(Almost) Optimal Coordination Mechanisms for Unrelated Machine Scheduling

YOSSI AZAR * KAMAL JAIN † VAHAB MIRROKNI ‡

Abstract

We investigate the influence of different algorithmic choices on the approximation ratio in selfish scheduling. Our goal is to design local policies that minimize the inefficiency of resulting equilibria. In particular, we design optimal coordination mechanisms for unrelated machine scheduling, and improve the known approximation ratio from $\Theta(m)$ to $\Theta(\log m)$, where m is the number of machines.

A *local* policy for each machine orders the set of jobs assigned to it only based on parameters of those jobs. A *strongly local* policy only uses the processing time of jobs on the the same machine. We prove that the approximation ratio of any set of strongly local ordering policies in equilibria is at least $\Omega(m)$. In particular, it implies that the approximation ratio of a greedy shortest-first algorithm for machine scheduling is at least $\Omega(m)$. This closes the gap between the known lower and upper bounds for this problem, and answers an open question raised by Ibarra and Kim [16], and Davis and Jaffe [10]. We then design a local ordering policy with the approximation ratio of $\Theta(\log m)$ in equilibria, and prove that this policy is optimal among all local ordering policies. This policy orders the jobs in the non-decreasing order of their inefficiency, i.e, the ratio between the processing time on that machine over the minimum processing time. Finally, we show that best responses of players for the inefficiency-based policy may not converge to a pure Nash equilibrium, and present a $\Theta(\log^2 m)$ policy for which we can prove fast convergence of best responses to pure Nash equilibria.

*azar@tau.ac.il. Microsoft Research, Redmond and Tel-Aviv University, Tel-Aviv, 69978, Israel. Research supported in part by the Israel Science Foundation.

†kamalj@microsoft.com. Microsoft Research, Redmond.

‡mirrokni@microsoft.com. Microsoft Research, Redmond.

1 Introduction

In order to study the influence of algorithmic choices in the presence of selfish users, we need to study the inefficiency of equilibrium points. The approximation ratio of a decentralized algorithm in lack of coordination can be captured by the the worst case performance of a Nash equilibrium over a global social optimum, i.e., the price of anarchy [19]. A natural question is to design decentralized algorithms to reduce the price of anarchy for selfish users. In these algorithms, a central authority can only design protocols and define rewarding rules and hope that the independent and selfish choices of the users -given the rules of the protocols- result in a socially desired outcome. To this end, different approaches have been proposed such as imposing economic incentives in the form of monetary payments [5, 8, 13], and using the Stackelberg strategy [4, 18, 22, 25] which is enforcing strategies upon a fraction of users. The main disadvantage of these two strategies is that they assume global knowledge of the system and thus have high communication complexity. In many settings, it is important to be able to compute mechanisms locally. A different approach, which is the focus of our paper, is called *coordination mechanisms*, first introduced by Christodoulou, Koutsoupias and Nanavati [7]. A coordination mechanism is a *local* policy that assigns a cost to each strategy s, where the cost of s is a function of the users who have chosen s.

Consider, for example, the *selfish scheduling game* in which there are n jobs owned by independent users, m machines and a processing time p_{ij} for job i on machine j. We concentrate on *pure strategies* case where each user selects one machine to assign his job. Each user is aware of the decisions made by other users and behaves selfishly. Specifically, it wishes to minimize its completion time by assigning its job to the machine at which its job would complete first. The global objective however, is to minimize the make span - maximum completion time. A coordination mechanism [7] for this game is a set of local policies, one for each machine, that determines how to schedule jobs assigned to that machine. A machine's policy is a function only of the jobs assigned to that machine. This allows the policy to be implemented in a completely distributed and local fashion.

We mainly study *ordering* policies. Ordering policies characterize all deterministic non-preemptive policies that satisfy the independence of irrelevant alternatives or IIA property[1]. We consider *strongly local* policies in which the ordering of jobs on machine j only depends on the processing time of the set S_j of jobs on machine j, and *local* policies in which the ordering for machine j depends on all parameters of jobs in S_j. Two examples of the strongly local ordering policies are the ShortestFirst and LongestFirst policies in which we order the jobs in non-decreasing and non-increasing order of their processing times, respectively.

Several local policies have been studied for machine scheduling problems, both in the context of greedy or local search algorithms for machine scheduling [16, 12, 23, 10, 1, 3, 6, 26], and also in the context of coordination mechanisms [19, 9, 7, 17]. Ibarra and Kim [16] present a greedy shortest-first algorithm and proved an upper bound of m for its approximation factor. It has been shown that the output of this greedy algorithm is equivalent to the pure Nash equilibria of the ShortestFirst policy in selfish scheduling [17]. An $\Omega(\log m)$ lower bound has been proved for the approximation factor of this algorithm [10].

[1]For the definition of non-preemptive policies and the IIA property, see Section 2.

Our Results. In Section 3, we show that any set of strongly local ordering policies results in the price of anarchy of $\Omega(m)$. This result implies that the ShortestFirst policy has the price of anarchy of $\Theta(m)$. Moreover, this bound closes the gap between the known lower and upper bounds of the approximation ratio of the shortest-first greedy algorithm (i.e., Algorithm D by Ibarra and Kim [16]) and answers an open question originally raised in 1977 [16, 10, 17].

In Section 4, we design a local ordering policy for which the price of anarchy is $\Theta(\log m)$. Specifically, on each machine, we order the jobs by in the non-decreasing order of their *inefficiency*, i.e., the ratio of the job's processing time on this machine to its fastest processing time. Also we show that any deterministic non-preemptive set of local policies satisfying the IIA property results in the price of anarchy of $\Omega(\log m)$. In particular, it shows that the inefficiency-based policy is almost optimal among local ordering policies. In Section 6, we study existence of pure Nash equilibria for ordering policies and prove convergence to pure Nash equilibria for some special cases. The main result of this section is that pure Nash equilibria may not exist for the inefficiency-based policy and the best responses of players may not converge to it. Finally, in Section 7, we design a local policy for which the best-response dynamics of players converges to a pure Nash equilibrium in polynomial time and the price of anarchy is $\Theta(\log^2 m)$.

Related work. Coordination mechanisms are related to local search algorithms. Starting from a solution, a local search algorithm iteratively moves to a neighbor solution which improves the global objective. This is based on a neighborhood relation that is defined on the set of solutions. The local improvement moves in the local search algorithm correspond to the best-response moves of users in the game defined by the coordination mechanism. The speed of convergence and the approximation factor of local search algorithms for scheduling problems have been studied in several papers [10, 11, 12, 16, 23, 24, 26, 1, 3]. Vredeveld surveyed some of the results on local search algorithms for scheduling problems in his thesis [26].

Ibarra and Kim [16] analyzed several greedy algorithms for unrelated machine scheduling. In particular, they proved that the shortest-first greedy algorithm is an m-approximation for the maximum completion time. Davis and Jaffe [10] showed that the approximation factor of this greedy algorithm is at least $\log m$. The best known approximation factor is given by a central 2-approximation algorithm due to Lenstra, Shmoys and Tardos [20].

A widely studied scheduling policy is the Makespan policy in which we process all jobs on the same machine in parallel so that the completion time of a job on machine j is the makespan of machine j. The price of anarchy of this policy is unbounded even for two machines. Tight price of anarchy results for (mixed) Nash equilibria are known for this policy for special cases of the unrelated scheduling problem [9, 2, 14, 19].

Coordination mechanism design was introduced by Christodoulou, Koutsoupias and Nanavati [7]. In their paper, they analyzed the LongestFirst policy for $P||C_{\max}$ and also studied a selfish routing game. Immorlica, Li, Mirrokni, and Schulz [17] study four coordination mechanisms for four types of machine scheduling problems and survey the results for these problems. They further study the speed of convergence to equilibria and existence of pure Nash equilibria for the ShortestFirst and LongestFirst policies.

2 Preliminaries

The *unrelated machine scheduling problem* or $R||C_{\max}$ is defined as follows: there are m machines and n users, where user i ($i = 1, \ldots, n$) has a job that can be assigned to any machine. Job i for $i = 1, \ldots, n$ is associated with an m-vector $\vec{p}_i$, where p_{ij} indicates the processing time of job i if assigned to machine j.

Given an instance of the $R||C_{\max}$ problem, we define the *global optimum* (denote it by OPT) to be the assignment of jobs to machines that minimizes the makespan, i.e., the maximum completion time. We slightly abuse the notation and use OPT to denote also the value of the optimal solution. The goal is to find a schedule which minimizes the total makespan.

In *selfish scheduling*, each job is owned by an independent user whose goal is to minimize the completion time of his job. In order to make the selfish users to take globally near-optimal actions, we can define the following notion of *coordination mechanism* [7]. A coordination mechanism is a set of *local scheduling policies*, one for each machine. A scheduling policy $\mathcal{P}_j$ for a machine j maps any set S of jobs on machine j to a schedule of all jobs in S. The policy is run locally at a machine, and so does not have access to information regarding the global state of the system, for example the set of jobs scheduled on other machines. As a result, for any policy $\mathcal{P}_j$ and a set of jobs S for machine j, each job $i \in S$ is mapped to a completion time $\mathcal{P}_j(S, i)$.

A scheduling policy $\mathcal{P}_j$ is *strongly local* if it only looks at the processing time of jobs in S_j on machine j and assign each job $i \in S_j$ a completion time. A strongly local policy $\mathcal{P}_j$ may have an arbitrary tie-breaking rule for jobs of the same processing time. In order to formally define the *tie-breaking rules*, we assume that each job has a *unique ID* and a local policy's tie breaking rule is a function of the set of IDs of jobs. A *local* policy looks at all parameters of jobs assigned to machine j and assigns each job $i \in S_j$ a completion time . Note that a local policy that is not strongly local may use the processing times of the jobs of S_j on other machines, but it does not have any information about other jobs that are not assigned to this machine.

A policy is a *non-preemptive* policy if it processes each job in an un-interrupted fashion without any delay. A policy is a *preemptive* policy if it can interrupt jobs during the scheduling and can put some delay on the machine. We say that a policy satisfies the *independence of irrelevant alternatives* or *IIA* property if for any set S of jobs and any two jobs $i, i' \in S$, if i has a smaller completion time than i' in S, then i should have a smaller completion time than i' in any set $S \cup \{k\}$. In other words, whether i or i' is preferred should not be changed by the availability of a job k. The IIA property appears as an axiom in voting theory, bargaining theory, and logic [27].

A scheduling policy is an *ordering* policy if for each instance of the scheduling problem, it orders the jobs non-preemptively based on a global ordering. It is not hard to show that any deterministic non-preemptive policy that satisfies the IIA property is an ordering policy. The ShortestFirst and LongestFirst policies are ordering policies in which we order the jobs in non-decreasing and non-increasing order of their processing times, respectively. Note that the ShortestFirst and LongestFirst may have arbitrary tie-breaking rules based on the IDs of jobs.

A special class of the $R||C_{\max}$ problem is the machine scheduling for *restricted assignment* ($B||C_{\max}$) in which each job i can be scheduled on a subset T_i of machines, i.e., p_{ij} is equal to p_i if $j \in T_i$ and is equal to ∞ otherwise.

3 A Lower Bound for Strongly Local Policies

In this section, we show that the approximation ratio of any set of strongly local ordering policies is $\Omega(m)$. In the next section, we present a local ordering policy that achieves the factor $O(\log m)$ and will prove a matching lower bound for local policies.

Theorem 3.1 *The price of anarchy for any set of deterministic non-preemptive strongly local policies satisfying the IIA property is at least* $\Omega(m)$.

Proof: We observe that any deterministic non-preemptive policy satisfying the IIA property is an ordering policy. As a result, we show that for any strongly local ordering policy, the price of anarchy is at least $\Omega(m)$. Let $n_j = \frac{2(m-1)!}{(j-1)!}$ for $1 \leq j \leq m$ and $n = \sum_{j=1}^{m} n_j$. Consider the set of m machines and a set $\mathcal{P}_1, \ldots, \mathcal{P}_m$ of strongly local ordering policies on these m machines. Given this set of policies, we construct an instance of n jobs for which the price of anarchy is $\Omega(m)$. Since the policy $\mathcal{P}_j$ is a strongly local ordering policy, it only looks at the processing time of jobs on machine j and their IDs. As a result, if the processing time of all jobs on machine j is equal to $\frac{(j-1)!}{(m-1)!}$, $\mathcal{P}_j$ orders the jobs based on a global ordering of IDs. Let σ_j be this ordering on the IDs of jobs. We construct an instance in which all jobs that can be scheduled on machine j has the same processing time $\frac{(j-1)!}{(m-1)!}$. We define a family of subsets $S_1, \ldots, S_m$ such that $|S_j| = n_j$ for $1 \leq j \leq m$. Jobs in S_j can be scheduled only on machines j and $j+1$ for $1 \leq j \leq m$ (jobs in S_m can only go to machine m). The processing time of all jobs on machine j is $\frac{(j-1)!}{(m-1)!} = \frac{2}{n_j}$.

In order to define S_j's, we use the following notation. Given any ordering σ on the IDs of n jobs, a set $T \subset \{1, \ldots, n\}$ of IDs of jobs, and a number k, let $\sigma^k(T)$ be the set of first k IDs in ordering σ that are in set T. In particular, $\sigma^k(\{1, 2, \ldots, n\})$ is the set of first k IDs in an ordering σ. Also, for $1 \leq j \leq m$, let $w_j = \sum_{t=1}^{j} n_t$. Now, we are ready to define S_js as a function of all orderings σ_k as follows: S_m is the set of last n_m IDs in the ordering σ_m, i.e.,

$$S_m := \{\sigma_m(n), \sigma_m(n-1), \ldots, \sigma_m(n-n_j+1)\}, \text{ and } M_m := \{1, 2, \ldots, n\} \backslash S_m.$$

Also, for each j $(m-1 \geq j \geq 1)$, we have

$$M_j := \sigma_j^{w_{j-1}}(M_{j+1}), \text{ and } S_j := M_{j+1} \backslash M_j.$$

We claim that the price of anarchy of this instance is $\frac{m}{2}$. An optimal solution of this instance schedules jobs of set S_j on machine j for $1 \leq j \leq m$. The makespan of this schedule is $\frac{2}{n_j} n_j = 2$. We prove the following Lemma on this instance.

Lemma 3.2 *In any pure Nash equilibrium of this instance, the makespan of machine j is equal to j for any j from 1 to m. In particular, half of the jobs of S_j are scheduled on machine j and half of them are scheduled on machine $j+1$ for any j from 1 to $m-1$.*

Proof: By the construction of S_j $(1 \leq j \leq m)$, policy $\mathcal{P}_j$ puts all jobs of S_j after all jobs of S_{j-1} on machine j, since all jobs of M_j go before all jobs of S_j on machine j and $S_{j-1} \subseteq M_j$. We prove the lemma by induction on j. For the base of the induction, define S_0 as an empty set and machine

0 as a dummy machine. For the induction hypothesis, assume that for $k \leq j-1$, in any pure Nash equilibrium, half of the jobs of S_k are scheduled on machine k and half of them are scheduled on machine $k+1$. As a result, the load of machine k for $k \leq j-1$ is exactly k, and the load of machine j from jobs in S_{j-1} is $\frac{n_{j-1}}{2}\frac{2}{n_j} = j-1$. We prove that in any pure Nash equilibrium, half of jobs of S_j go to machine j and half of them go to machine $j+1$. We prove the induction step by contradiction. If in a pure Nash equilibrium, less than half of the jobs in S_j are at machine $j+1$, then the completion time of the last job q of S_j on machine j is strictly more than $j-1+\frac{n_j}{2}\frac{2}{n_j} = j$, since all jobs of S_{j-1} will be scheduled before all jobs of S_j on machine j. Since only jobs in S_j and S_{j+1} can be scheduled on machine $j+1$ and $q \in S_j$ will be scheduled before any job S_{j+1}, if q moves to machine $j+1$, its completion time is at most $\frac{n_j}{2}\frac{2}{n_{j+1}} = j$. Therefore, q has incentive to switch to machine $j+1$. In addition, if in a pure Nash equilibrium, more than half of the jobs in S_j are scheduled on machine $j+1$, then the completion time of the last job is more than j on machine $j+1$ and this job can move to machine j and improve its completion time. This proves the induction step. ■

The above lemma proves that in any pure Nash equilibrium the makespan of machine m is m, and therefore, the price of anarchy for this instance is at least $\frac{m}{2}$. This completes the proof of the theorem. ■

Since the ShortestFirst policy is a strongly local policy, the above theorem implies that the price of anarchy of the ShortestFirst policy is at least $\frac{m}{2}$. Immorlica et.al. [17] observed that the set of pure Nash equilibria of the ShortestFirst policy is equivalent to the output of the shortest-first greedy algorithm of Ibarra and Kim [16]. Therefore, the above lower bound implies the lower bound of $\frac{m}{2}$ for the shortest-first greedy algorithm, and answers an open question raised by Ibarra and Kim [16], and Davis and Jaffe [10]. As a result, we have the following theorem:

Theorem 3.3 *The price of anarchy of the* ShortestFirst *policy is at least* $\frac{m}{2}$*. In particular, it implies that the approximation factor of* m *proved by Ibarra and Kim [16] for the shortest-first greedy algorithm is almost tight.*

It is worth mentioning that the proof of Theorem 3.1 uses jobs of the same size and argue about tie breaking rules. For the ShortestFirst policy, we can actually perturb the example such that all jobs have different sizes, and hence the shortest-first algorithm is uniquely define. A proof of Theorem 3.3 without jobs of the same size is given in the appendix.

4 A Logarithmic Upper Bound

In this section, we give a deterministic non-preemptive local policy with the IIA property for which the price of anarchy is $\Theta(\log m)$. Recall that in the unrelated links model, a job i is associated with an m-vector $\vec{p}_i = (p_{i1}, \ldots, p_{im})$ specifying its processing time on each machine. Denote by $p_i = \min_j p_{ij}$ which is the fastest processing time of that job on any of the machines. The inefficiency of job i on machine j is $e_{ij} = p_{ij}/p_i$. By definition $e_{ij} \geq 1$ for all i and j. The *min-weight* of a set S of jobs is equal to $\sum_{i \in S} p_i$. Also, let $W = \sum_{1 \leq i \leq n} p_i$.

The *inefficiency-based* policy for machine j orders the jobs assigned to it in the non-decreasing order of their inefficiency e_{ij}.

Theorem 4.1 *The price of anarchy for $R||C_{\max}$ for the inefficiency-based policy is at most $2\log m+4$.*

Proof: Given this ordering strategy for each machine and a pure Nash equilibrium, we partition the assignment into layers. For any $k \geq 0$, we denote by M_{kj} all jobs (and parts of jobs) that are processed on machine j after time $2kOPT$. Let M_k be the union over all machines j of M_{kj}, i.e., $M_k = \cup_{1\leq j\leq m} M_{kj}$.

Let R_{kj} denote the min-weight of jobs in M_{kj}, i.e., $R_{kj} = \sum_{i\in M_{kj}} p_i$. Specifically if job i is partially processed on machine j for x units of time after time $2kOPT$, then its contribution to R_{kj} is $x/e_{ij} = xp_i/p_{ij}$.

Let $R_k = \sum_{1\leq j\leq m} R_{kj}$ which is the min-weight of jobs processed after time $2kOPT$. Note that $R_0 = W$ since it is the total min-weight of all jobs. Our main lemma is the following:

Lemma 4.2 *For all $k \geq 1$, $R_k \leq \frac{1}{2} \cdot R_{k-1}$.*

Proof: Let O_j be the set of jobs processed on machine j by OPT. Let O_{kj} be the intersection of O_j and M_k. Let f_{kj} be the minimum inefficiency of all jobs in O_{kj} in the equilibrium assignment. Each job in O_{kj} could switch to machine j. If O_{kj} is not empty, then in the equilibrium assignment, machine j is processing jobs of inefficiency of at most of f_{kj} up to time $(2k-1)OPT$ otherwise the job with the minimum inefficiency in O_{kj} would move to machine j and complete by time $(2k-1)OPT + OPT = 2kOPT$.

Hence, machine j processes jobs of inefficiency at most f_{kj} between times $(2k-2)OPT$ and $(2k-1)OPT$ which implies that

$$R_{k-1,j} - R_{kj} \geq OPT/f_{kj} \ .$$

On the other hand, all jobs in O_{kj} are processed by OPT on machine j with inefficiency of at least f_{kj} and hence their total min-weight is at most OPT/f_{kj}. By combining the last two inequalities, we conclude that $R_{k-1,j} - R_{kj}$ is at least the min-weight of jobs in O_{kj}. Summing up over all j, we get that

$$R_{k-1} - R_k \geq R_k,$$

since M_k is the union of O_{kj} over all machines j. We conclude that $R_{k-1} \geq 2R_k$ as required.

■

We are now ready to complete the proof of the Theorem. By applying the main lemma, $b = \lceil \log m \rceil$ times we get that

$$R_b \leq \frac{1}{m} \cdot R_0 = \frac{W}{m} \leq OPT \ .$$

In particular, this implies that the total processing time of jobs of inefficiency 1 in M_b is at most OPT. Hence each such job ends by time $(2b)OPT + OPT = (2b+1)OPT$. Consider a job that has not been completed by time $2bOPT$. Such job has an option to run on a machine of inefficiency of 1. In that case it would start no later than $(2b+1)OPT$ and would finish no later than $(2b+1)OPT + OPT = (2b+2)OPT$ (since its min-weight is at most OPT). Since the assignment is a Nash equilibrium, we conclude that the maximum completion of any job is at most $(2b+2)OPT \leq (2\log m + 4)OPT$. ■

Remark 1 *The above proof can be extended to bound the price of anarchy for mixed Nash equilibria of the inefficiency-based policy. We can prove a lemma for mixed strategies similar to Theorem 4.1 with the bound of $O(\log m)$. Then using the Hoeffding inequality and the framework developed by Czumaj and Vocking [9] (and also used by Awerbuch et. al. [2]), we can prove that the price of anarchy for mixed Nash equilibria for this policy is $\Theta(\log m)$.*

5 A Lower Bound for Local Policies

In Section 3, we proved that the price of anarchy for any strongly local ordering set of policies is at least $\Omega(m)$. Here, we show that the price of anarchy for any set of local ordering policies is at least $\Omega(\log m)$. As a warm-up example, we show that our analysis is almost tight for the inefficiency-based policy.

Theorem 5.1 *The price of anarchy for $R||C_{\max}$ when the ordering strategy is by non-decreasing inefficiency is at least $\log m$.*

Proof: We use a standard example to show that even for the restricted assignment model ($B||C_{\max}$) the price of anarchy of this strategy is at least $\log m$. Not that for $B||C_{\max}$ the inefficiency of every job is precisely 1 on any legal machine for that job. Hence the algorithm may order the jobs on each machine in any order. In this proof, we assume a global tie breaking rule on the order of all jobs. Without loss of generality a job with a lower index has a higher priority (otherwise we can rename the jobs). In the example, there are $m = 2^q$ machines and $m - 1$ jobs. All jobs have unit size. Each job can be assigned to two machines. The jobs are partitioned into $\log m$ groups. For $1 \leq k \leq q$, there are $m/2^k$ jobs in group k. Job l of group k for $1 \leq l \leq m/2^k$ can be assigned to machines l and $m/2^k + l$. The optimal algorithm can assign that job to machine $m/2^k + l$ and get a makespan of 1. We claim that if this job is assigned to machine l, it is a Nash equilibrium and results in a makespan of $\log m$ (machine 1 has $\log m$ completion time). It is easy to verify that all jobs in group k have a completion time of k and if they would move to the other option they would still have a completion time of k. Hence this assignment is a Nash Equilibrium which completes the proof. ■

Now, we use the structure of the standard example in Theorem 5.1 to prove the following general lower bound:

Theorem 5.2 *The price of anarchy for all deterministic non-preemptive local policies satisfying the IIA property for $R||C_{\max}$ is at least $\Omega(\log m)$.*

Proof: Without loss of generality, we assume that $m = 2^q$. We recall that deterministic non-preemptive local policies satisfying the IIA property correspond to ordering the jobs in a certain order according to all parameters of the jobs assigned to that machine. That means that the order depends on the IDs of jobs and their full vector of processing times on all machines. Given a set of local ordering policies, we construct an instance similar to the example used in Theorem 5.1. We start with $\frac{m^2-m}{2}$ jobs from which exactly $m-1$ jobs are used in the final instance. In particular, all jobs are of unit size and can be assigned to precisely two machines. Moreover, the ID of a possible job that can be assigned to machine

j and machine j' is unique (say it is $mj + j'$). If we restrict ourselves only to these types of jobs, then there are at most $m - 1$ jobs that can be assigned to each machine j. Specifically, these jobs can be described as (j, j') for all $j \neq j'$, since all remaining parameters (i.e. ID and the full load vector) are exact functions of the pair (j, j'). A local policy of each machine j for any $1 \leq j \leq m$ corresponds to an ordering of these jobs to be processed on machine j. Let σ_j be this ordering.

Let $A^0 = \{1, \ldots, m\}$ and $J^0 = \emptyset$. For k from 1 to $\log m$, we construct A^k and J^k from A^{k-1} as follows: first, let $A^k = \emptyset$, and $J^k = \emptyset$. We perform the following process $\frac{m}{2^k}$ times: Choose an arbitrary machine j from A^{k-1}. Find the job of the highest priority to run on machine j among all jobs (j, j') where $j' \in A^{k-1}$, and denote its ID by $(j, m^k(j))$, i.e., $(j, m^k(j))$ is the first job in σ_j among jobs $(j, j') \in A^{k-1} \times A^{k-1}$. Then, let $A^k = A^k \cup \{j\}$ and $J^k = J^k \cup \{(j, m^k(j))\}$. Also, let $A^{k-1} = A^{k-1} \backslash \{j, m^k(j)\}$. At the end of the process, A^{k-1} becomes empty, A^k has $\frac{m}{2^k}$ indices, and J^k has $\frac{m}{2^k}$ jobs.

The set of jobs for the final instance is the union of the jobs J^k for $1 \leq k \leq \log m$, i.e., $\cup_{1 \leq k \leq \log m} J^k$. Hence we have $m-1$ jobs in the resulting instance. The following solution of makespan 1 is the optimal solution: assign job $(j, m^k(j)) \in J^k$ to machine $m^k(j)$. Consider an assignment $\mathcal{A}$ in which each job $(j, m^k(j)) \in J^k$ is assigned to machine j. We prove that this assignment is a pure Nash equilibrium.

Using induction on k, we prove that for each k from 1 to $\log m$, in assignment $\mathcal{A}$, each job $(j, m^k(j)) \in J^k$ is completed exactly at time k on machine j. Moreover, if it switches to machine $m^k(j)$, its completion time is not less than k. For the base of induction, job $(j, m^1(j)) \in J^1$ has more priority than all jobs $(j, m^{k'}(j)) \in J^{k'}$ for $2 \leq k' \leq \log m$, and hence, its completion time is 1. Also, this job would not want to switch to machine $m^1(j)$. The proof of the induction step is similar to the base case and follow from the fact that by the construction of J^k, each job $(j, m^k(j)) \in J^k$ has more priority than job $(j, m^{k'}(j)) \in J^{k'}$ for any $k < k'$. This inductive argument proves that assignment $\mathcal{A}$ is a pure Nash equilibrium, and its makespan is $\log m$. Specifically, machine $j^* \in A^{\log m}$ has makespan $\log m$, since one job from each of $J^1, J^2, \ldots, J^{\log m}$ is scheduled on this machine. This instance shows that for any set of local ordering policies, there is an instance for which the price of anarchy is at least $\Omega(\log m)$. ■

6 Existence of Pure Nash Equilibria

Pure Nash equilibria may not exist for some strategic games, and even if they exist, a sequence of best responses of players may not converge to them. *Potential games* are games for which we can find a *potential function* that maps any state (or any set of strategies) in the game to a number (or a vector) such that after any best response of any player the value of the function strictly decreases (or lexicographically decreases). Potential games possess pure Nash equilibria and any random sequence of best responses of players converge to pure Nash equilibria with probability one.

We can prove the corresponding game of any ordering policy for $B||C_{\max}$ is a potential game and thus, possess pure Nash equilibria, but this is not the case for $R||C_{\max}$ even for two machines. Moreover, we can prove that the game corresponding to the inefficiency-based policy for two machines always possess pure Nash equilibria, but this is not true for any number of machines. Here, we only prove the main result of this section, and leave the rest of them to the appendix.

	1	2	3	4
A	20	∞	∞	∞
B	2	12	∞	1.98
C	4	24	25	3.95
D	5	28	∞	4.9

Table 1: An example without pure Nash equilibria: The processing time of four jobs on four machines.

Theorem 6.1 *The corresponding game to inefficiency-based policy for $R||C_{\max}$ may not possess any pure Nash equilibrium.*

Proof: Consider an instance of $R||C_{\max}$ with 4 machines and 5 jobs A, B, C, D, and T. Job T can only be scheduled on machine 4 and its processing time is 50. The ordering on machine 4 is T, B, C, D, A. The processing times of jobs A, B, C, D on machines $1, 2, 3, 4$ are depicted in Table 1.

As a result, the ordering of jobs in the inefficiency-based policy for machine 1 is (A, B, C, D, T), and for machine 2 is (D, B, C, A, T), and for machine 3 is (C, A, B, D, T). We claim that no pure Nash equilibria exist for this example. We have found this example by solving a mathematical program that captures the inequalities required to prove that no pure Nash equilibrium exists. Here, we give a brief description of why this instance does not have any pure Nash equilibrium. Job T is always scheduled on machine 4 and no other job wants to go to machine 4. We can show a schedule on four machines as a sequence of subsets of jobs in each machine, for example, if jobs A, B, and C are on machine 1, job D is on machine 2, and job T is on machine 4, the corresponding sequence is $(ABC, D, \ , T)$. From this schedule, job C has incentive to switch to machine 3, and the resulting schedule is (AB,D,C,T). This move is shown briefly by $(ABC, D, \ , T) \rightarrow (AB, D, C, T)$. Similarly, $(ABD, \ , C, T) \rightarrow (ABD, C, \ , T) \rightarrow (AD, BC, \ , T) \rightarrow (ACD, B, \ , T) \rightarrow (AD, DB, \ , T) \rightarrow (ABC, D, \ , T) \rightarrow (AB, D, C, T) \rightarrow (ABD, \ , C, T)$. Also $(AD, B, C, T) \rightarrow (ACD, B, \ , T)$. Checking that no other pure Nash equilibrium exists is straightforward. ■

This theorem indicates the need for a coordination mechanism with small price of anarchy for which we can prove convergence to pure Nash equilibria.

7 A Polylogarithmic Upper Bound with Fast Convergence

In Section 4, we designed a scheduling policy for each machine that has a low price of anarchy. However, in Section 6, we proved there may be no (pure) Nash Equilibrium for the jobs and the system may not converge. In this section, we show that we can increase slightly the price of anarchy from $O(\log m)$ to $O(\log^2 m)$, but guarantee existence of Nash Equilibria as well as convergence to pure Nash equilibria.

The algorithm is as follows. Each machine simulates $b = \lceil \log m \rceil$ sub-machines. Sub-machine l for $0 \leq l \leq b-1$ of machine j runs only jobs of inefficiency of at least 2^l and less than 2^{l+1}. Machine j allocates continuously the same time for each of its sub-machines even if there are no jobs to process

on some sub-machines (this requires preemption and idle time). A job assigned to machine j will run on sub-machine l of machine j where $l = \lfloor e_{ij} \rfloor$ given that $e_{ij} < m$. If $e_{ij} \geq m$, the job will be delayed for ever on machine j. To complete the description of the processing strategy, we need to define the order in which each sub-machine processes its jobs. If it is an arbitrary order, we call the family of strategies *Split & Any*. If it is ordered according to ShortestFirst we call it *Split & Shortest*.

Given an instance of the $R||C_{\max}$ problem on m machines, we create a corresponding instance of those jobs to mb sub-machines as follows: if in the original instance job i has processing time p_{ij}, then it would have processing time bp_{ij} on sub-machines l_j of machine j where $l_j = \lfloor e_{ij} \rfloor$ given that $e_{ij} < m$. On all other sub-machines of j (in case $e_{ij} \geq m$ on all sub-machines of j) the processing time is infinite. We start with the following lemma

Lemma 7.1 *Given an instance to the $R||C_{\max}$ problem and its corresponding instance on mb sub-machines.*

1. *Given an assignment for the original instance on the m machines, we can get an assignment for the corresponding instance on the mb sub-machines while increasing the makespan by a factor of at most $2b$. In particular, the optimal makespan increases by a factor of at most $2b$.*
2. *Given an assignment for the corresponding instance on the mb sub-machines, we can get an assignment for the original instance where the completion time of each job remains the same (and in particular the makespan does not increase).*

Now, we can easily prove the following:

Theorem 7.2 *The price of anarchy for $R||C_{\max}$ using* Split & Any *is $O(\log^2 m)$. In particular, the price of anarchy for unrelated machines using* Split & Shortest *is $O(\log^2 m)$.*

Now, we show that our analysis is tight.

Theorem 7.3 *The price of anarchy for $R||C_{\max}$ using* Split & Shortest *is at least $\log^2 m$.*

Finally, we show that this policy converges to a Nash equilibrium very fast.

Theorem 7.4 *The corresponding game for the* Split & Shortest *policy is a potential game. Moreover, any sequence of best responses of players consisting of n rounds of all players converges to a pure Nash equilibrium.*

8 Open Problems

In this paper, we proved that the best achievable price of anarchy by strongly local and local ordering policies are $\Theta(m)$ and $\Theta(\log m)$. Ordering policies characterize all deterministic non-preemptive policies satisfying the IIA property. An interesting open problem is to design preemptive or randomized policies with a constant price of anarchy, or to prove that this is not possible. Another interesting open problem is the speed of convergence to approximate solutions for the inefficiency-based policy [21].

Finally, since pure Nash equilibria for the inefficiency-based policy do not necessarily exist, it would be interesting to bound the approximation ratio of the sink equilibria [15].

Acknowledgements. We thank Allan Borodin for interesting discussions about related work.

References

[1] Aspnes, Y. Azar, A. Fiat, S. Plotkin, and Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *J. ACM 44*, 3, 1997.

[2] B. Awerbuch, Y. Azar, Y. Richter, and Dekel Tsur. Tradeoffs in worst-case equilibria. 2003.

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

[4] A. Bagchi. Stackelberg differential games in economic models. *Springer-Verlag*, 1984.

[5] M. Beckman, C. B. McGuire, and C. B. Winsten. *Studies in the Economics of Transportation*. Yale University Press, 1956.

[6] A. Borodin, M. Nielsen, and C. Rackoff. (incremental) priority algorithms. In *SODA*, pages 752 – 761, 2002.

[7] G. Christodoulou, E. Koutsoupias, and A. Nanavati. Coordination mechanisms. pages 345–357, Turku, Finland, 12–16 July 2004.

[8] R. Cole, Y. Dodis, and T. Roughgarden. How much can taxes help selfish routing? In *EC*, pages 98–107, 2003.

[9] A. Czumaj and B. Vocking. Tight bounds for worst-case equilibria. In *SODA*, pages 413–420, 2002.

[10] E. Davis and J.M. Jaffe. Algorithms for scheduling tasks on unrelated processors. *J. ACM*, 28(4):721–736, 1981.

[11] E. Even-dar, A. Kesselman, and Y. Mansour. Convergence time to nash equilibria. In *ICALP*, pages 502–513, 2003.

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

[13] L. Fleischer, K. Jain, and M. Mahdian. Tolls for heterogeneous selfish users in multicommodity networks and generalized congestion games. In *FOCS*, pages 277–285, 2004.

[14] M. Gairing, T. Lucking, M. Mavronicolas, and B. Monien. Computing nash equilibria for scheduling on restricted parallel links. In *STOC*, pages 613–622, 2004.

[15] M.X. Goemans, V.S. Mirrokni, and A. Vetta. Sink equilibria and convergence. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science(FOCS)*, pages 142–154, 2005.

[16] O.H. Ibarra and C.E. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *J. ACM*, 24(2):280–289, 1977.

[17] N. Immorlica, L. Li, V. Mirrokni, and A. Schulz. Coordination mechanisms for selfish scheduling. In *Workshop of Internet and Economics*, 2005.

[18] Y.A. Korilis, A.A. Lazar, and A. Orda. Achieving network optima using Stackelberg routing strategies. *IEEE/ACM Transactions on Networking*, 5(1):161–173, 1997.

[19] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *STACS*, pages 404–413, 1999.

[20] J. Lenstra, D. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.

[21] V.S. Mirrokni and A. Vetta. Convergence issues in competitive games. In *APPROX*, pages 183–194, 2004.

[22] T. Roughgarden. Stackelberg scheduling strategies. In *STOC*, pages 104–113, 2001.

[23] S. Sahni and Y. Cho. Bounds for list schedules on uniform processors. *Siam J. of Computing*, 9:91–103, 1980.

[24] P. Schuurman and T. Vredeveld. Performance guarantees of local search for multiprocessor scheduling. In *IPCO*, pages 370–382, 2001.

[25] H. von Stackelberg. Marktform und Gleichgewicht. *Springer-Verlag*, 1934. English translation entitled *The Theory of the Market Economy*.

[26] T. Vredeveld. *Combinatorial approximation algorithms. Guaranteed versus experimental performance*. 2002. Ph.D. thesis.

[27] Wikipedia. http://en.wikipedia.org/wiki/Independence_of_irrelevant_alternatives.

Appendix

A Proof of Theorem 3.3

Proof: In order not to deal with the issue of breaking ties (which plays a major role in the general lower bound), we would make all jobs of different size. We construct the following instance. There are $m-1$ types of jobs. For $j=1$ to $m-1$, there are $n_j = 2\frac{(m-1)!}{(j-1)!}$ jobs of type j. Job k for $1 \le k \le n_j$ of type j has processing time $\frac{2}{n_j}(1+\varepsilon_{kj}) = \frac{(j-1)!}{(m-1)!}(1+\varepsilon_{kj})$ on machine j and $\frac{2j}{n_j}(1+\varepsilon_{kj}) = \frac{j!}{(m-1)!}(1+\varepsilon_{kj})$ on machine $j+1$ and infinite (or large enough) on all other machines. We choose $0 < \varepsilon_{kj} < \varepsilon$ for some small enough ε $j+1$ where $\varepsilon_{kj} < \varepsilon_{k+1,j}$ for all k and j and $\varepsilon_{n_j,j} < \varepsilon_{1,j+1}$.

The optimal solution may use the following assignment. Assign all n_j jobs of type j on machine j. This assignment results in completion time of at most $2(1+\varepsilon)$ for each machine (except the m'th one which remains empty).

Consider the following assignment. Half of the jobs of type j are assigned to machine j and half to machine $j+1$ (we later specify which half). Then Machine $j+1$ for $j=1$ to $m-2$ would have a load of slightly more than $(n_j/2)(2j/n_j) = j$ of jobs of type j and slightly more than $(n_j/2)(2/n_j) = 1$ of jobs of type $j+1$. Machine 1 has a load of slightly more than 1 (type 1 jobs) and machine m a load of slightly more than $m-1$ (type $m-1$ jobs).

Note that all jobs on each machine have approximately the same size. Since we set $\varepsilon_{kj} < \varepsilon_{k',j+1}$ for all j and k, k' this implies that jobs of type j are processed before jobs of type $j+1$ (on machine $j+1$).

Finally, we have to specify which set of jobs are actually assigned to each machine. This assignment defines the order of jobs on each machine. Assume for a moment that ε_{kj} would have been 0. This would define a set of completion times for all jobs of type j on machines j and $j+1$. Assign the jobs of type j to the two machines (j and $j+1$) in non-decreasing order of the ID k according to the non-decreasing order of completion time of that jobs. We claim that this assignment is a Nash Equilibrium. Moreover the price of anarchy is about $m/2$.

Immorlica et.al. [17] observed that the set of pure Nash equilibria of the ShortestFirst policy is equivalent to the output of the shortest-first greedy algorithm of Ibarra and Kim [16]. Therefore, the above lower bound implies the lower bound of $\frac{m}{2}$ for the shortest-first greedy algorithm. ■

B Pure Nash equilibria for Special Cases

In this section, we investigate the existence of pure Nash equilibria for general ordering policies and for some special cases. In particular, we prove the following theorems.

Theorem B.1 *The corresponding game of any ordering policy is a potential game for $B||C_{\max}$. Thus, it has pure Nash equilibria for $B||C_{\max}$. Also, if the global ordering for all machines is the same, then pure Nash equilibria exist for the corresponding game of the $R||C_{\max}$. However, for $R||C_{\max}$, there are ordering policies without any pure Nash equilibria even for two machines.*

Proof: Let $w(i,j)$ be the position or *rank* of job i in the global ordering of machine j, i.e., job i is at the $w(i,j)$s position in the global ordering of machine j. Given a schedule S of jobs on all machines, let m_i be the machine of job i and T_i be the starting time of job i. In order to define the potential function for S, we add a dummy job d_j of length ∞ to the end of each machine j. The rank of the dummy job d_j on machine j is $n+1$, i.e., $w(d_j, j) = n+1$, and $m_{d_j} = j$. After adding these dummy jobs, we find the potential function for schedule S as follows: sort the jobs in the non-decreasing order of their starting time, and if there are ties between the starting times, sort them in the non-decreasing order of their ranks $w(i, m_i)$. Since we added a dummy job for each machine, the length of the vector of the potential function is $n+m$. Let the vector of jobs in this order be $(1, 2, \ldots, n+m)$. Therefore, by definition, $T_1 \leq T_2 \leq \ldots \leq T_{n+m}$ and if $T_l = T_{l+1}$, then $w(l, m_l) \leq w(l+1, m_{l+1})$. The potential function for this schedule S is $(w(1, m_1), w(2, m_2), \ldots, w(n+m, m_{n+m}))$. If job k plays his best response and goes to machine m'_k instead of machine m_k, the starting time of job k decreases (since for $B||C_{\max}$ when a job improves its completion time, it improves its starting time as well). As a result, job k occupies an earlier position in the corresponding vector of the new schedule. Job k cannot be the last job on machine m'_k, since each machine has a dummy job who is the last. Let job k' be the job after

k on machine m'_k after k moves (note that k' might be a dummy job). The rank of job k is less than the rank of job k' on machine m'_k. This proves that the potential function decreases lexicographically. Therefore, the game is a potential game.

It is not hard to prove that if the global ordering for all machines is the same, then pure Nash equilibria exist for the corresponding game of the $R||C_{\max}$ and the game is a potential game. If the global ordering on all machines is $(1, 2, \ldots, n)$ and the completion time of job i in schedule S is $C_i(S)$, then the potential function in this case for schedule S is $(C_1(S), C_2(S), \ldots, C_n(S))$.

Finally, for $R||C_{\max}$, there are examples even for two machines for which the corresponding game does not have any pure Nash equilibrium. Consider an example with two machines 1 and 2, and three jobs A, B, C. The global ordering for machine 1 is (A, B, C) and the global ordering for machine 2 is (C, A, B). The processing time of jobs on machines are $p_{A1} = 12$, $p_{B1} = 16$, $p_{C1} = 2$, $p_{A2} = 10$, $p_{B2} = 10$, $p_{C2} = 16$. It is not hard to check that no set of strategies of players is a pure Nash equilibrium in this game. ■

The above theorem shows that an arbitrary set of ordering policies may not have pure Nash equilibria even for two machines. We showed that the corresponding game of the inefficiency-based policy may not possess pure Nash equilibria. The following theorem shows that the inefficiency-based policy always have pure Nash equilibria for two machines.

Theorem B.2 *The inefficiency-based mechanism always possess pure Nash equilibria for two machines.*

Proof: The proof is by induction. The base of induction is for one job for which the proof is trivial. Consider the most inefficient job on both machines and call it A. We do not let A go on the machine for which it is less efficient, say machine 1. The induction is on the number of pairs of jobs and machines (i, j) such that job i can be scheduled on machine j. For the instance for which job A cannot be scheduled on machine 1, we find a pure Nash equilibrium S by induction. For the induction step, we would like to change this equilibrium S to an equilibrium for the original instance. The only possibility is that job A in S wants to switch to machine 1. If we let A move to machine 1, no other job from machine 2 wants to move to machine 1. We claim that jobs from machine 1 do not want to switch to machine 2 either. Note that job A is larger on machine 1 than on machine 2 and hence machine 1 ends in schedule S (without job A) before job A starts on machine 2, otherwise A would not like to move from machine 2. Hence no jobs from machine 1 want to move to machine 2 (although job A left machine 2), since they would finish later if they move. ■

C Proofs of Section 7

Proof of Lemma 7.1. The second part of the lemma is easy. Each machine simulates the b sub-machines continuously and provides $1/b$ of the time for each. Since the processing time of each job in the corresponding instance is b times its original processing time then the completion time of each job remains the same as needed.

Next, we prove the first part of the lemma. Given the an assignment to the original instance we create a feasible assignment to the new instance with increase in makespan by factor of at most $2b$. We

do it in two steps. In the first step we create a new assignment for the original instance where no job i runs on machine j with $e_{ij} \geq m$. This will (at most) double double the makespan. We do it by simply moving each job i that runs on machine j with $e_{ij} \geq m$ to the best machine for that job, i.e., to machine j' where $e_{ij'} = 1$. Let I be the set of such jobs. Clearly the makespan has increased additively by at most $\sum_{i \in I} p_i$ (even if all these jobs were to go on the same machine). However $\sum_{i \in I} p_{ij} \geq \sum_{i \in I} m p_i$. Hence the original makespan was at least

$$\frac{1}{m} \sum_{i \in I} p_{ij} \geq \frac{1}{m} \sum_{i \in I} m p_i = \sum_{i \in I} p_i$$

which means that the makespan at most doubled.

In the second step, we create from the modified assignment an assignment for the sub-machines instance by increasing the makespan by a multiplicative factor of b. This is easily done by assigning job i that is assigned to machine j to the sub-machine $l = \lfloor e_{ij} \rfloor$ of machine j which is feasible and always exists since $e_{ij} < m$. The load of each sub-machine of machine j does not increase since the jobs were split among the sub-machines. However, since the processing time is multiplied by b, the completion time is scaled up by a factor of b. Hence, after applying the two steps the makespan for the corresponding instance is increased by at most $2b$ as required. □

Proof of Theorem 7.2. We can view *Split & Any* for the original instance as processing the jobs on the corresponding instance in 'almost' non-decreasing order of the inefficiency. All jobs on each sub-machine have 'almost' the same (i.e. up to factor of 2) inefficiency. If we change the size of jobs to have precisely the same inefficiency then by using Theorem 5.1 the price of anarchy is at most $O(\log m)$ with respect to the optimal assignment for the corresponding instance (with the original size we lose only additional factor of 2). Nevertheless, the makespan of the optimal assignment for the corresponding instance is at most $O(\log m)$ times the the makespan of the optimal assignment of the original instance. Hence the price of anarchy of *Split & Any* is $O(\log^2 m)$ with respect to its optimum. Since, *Split & Shortest* belongs to the family of *Split & Any* its price of anarchy is not larger. □

Proof of Theorem 7.3. We use again a variation on the standard example from Theorem 5.1 to show that even for the restricted assignment model ($B||C_{\max}$) the price of anarchy of this strategy is at least $\log^2 m$. Note again that for $B||C_{\max}$, the inefficiency of every job is precisely 1 on any legal machine for that job. Hence, only the first sub-machine of each machine is doing any work. We use the example from Theorem 5.1 but we slightly perturb the job sizes. All jobs are of processing time slightly smaller than 1 where all jobs in class k are slightly shorter than all jobs in class $k+1$. Hence the algorithm may order the jobs on each machine (on the first sub-machine) according to classes and hence we get a similar (up to a small perturbation) example as in Theorem 5.1. Since only one sub-machine is active, the makespan of the example described is multiplied by $\log m$ and becomes $\log^2 m$ where the optimum remains the same i.e., 1. □

Proof of Theorem 7.4. The completion time of each job in *Split & Shortest* is precisely equal to the completion time of each job in the corresponding instance on the mb sub-machines. That instance is ShortestFirst on each sub-machine. Hence, any sequential improvement process converges to a Nash equilibrium [11, 17]. A potential function for the ShortestFirst policy is the vector of the completion time (sorted in non-decreasing order) of all jobs which decreases lexicographically after each best

response. Also it is proved in [17] that at most n rounds of best responses of players converges to pure Nash equilibria in this game. $\square$