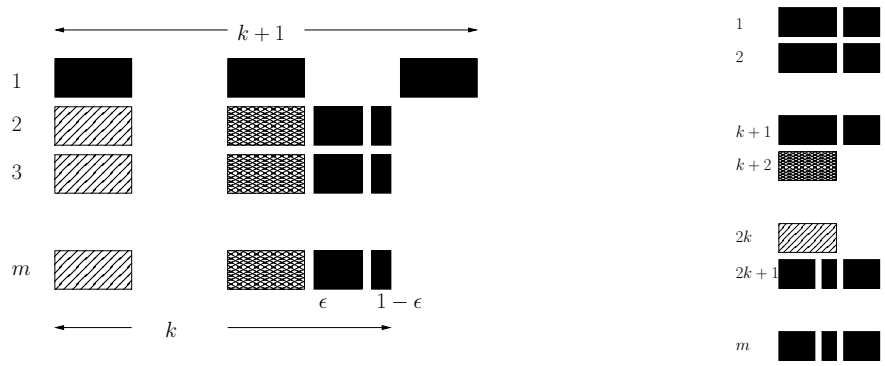


Figure 3: A lower bound of the Price of Anarchy for `Makespan`, on identical machines.

Theorem 5 *The Price of Anarchy of strong equilibria under* `Makespan` *for the scheduling game with setup times is* 2 *for* $m \geq 3$*, and* $\frac{3}{2}$ *for* $m = 2$ *machines.*

Proof. We give the proof for the case $m \geq 3$. The reader is referred to [30] for the case $m = 2$. Let s be a SE, s^* the socially optimum assignment, and $C(s) = C_1(s)$. If $C_1(s) \leq C(s^*)$ we get $SPoA = 1$. If $C_1(s) > C(s^*)$, there is machine $i \neq 1$ with $C_i(s) \leq C(s^*)$, because otherwise s would not be a SE; all jobs would reduce their completion time by switching from s to s^*. For any job x with $s_x = 1$, it is $c_x(s) \leq c_x(s_{-x}, i)$. Thus $C_1(s) = c_x(s) \leq C_i(s) + w(t_x) + \ell_x$. Thus $C(s) = c_x(s) \leq 2C(s^*)$, because $C_i(s) \leq C(s^*)$ and (4b). For the lower bound, take 3 machines and 4 jobs, with $t_1 = t_2 = \theta_1$ and $t_3 = t_4 = \theta_2$. Set $w(\theta_1) = \epsilon$, $\ell_1 = \ell_2 = 1$ and $w(\theta_2) = 1$, $\ell_3 = \ell_4 = \epsilon$. An assignment where jobs $1, 2$ play machine 1 and jobs $3, 4$ play machines $2, 3$ respectively is a strong equilibrium of makespan $2 + \epsilon$. In the social optimum jobs $3, 4$ are assigned to the same machine and 1 and 2 on dedicated machines; the makespan becomes then $1 + 2\epsilon$. Thus $SPoA \geq \frac{2+\epsilon}{1+2\epsilon} \to 2$, as $\epsilon \to 0$. □

6 Type Ordering Mechanisms

We describe a class of (deterministic) *type ordering* mechanisms, for *batch scheduling* of same-type jobs. Each machine i groups together jobs of the same type θ, into a *batch* of type θ. A type batch is executed by the machine as a whole; the setup is executed first, followed by *preemptive* execution of all jobs in the batch, in a `Makespan` fashion. Jobs within the same batch have equal completion times and are scheduled preemptively in parallel. Type batches are executed serially by each machine.

Policies in type ordering mechanisms satisfy a version of the property of *Independence of Irrelevant Alternatives* (IIA) [10]. Under the IIA property, for any set of jobs $J_i \subseteq \mathcal{J}$ assigned to machine $i \in \mathcal{M}$ and for any pair of types $\theta, \theta' \in \mathcal{U}$ with jobs in J_i if the θ-type batch has smaller completion time than the θ'-type batch, then the θ batch has a smaller completion time than the θ' batch in any set $J_i \cup \{j\}$, $j \in \mathcal{J} \setminus J_i$. Presence of j does not affect the relative order of execution of θ and θ' batches. The IIA property was used in [10] for proving a lower bound on the PoA of a class of job ordering mechanisms in the context of unrelated machines scheduling. Type ordering policies do not introduce delays in the execution of batches, but only decide their relative order of their execution, based on a batch's type index and setup time. They do not use the number of jobs within each batch; otherwise the IIA property may not be satisfied. Job lengths are used only for `Makespan`-wise scheduling within batches. Hence type ordering mechanisms function obliviously of "hidden" players with zero job lengths.

6.1 Pure Nash Equilibria

We prove next existence of PNE for any number of machines, and SE for $m = 2$ under type ordering mechanisms. An algorithm for finding PNE follows. Let $o(i)$ be the ordering of types on machine i, and $O = \{o(i)|i \in \mathcal{M}\}$ be the set of all orderings of the mechanism. By $\prec_o$ denote the precedence relation of types, prescribed by $o \in O$. Let M_o be the set of machines that schedule according to $o \in O$. Initialize $o \in O$ arbitrarily, and **repeat** until all jobs are assigned:

1. Find the earliest type θ according to $\prec_o$, with at least one unassigned job.
2. Let j be the largest length unassigned job with $t_j = \theta$.
3. Pick $i \in \mathcal{M}$ minimizing completion time of j [3] (break ties in favor of $i \in M_o$).
4. **If** $i \in M_o$ set $s_j = i$ **else** switch ordering o to $o(i)$.

Theorem 6 *The scheduling game with setup times has pure Nash equilibria, under type ordering mechanisms.*

Proof. The algorithm terminates in polynomial time; once a job is assigned, it is never considered again and within every $O(m + n)$ iterations some job is always assigned. For any type θ, denote by $\hat{s}_\theta$ the partial assignment up to the time after the last job of type θ has been assigned. We show by contradiction that no job j has incentive to deviate under an assignment s returned by the algorithm.

Assume that j does have incentive to deviate from s_j, and let s' be the resulting assignment after deviation of j. At the time corresponding to the partial assignment $\hat{s}_{t_j}$, there is no type $\theta \neq t_j$ and machine i such that $\theta \in T_i(\hat{s}_{t_j})$ and $t_j \prec_{o(i)} \theta$. If it was the case, the first job of type $\theta \neq t_j$ assigned to i would have been chosen before jobs of type t_j were exhausted, which contradicts step 1. of the algorithm. Thus, batches of type t_j are scheduled - under $\hat{s}_{t_j}$ - last on all machines with $t_j \in T_i(\hat{s}_{t_j})$. Furthermore, if j wishes to deviate to a machine $i \neq s_j$, then $c_j(s) = c_j(\hat{s}_{t_j}) > C_i(\hat{s}_{t_j}) + \ell_j = c_j(s')$, if $t_j \in T_i(\hat{s}_{t_j})$, and $c_j(s) = c_j(\hat{s}_{t_j}) > C_i(\hat{s}_{t_j}) + w(t_j) + \ell_j = c_j(s')$, if $t_j \notin T_i(\hat{s}_{t_j})$. Let j' be the last job of type t_j assigned to machine s_j (it may be $j' = j$). Because $\ell_{j'} \leq \ell_j$, it is also $c_{j'}(\hat{s}_{t_j}) = c_j(\hat{s}_{t_j}) > C_i(\hat{s}_{t_j}) + \ell_{j'}$ or $c_{j'}(\hat{s}_{t_j}) = c_j(\hat{s}_{t_j}) > C_i(\hat{s}_{t_j}) + w(t_{j'}) + \ell_{j'}$ accordingly. By the time j' was assigned, the completion time of i was at most $C_i(\hat{s}_{t_j})$. This contradicts step 3. of the algorithm with respect to j'. □

The following result identifies performance limitations of type ordering mechanisms, due to lack of a priori knowledge of $\mathcal{T} \subseteq \mathcal{U}$.

Theorem 7 *The Price of Anarchy of the scheduling game with setup times is $\frac{m+1}{2}$ for every deterministic type ordering mechanism.*

Proof. For any deterministic type ordering mechanism, assume there is a subset $\mathcal{T} \subseteq \mathcal{U}$ of $k = 2m - 1$ types, say $\mathcal{T} = \{1, \cdots, 2m - 1\}$, such that: all types of $\mathcal{T}$ are scheduled in order of ascending index in a machines and in order of descending index in $d = m - a$ machines. Then, there is a family of instances with $PoA \geq \frac{m+1}{2}$. Next we prove existence of $\mathcal{T}$. Set $w(\theta) = 1$ for all $\theta \in \mathcal{U}$. When $a = m$ or $d = m$, take an instance of m zero length jobs for each type $\theta \in \{1, \cdots, m\}$. Placing one job of each type on every machine yields a PNE with makespan m. An assignment of makespan 1 has all same-type jobs assigned to a dedicated machine, thus $PoA \geq m$. When $a \geq 1$ and $d \geq 1$, the instance has:

[3] j incurs processing load $w(t_j) + \ell_j$ if a t_j-type job is not already assigned to i.

- a jobs of zero length for each type $\theta \in \{1, \cdots, m-1\}$
- d jobs of zero length for each type $\theta \in \{m+1, \cdots, 2m-1\}$
- $m-1$ jobs of of zero length and type m
- one job of length 1 and type m
- no jobs for $\theta \in \mathcal{U} \setminus \mathcal{T}$

Assign one job of type $\theta \in \{1, \cdots, m-1\}$ on each of the a *ascending* type index machines, and one job of type $\theta \in \{m+1, \cdots, 2m-1\}$ on each of the d *descending* type index machines. Put one job of type m on every machine. This is a PNE of makespan $m+1$. Placing all jobs of type $\theta \in \{i, 2m-i\}$ on machine i yields makespan 2. Thus it is $PoA \geq \frac{m+1}{2}$.

We show existence of $\mathcal{T}$ for sufficiently large universe $\mathcal{U}$. We use the fact that any sequence of n different real numbers has a monotone (not necessarily contiguous) subsequence of $\sqrt{n}$ terms (a corollary of Theorem 4.4, page 39 in [35]). By renaming types in $\mathcal{U}$ we can assume w.l.o.g. that $\mathcal{U}$ is ordered monotonically (index-wise) on machine 1, and set $T_1 = \mathcal{U}$. Then, there is $T_2 \subseteq T_1$ such that $|T_2| \geq \sqrt{|T_1|}$ and all the types of T_2 are ordered monotonically according to index, on machines 1 and 2. After $m-1$ applications of the corollary, we obtain a set $T_m \subseteq T_{m-1} \subseteq \cdots \subseteq T_1 = \mathcal{U}$ with $|T_m| \geq |\mathcal{U}|^{2^{1-m}}$ and all its types are scheduled monotonically to their index on every machine. We set $\mathcal{T} = T_m$, and take a universe $\mathcal{U}$ of types with $|\mathcal{U}| = (2m-1)^{2^{m-1}}$, to ensure existence of $\mathcal{T}$ with $k = |\mathcal{T}| = 2m-1$ types. □

Let us note that "longest" or "shortest batch first" policies are no more powerful than type ordering mechanisms; they reduce to them for zero length jobs.

6.2 An Optimal Type Ordering Mechanism

We analyze the PoA of a type ordering mechanism termed `AD` (short for *Ascending-Descending*), that schedules type batches by ascending type index on half of the machines, and by descending type index on the rest. If m is odd one of the policies in applied to one machine more. First we prove the following lemma.

Lemma 1 *Let $\mathcal{T}' \subseteq \mathcal{T}$ include types with non-zero setup times. If two jobs of the same type in $\mathcal{T}'$ play an ascending and a descending index machine respectively under the* `AD` *mechanism, their type batches are scheduled last on the respective machines.*

Proof. We show the result by contradiction. Let jobs x_1, x_2 with $t_{x_1} = t_{x_2} = \theta$ be assigned on the ascending and descending machines 1, 2 respectively. Assume that a job y, $t_y = \theta' \neq \theta$, is scheduled on 1 after type θ. Because s is a PNE, job x_2 does not decrease its completion time if it moves to machine 1; because y is scheduled after x_1 on 1:

$$c_{x_1}(s) \geq c_{x_2}(s) - \ell_{x_2}, \text{ and } c_y(s) \geq c_{x_1}(s) + w(\theta') + \ell_y \tag{5}$$

If y switches to processor M_2 then it will be scheduled before type θ, thus its completion time will be at most $c_{x_2}(s) - w(\theta) - \ell_{x_2} + w(\theta') + \ell_y$ if $\theta' \notin T_2(s)$ (and at most $c_{x_2}(s) - w(\theta) - \ell_{x_2} + \ell_y$ otherwise). In the worst case, we obtain:

$$c_y(s) \leq c_{x_2}(s) - w(\theta) - \ell_{x_2} + w(\theta') + \ell_y \tag{6}$$

By (5) and (6), $c_y(s) \leq c_y(s) - w(\theta) < c_y(s)$, a contradiction, because $\theta \in \mathcal{T}'$. □

The next result identifies upper bounds on the PoA of AD. A proposition that follows proves tightness, through lower bounds on the Price of Stability, the ratio of the *least* expensive PNE makespan over the optimum makespan. We take $k \geq 2$; AD is identical to Makespan for $k = 1$.

Theorem 8 *The Price of Anarchy of the* AD *mechanism for the scheduling game with setup times is at most* $\frac{m+1}{2}$ *when* $m \leq k$ *and at most* $\frac{k+3}{2} - \rho$ *(*$\rho = \frac{k}{m}$ *when* m *is even and* $\rho = \frac{k-1}{m-1}$ *otherwise), when* $m > k$.

Proof. Let s be the most expensive PNE assignment and $C(s) = C_1(s) = \max_i C_i(s)$. Let θ_0 be the type scheduled last on machine 1 and x a job with $t_x = \theta_0$. Define $\mathcal{T}'_C \subseteq \mathcal{T}'$ to be types with jobs assigned to both ascending and descending machines under s. Let $T'_A \subseteq T' \setminus T'_C$ and $T'_D \subseteq T' \setminus T'_C$ contain types exclusively assigned to ascending and descending machines respectively. Notice that at most one type $\theta_1 \in \mathcal{T}'_C$ may appear in at least $\frac{m}{2} + 1$ machines (when m even) and $\frac{m+1}{2}$ machines (when m odd); thus any type in $\mathcal{T}'_C \setminus \{\theta_1\}$ appears on at most $\frac{m}{2}$ machines (actually, $\frac{m-1}{2}$ machines when m is odd). We study two cases depending on whether θ_1 exists and whether it coincides with θ_0 or not.

CASE 1: $\theta_0 = \theta_1$ or θ_1 does not exist. Job x will not decrease its completion time by moving to machine p for $p = 2, \ldots, m$. If $M_{\theta_0}(s)$ are the indices of machines which contain type θ_0, then:

$$\forall p \in M_{\theta_0}(s),\ c_x(s) \leq C_p(s) + \ell_x \text{ and } \forall p \notin M_{\theta_0}(s),\ c_x(s) \leq C_p(s) + w(\theta_0) + \ell_x \tag{7}$$

To obtain the upper bound we sum up (7) for $p \in \{2, \ldots, m\}$, add $C_1(s)$ in the left and right hand part, and take $\sum_{\theta \notin \mathcal{T}'} w(\theta) = 0$. We will do this analysis below, collectively for cases 1 and 2.

CASE 2: $\theta_0 \neq \theta_1$ and θ_1 exists. Assume $\theta_0 < \theta_1$ and let R contain the indices of ascending machines which have at least one job of type θ_1 assigned (if $\theta_0 > \theta_1$, we consider the indices of descending machines). Let R be the indices of these machines and $R' \subseteq R$ be the indices of machines that are also assigned type θ_0 jobs (note that $\theta_0 \notin \mathcal{T}'_C$ if $R' \neq \emptyset$). If job x moves to a machine with index in $p \in R'$, the completion time of x becomes at most $C_p(s) - w(\theta_1) + \ell_x$ and $C_p(s) - w(\theta_1) + w(\theta_0) + \ell_x$ if $p \in R'' = R \setminus R'$. Since s is a PNE:

$$\forall p \in R',\ c_x(s) \leq C_p(s) - w(\theta_1) + \ell_x,\ \forall p \in R'',\ c_x(s) \leq C_p(s) - w(\theta_1) + w(\theta_0) + \ell_x \tag{8}$$

We will sum up inequalities (7) or (8) for $p \in \{2, \ldots, m\}$ depending on whether $p \in R$ or not. As in case 1 we add $C_1(s)$ to left and right hand parts and consider $\sum_{\theta \notin \mathcal{T}'} w(\theta) = 0$. Before summing note that when m is even, each type in $\mathcal{T}'_A \cup \mathcal{T}'_D$ has jobs assigned to at most $\frac{r}{2}$ machines, for $r = m$. When m is odd assume w.l.o.g. that there are $\frac{m+1}{2}$ descending machines. *We ignore one of them* - different than M_1 - in the summation (we assume $m \geq 3$; otherwise $m = 1$ and $C(s) = C(s^*)$). Then, in case 2, type θ_1 appears at most $\frac{r}{2}$ times, $r = m - 1$, in the remaining $m - 1$ machines.

$$\begin{aligned}
rC_1(s) &\leq \frac{r}{2}\left(\sum_{\theta \in \mathcal{T}'_A \cup \mathcal{T}'_D \setminus \{\theta_0\}} w(\theta) + \sum_{\theta \in \mathcal{T}'_C \setminus \{\theta_0\}} w(\theta)\right) + rw(\theta_0) + \sum_{j \in \mathcal{J}} \ell_j + (r-1)\ell_x \\
&= \frac{r}{2}\left(\sum_{\theta \in \mathcal{T}} w(\theta) + \sum_{j \in \mathcal{J}} \ell_j\right) + \frac{r}{2} w(\theta_0) + (r-1)\ell_x - \frac{r-2}{2}\sum_{j \in \mathcal{J}} \ell_j \qquad (9) \\
&\leq \frac{r}{2}\left(\sum_{\theta \in \mathcal{T}} w(\theta) + \sum_{j \in \mathcal{J}} \ell_j\right) + \frac{r}{2}\left(w(\theta_0) + \ell_x\right) \text{ since } \textstyle\sum_{j \in \mathcal{J}} \ell_j \geq \ell_x \qquad (10)
\end{aligned}$$

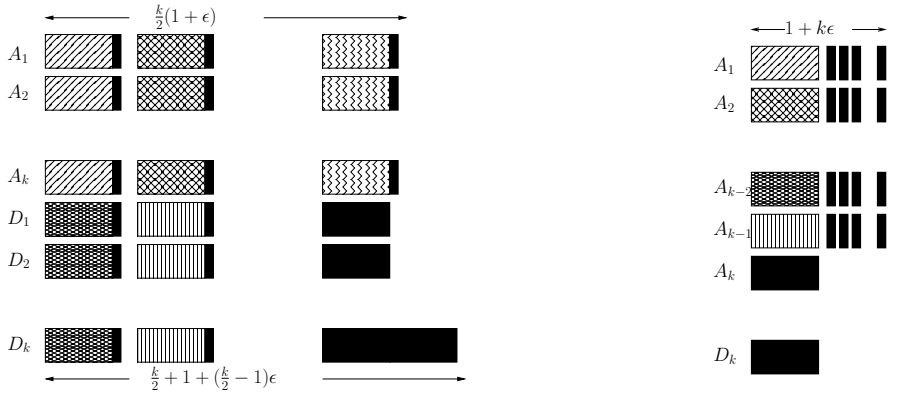


Figure 4: A lower bound for the performance of AD for even m and when $k = \frac{m}{2}$.

When $k \geq m$ we use (4a,b) with (10) to obtain $C_1(s) \leq \frac{m+1}{2} OPT$. When $k < m$ we rewrite (9) as:

$$C(s) \leq \frac{1}{r}\left(\sum_{\theta\in\mathcal{T}} w(\theta) + \sum_{j\in\mathcal{J}} \ell_j\right) + (\frac{1}{2} - \frac{1}{r})\left(\sum_{\theta\in\mathcal{T}} w(\theta) + \ell_x\right) + \frac{1}{2}\left(w(\theta_0) + \ell_x\right) \tag{11}$$

Using (4b,c), we get $kC(s^*) \geq \sum_{\theta\in\mathcal{T}} w(\theta) + \ell_x$ and replacing $r = m$ and $r = m-1$ for even and odd m respectively, yields the stated bounds with respect to k. □

Theorem 9 *The Price of Stability of the scheduling game with setup times under the* AD *mechanism is* $\frac{m+1}{2}$ *when* $k > m$ *and* $\frac{k+3}{2} - \rho$ *(*$\rho = \frac{k}{m}$ *when* m *is even and* $\rho = \frac{k-1}{m-1}$ *otherwise) when* $k \leq m$.

Proof. For $k > m$ we use the same example as in the proof of theorem 7, but replace the zero length jobs with very small $\epsilon > 0$ length. For AD the described assignment for $a, d \geq 1$ applies, and it is a PNE with makespan $m + 1 + (m-1)\epsilon$; the socially optimum makespan has length $2 + m\epsilon$. In any PNE, all jobs of types 1 and $2m-1$ will play exactly the strategies specified in the described PNE assignment, because a lower completion time is not achievable for them in any assignment. Inductively, jobs of types i and $2m-i$, $i = 2 \ldots m-1$, follow the same practice, given the strategies of jobs of types $i-1$, $2m-i+1$. For the jobs of type m, the strategies described in the aforementioned assignment are best possible, given the strategies of all other jobs. Therefore the described PNE is unique, hence $PoS \to \frac{m+1}{2}$ for $\epsilon \to 0$. The same uniqueness argument holds when $k \leq m$, for the instances given below.

For $k < m$, take k even and $m \geq 2k$. Assume that machines $1, \ldots, \lceil \frac{m}{2} \rceil$ are ascending index and machines $\lceil \frac{m}{2} \rceil + 1, \ldots, m$ are descending index ($\lfloor \frac{m}{2} \rfloor$ in total). Let $\hat{\theta}$ denote the *median* type index $\frac{k}{2} + 1$. For all types $\theta \neq \hat{\theta}$, that is $\theta \in \{1, 2, \ldots, k\} \setminus \{\hat{\theta}\}$, set $w(\theta) = 1$. For each type $\theta < \hat{\theta}$ assume $\lceil \frac{m}{2} \rceil$ jobs of small processing length $\epsilon > 0$. For each type $\theta > \hat{\theta}$ assume $\lfloor \frac{m}{2} \rfloor$ jobs of small processing length ϵ. Finally, let $w(\hat{\theta}) = 0$ and let there be $m - k + 1$ jobs of type $\hat{\theta}$ and processing length 1 each. Let us describe first the socially optimum configuration s^*. For each type $\theta \in \{1, \ldots, k\} \setminus \{\hat{\theta}\}$ assign its jobs to a dedicated machine for this type. Assign each of the jobs of type $\hat{\theta}$ to a separate machine each, out of the remaining $m - k + 1$ ones. The makespan of s^* is $C(s^*) = 1 + \lceil \frac{m}{2} \rceil \epsilon$. See the right of Figure 4 for an example when $m = 2k$.

Now let us describe a PNE configuration s. Each of the first $\lceil \frac{m}{2} \rceil$ ascending index machines has one job of each type $\theta \in \{1, \ldots, \hat{\theta} - 1\}$. Each of the last $\lfloor \frac{m}{2} \rfloor$ descending index machines has one job of each type $\theta \in \{\hat{\theta} + 1, \ldots, k\}$. See the left of Figure 4 for an example when $m = 2k$.

Finally, we assign appropriately the remaining length-1 jobs of type $\hat{\theta}$. In doing so, notice that right before this final assignment, the ascending index machines have currently makespan $\frac{k}{2}(1+\epsilon)$ and the descending index ones $(\frac{k}{2}-1)(1+\epsilon)$. Under this current partial assignment no unilateral deviation may occur, because same-type jobs have equal completion times (see left of Figure 4). No job of a type scheduled last on an ascending index machine may decrease its completion time, by switching to a decreasing index machine with makespan, because its batch of total length $(1+\epsilon)$ will be scheduled last on this machine as well.

Now assign first one job of type $\hat{\theta}$ to each of the $\lceil \frac{m}{2} \rceil$ descending index machines, so that their makespan becomes $\frac{k}{2}+\epsilon(\frac{k}{2}-1)$. Now continue assigning the remaining $m-k+1-\lfloor \frac{m}{2} \rfloor > 0$ (because $m \geq 2k$) type $\hat{\theta}$ jobs on descending index machines and increase their makespan to $\frac{k}{2}+1+\epsilon(\frac{k}{2}-1)$. In the left of Figure 4 we have added one last such job on the last descending index machine. Now because k is even and $m \geq 2k$, we have that $\rho = \frac{k}{m} \leq \frac{1}{2}$ when m is even and $\rho = \frac{k-1}{m-1} \leq \frac{1}{2}$ when m is odd. Then, we obtain $\lim_{\epsilon \to 0} \frac{C(s)}{C(s^*)} \geq \frac{k}{2}+1 \geq \frac{k+3}{2} - \rho$. Configuration s is a unique PNE, by a similar argument to the one used for the case $k > m$, which we omit for brevity. Therefore the lower bound applies to the PoS as stated. □

7 Research Directions

Let us summarize here some of the important current research directions that emerge from the study of recent literature. One of the most important questions emerging from the work of Caragiannis [14], concerns resolving the performance of *preemptvie* scheduling policies for unrelated machines scheduling. Caragiannis managed to break through a $\Theta(\log m)$ barrier of PoA by devising a mechanism with $PoA = \Theta\left(\frac{\log m}{\log \log m}\right)$ that does not always induce PNE. However, there is no general lower bound at the time of this writing, with respect to the performance of preemptive policies.

Two related questions concern non-preemptive mechanisms and stems from the results of Azar *et al.* [10]; they proved an $\Omega(\log m)$ lower bound for the PoA of the class of deterministic ordering mechanisms, and devised a mechanism with $PoA = O(\log m)$, that does not always induce PNE though. The first question concerns devising a mechanism that achieves the same upper bound and always induces PNE that can be computed efficiently. The second concerns breaking through the $\Omega(\log m)$ lower bound using randomization; what is the performance of randomized non-preemptive mechanisms?

Concerning randomized mechanisms, the performance of `RANDOM` is yet to be fully determined. It is not known whether `RANDOM` induces PNE for $Q||C_{\max}$, $B||C_{\max}$ or $R||C_{\max}$ in general; the results of Dürr and Nguyen Kim [22] show that `RANDOM` does induce PNE for uniformly related machines, only when the speeds do not differ too much. This mechanism is the only known non-preemptive non-clairvoyant mechanism, which also achieves a bounded PoA even for unrelated machines scheduling. Therefore, settling its theoretical merits remains an important open question.

We believe that designing and analyzing of coordination mechanisms has reached a maturity, that should allow it to serve as a model for decentralized congestion manipulation in large scale systems of distributed resources. To this end, more involved congestion effects should be studied, motivated by the practice of these systems. The introduction of *setup times* is such an attempt, to bring more involved considerations in the scene, that require the design of novel machinery for dealing with them. Some open questions naturally arise from this attempt. Notice that the universe of types $\mathcal{U}$ is required to be huge in the proof of Theorem 7 (double exponential). This size is non-realistic for most interesting practical settings. Is there a lower size of $\mathcal{U}$ that also yields

$PoA \geq \frac{m+1}{2}$ for type ordering mechanisms? For example, the proof of Theorem 7 requires that $|\mathcal{U}| \geq 9$ when $m = 2$, although $|\mathcal{U}| \geq 3$ is enough. The performance of type ordering mechanisms is not fully characterized by Theorem 7; there may be certain sizes of $|\mathcal{U}|$ below which these mechanisms may perform better. Another interesting issue to be examined, is when type ordering mechanisms are a priori aware of the subset of types $\mathcal{T}$ that corresponds to players $\mathcal{J}$. What is the impact of such an a priori knowledge to the achievable PoA by type ordering mechanisms? Finally, we have not considered in the context of setup times any simply local mechanisms, or more challenging machine environments (uniformly related or unrelated machines). All these constitute very interesting aspects for future developments on the subject.

References

[1] N. Andelman, M. Feldman, and Y. Mansour. Strong price of anarchy. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 189–198, 2007.

[2] Eric Angel, Evripidis Bampis, and Fanny Pascual. Truthful algorithms for scheduling selfish tasks on parallel machines. *Theor. Comput. Sci.*, 369(1-3):157–168, 2006.

[3] Eric Angel, Evripidis Bampis, and Nicolas Thibault. Randomized truthful algorithms for scheduling selfish tasks on parallel machines. In *Proceedings of the 9th Latin American Theoretical Informatics Symposium (LATIN) - to appear*, 2010.

[4] E. Anshelevich, A. Dasgupta, J. M. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The Price of Stability for Network Design with Fair Cost Allocation. *SIAM Journal on Computing*, 38(4):1602–1623, 2008.

[5] J. Aspnes, Y. Azar, A. Fiat, S. A. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM*, 44(3):486–504, 1997.

[6] Vincenzo Auletta, Roberto De Prisco, Paolo Penna, and Giuseppe Persiano. Deterministic truthful approximation mechanisms for scheduling related machines. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS), Springer LNCS 2996*, pages 608–619, 2004.

[7] Robert J. Aumann. Acceptable points in games of perfect information. *Pacific Journal of Mathematics*, 10:381–417, 1960.

[8] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999.

[9] B. Awerbuch, Y. Azar, Y. Richter, and D. Tsur. Tradeoffs in worst-case equilibria. *Theoretical Computer Science*, 361(2–3):200–209, 2006.

[10] Y. Azar, K. Jain, and V. S. Mirrokni. (almost) optimal coordination mechanisms for unrelated machine scheduling. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 323–332, 2008.

[11] Y. Azar, J. Naor, and R. Rom. The competitiveness of on-line assignments. *Journal of Algorithms*, 18:221–237, 1995.

[12] D. Braess. Über ein paradoxon aus der Verkehrsplanung. *Unternehmensforschung*, 12:258–268, 1968.

[13] D. Braess. On a Paradox of Traffic Planning. *Transportation Science*, 39(4):446–450, 2005.

[14] I. Caragiannis. Efficient coordination mechanisms for unrelated machine scheduling. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 815–824, 2009.

[15] B. Chen. A better heuristic for preemptive parallel machine scheduling with batch setup times. *SIAM Journal on Computing*, 22:1303–1318, 1993.

[16] B. Chen, Y. Ye, and J. Zhang. Lot-sizing scheduling with batch setup times. *Journal of Scheduling*, 9(3):299–310, 2006.

[17] Y. Cho and S. Sahni. Bounds for list schedules on uniform processors. *SIAM Journal on Computing*, 9:91–103, 1980.

[18] G. Christodoulou, L. Gourvès, and F. Pascual. Scheduling Selfish Tasks: About the Performance of Truthful Algorithms. In *Proceedings of the International Computing and Combinatorics Conference (COCOON), Springer LNCS 4598*, pages 187–197, 2007.

[19] G. Christodoulou, E. Koutsoupias, and A. Nanavati. Coordination Mechanisms. In *Proceedings of the International Colloquium on Automata,Languages and Programming (ICALP), Springer LNCS 3142*, pages 345–357, 2004.

[20] A. Czumaj and B. Vöcking. Tight bounds for worst-case equilibria. *ACM Transactions on Algorithms*, 3(1), 2007.

[21] G. Dobson. Scheduling independent tasks on uniform processors. *SIAM Journal on Computing*, 13:705–716, 1984.

[22] C. Dürr and T. Kim Nguyen. Non-clairvoyant Scheduling Games. In *Proceedings of the International Symposium on Algorithmic Game Theory (SAGT), Springer LNCS 5814*, pages 135–146, 2009.

[23] E. Even-Dar, A. Kesselman, and Y. Mansour. Convergence time to Nash equilibrium in load balancing. *ACM Transactions on Algorithms*, 3(3), 2007.

[24] A. Fiat, H. Kaplan, M. Levy, and S. Olonetsky. Strong Price of Anarchy for Machine Load Balancing. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP), Springer LNCS 4596*, pages 583–594, 2007.

[25] G. Finn and E. Horowitz. A linear time approximation algorithm for multiprocessor scheduling. *BIT*, 19:312–320, 1979.

[26] D. K. Friesen. Tighter bounds for LPT scheduling on uniform processors. *SIAM Journal on Computing*, 16:554–560, 1987.

[27] M. Gairing, T. Lücking, M. Mavronicolas, and B. Monien. Computing Nash equilibria for scheduling on restricted parallel links. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 613–622, 2004.

[28] M. Gairing, T. Lücking, M. Mavronicolas, and B. Monien. The Price of Anarchy for Restricted Parallel Links. *Parallel Processing Letters*, 16(1):117–132, 2006.

[29] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H.Freeman & Co Ltd, 1979.

[30] L. Gourvès, J. Monnot, and O. Telelis. Selfish Scheduling with Setup Times. In *Proceedings of the International Workshop on Internet and Network Economics (WINE), Springer LNCS 5929*, pages 292–303, 2009.

[31] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[32] R. L. Graham. Bounds for certain multiprocessing anomalies. *SIAM Journal of Applied Mathematics*, 17:416–429, 1969.

[33] O. H. Ibarra and C. E. Kim. Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors. *Journal of the ACM*, 24(2):280–289, 1977.

[34] N. Immorlica, L. Li, V. S. Mirrokni, and A. Schulz. Coordination Mechanisms for Selfish Scheduling. *Theoretical Computer Science*, 410:1589–1598, 2009.

[35] S. Jukna. *Extremal Combinatorics with Applications in Computer Science*. Springer-Verlag, 2001.

[36] E. Koutsoupias and C. H. Papadimitriou. Worst-case Equilibria. In *Proceedings of the International Symposium on Theoretical Aspects of Computer Science (STACS), Springer LNCS 1543*, pages 404–413, 1999.

[37] J. K. Lenstra, D B. Shmoys, and E. Tardos. Approximation Algorithms for Scheduling Unrelated Parallel Machines. *Mathematical Programmming*, 46:259–271, 1990.

[38] D. Monderer and L. Shapley. Potential Games. *Games and Economic Behanior*, 14:124–143, 1996.

[39] Ahuva Mu'alem and Michael Schapira. Setting lower bounds on truthfulness: extended abstract. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1143–1152, 2007.

[40] J. Nash. Noncooperative Games. *Annals of Mathematics*, 54:289–295, 1951.

[41] N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, editors. *Algorithmic Game Theory*. Cambridge University Press, 2007.

[42] Noam Nisan and Amir Ronen. Algorithmic mechanism design (extended abstract). In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 129–140, 1999.

[43] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[44] A. C. Pigou. *The Economics of Welfare*. Macmillan, 1920.

[45] O. Rozenfeld and M. Tennenholtz. Strong and Correlated Strong Equilibria in Monotone Congestion Games. In *Proceedings of the International Workshop on Internet & Network Economics (WINE), Springer LNCS 4286*, pages 74–86, 2006.

[46] P. Schuurman and T. Vredeveld. Performance Guarantees of Local Search for Multiprocessor Scheduling. *INFORMS Journal on Computing*, 19(1):52–63, 2007.

[47] P. Schuurman and G. J. Wöginger. Preemptive scheduling with job-dependent setup times. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 759–767, 1999.

[48] G. J. Wöginger and Z. Yu. A heuristic for preemptive scheduling with set-up times. *Computing*, 49(2):151–158, 1992.