Scheduling Tasks from Selfish Multi-tasks Agents

Johanne Cohen[1] and Fanny Pascual[2](✉)

[1] LRI-CNRS, Université Paris Sud, bât 650, 91400 Orsay, France
johanne.cohen@lri.fr

[2] Sorbonne Universités, Université Pierre et Marie Curie (Univ. Paris 06), CNRS, LIP6 UMR 7606, 4 place Jussieu, 75005 Paris, France
fanny.pascual@lip6.fr

Abstract. We are interested in scheduling tasks from several selfish agents on a set of parallel identical machines. A coordination mechanism consists in giving a scheduling policy to each machine. Given these policies, each agent chooses the machines on which she assigns her tasks, and her aim is to minimize the average completion times of her tasks. The aim of the system (social cost) is to minimize the average completion time of all the tasks. We focus on coordination mechanisms inducing Nash equilibria, and on the performance of such mechanisms. When the machines do not know the owners of the tasks, the classical coordination mecanisms used for single-task agents do not work anymore and we give necessary conditions to obtain coordination mechanisms that induce Nash equilibria. When each machine is able to know the owner of each task it has to schedule, we give coordination mechanisms which always induce Nash equilibria.

1 Introduction

Among the most fundamental problems in algorithmic game theory are scheduling and load balancing problems. Since the seminal paper by Koutsoupias and Papadimitriou [16], these problems have been of growing interest [21]. Indeed, besides their conceptual simplicity, these problems are central in distributed environments where some machines are shared between selfish users, and where the users decide on which machines they will assign their tasks. In such environments, coordination mechanisms have been introduced by Christodoulou *et al.* [7] in order to obtain socially desirable solutions despite the selfishness of the agents. A *coordination mechanism* is a set of scheduling policies, one for each machine. A scheduling mechanism for a machine M_i takes as input a set of tasks assigned to machine M_i along with their processing times. The output is a schedule of the tasks on M_i. The aim is to design a coordination mechanism such that for each instance (set of tasks) there exists a Nash equilibrium (a schedule where no agent has incentive to change the assignement of her tasks).

When a coordination mechanism always induces Nash equilibria, it is useful to measure the quality of the Nash equilibria induced, which is usually done

J.L. Träff et al. (Eds.): Euro-Par 2015, LNCS 9233, pp. 183–195, 2015.
DOI: 10.1007/978-3-662-48096-0_15

using the price of anarchy [16]. The *price of anarchy* is defined as the maximal value, over all the instances, of the ratio between the social cost in the worst Nash and the social cost in an optimal solution.

Starting from the seminal paper of Christodoulou *et al.* [7], coordination mechanisms have been extensively studied for single tasks agents [3,4,6,9,11,13–15]. In these papers, each agent owns *a single* task, and her aim is to minimize the completion time of her task. The social cost is either the largest completion time of a task or the average completion time of the tasks. The coordination mechanism studied are often the ones which schedule the tasks in order of non decreasing lengths (ShortestFirst policy), in order of non increasing lengths (LongestFirst policy), or in a random order; for identical machines [7,14], related machines [13], or unrelated machines [3,4,9,11]. These coordination mechanisms usually induce pure Nash equilibria, and the aim is to measure their price of anarchy.

In our setting, each agent may own *several* tasks, and her aim is to minimize the average completion time of her tasks. We study the existence and the quality of coordination mechanisms for this extension of this classical game. The social cost that we consider is the sum of the completion times of all the tasks.

Most of the papers dealing with multi-task selfish agents sharing machines are interested by designing centralized fair solutions (see [2] for a recent survey). In these models, the agents cannot choose themself the machines on which their tasks will be scheduled. Starting from the seminal paper [20], some papers (e.g. [5,8,12]) consider a set of agents owing each one a set of tasks but also a set of machines. The aim is to design a centralized algorithm which assigns all the tasks to all the machines in a way which minimizes the overall makespan whilst ensuring that the cost of each agent is not increased compared to the solution where each agent schedules her own tasks on her own machines.

There is, up to our knowledge, only one paper which deals with coordination mechanisms with multi-tasks agents. In this paper, Abed et al. [1] consider that each agent owns several tasks, each task having a length and a weight. The machines are unrelated, and each agent aims at minimizing the weighted completion time of her tasks, whereas the social cost is the sum of agents' costs. The main difference to our paper is that the authors do not consider Nash equilibria but a superclass of Nash equilibria: they consider that a schedule is stable (they call such a schedule a *weak Nash equilibrium*) if no agent may decreases her cost by moving *exactly one* of her task to a different machine. They show that when the policies of the machines order the tasks according to their length to weight ratio, then there exists a weak Nash equilibrium, and that the price of anarchy (with respect to weak Nash equilibrium) is 4. They extend this policy by introducing some delays between tasks, and they show that the price of anarchy of this new coordination mecanism is about 2.6.

We now describe precisely the problem studied and the notions used in this paper.

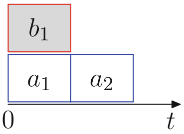
Model. We consider a set of K selfish agents $\{A_1, \ldots, A_K\}$, each agent A_i owning a set of n_i tasks. When we only consider two agents, these agents will be called A and B; the set of tasks of agent A will be $\{a_1, a_2, \ldots, a_{n_A}\}$, and the set of tasks of agent B will be $\{b_1, b_2, \ldots, b_{n_B}\}$. Each task has a unique identification number and an arbitrary processing time (length). It cannot be preempted. The agents share a set of $m \geq 2$ identical parallel machines $\{M_1, \ldots, M_m\}$. Each machine M_i has a public *policy*, which is an algorithm which returns a schedule (on M_i) of the tasks assigned to M_i. This policy may introduce idle times between the tasks. However, since we consider a totally decentralized setting, the policy of M_i depends only on the tasks assigned to M_i: it cannot be a function of the tasks assigned to the other machines. A set of policies, one for each machine, is called a *coordination mechanism*. We consider two models. In the first one, the machines cannot distinguish the tasks of one agent from the tasks of another agent: a machine is only aware of the length and identification number of the tasks it has to schedule. In the second one, the machines know the owner of each task.

Knowing the policies of the machines, the set of the tasks of the other agents and the strategies of the other agents, each agent chooses, for each of her tasks, on which machine it will be scheduled. The *strategy* of each agent is thus an assignment to a machine of each of her tasks. The aim of each agent is to minimize the average completion time of her tasks. This is equivalent to minimize the sum of completion times of her tasks: in the sequel the cost of each agent is thus the sum of the completion times of her tasks. A schedule is a (pure) *Nash equilibrium* if no agent can decrease the sum of completion times of her tasks by changing her assignment. In this paper, we focus on coordination mechanisms which always induce pure Nash equilibria (i.e., coordination mechanism such that, for each instance, there exists at least one pure Nash equilibrium). A game always has a mixed Nash equilibrium [19], but pure Nash equilibria are more natural and are the only possible solutions in some settings.

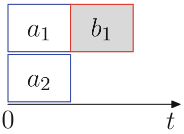
Our Contribution. In Sect. 2, we consider that the machines do not know the owners of the tasks. We show that if all the machines use the same deterministic policy then this policy necessarily have to introduce some idle times between the tasks in order to induce Nash equilibria. Moreover the price of anarchy of such a coordination mechanism is at least 2. In Sect. 3, we show that there exists coordination mechanisms which induce Nash equilibria when the machines are able to know the owner of each task. In particular, we introduce a simple and fair coordination mechanism which has a bounded price of anarchy if the number of agents is small. We conclude this paper in Sect. 4.

2 Properties of Coordination Mechanisms in Which the Machines Do Not Know the Owners of Their Tasks

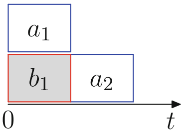
We consider in this section that the machines are not able to detect the owner of the tasks they have to schedule. We will focus on coordination mechanisms with deterministic identical policies. Given two tasks i and j, we note $i \prec j$ if and only if task i is scheduled before task j when a machine has only these two tasks to schedule.



(a) $C_A = 3$ and $C_B = 1$



(b) $C_A = 2$ and $C_B = 2$



(c) $C_A = 3$ and $C_B = 1$

Fig. 1. Instance with no pure Nash equilibrium when all the machines have the same deterministic policy without idle times. C_A (resp. C_B) is the cost of agent A (resp. B).

Proposition 1. *If all the machines have the same deterministic policy, and if this policy does not introduce idle times between the tasks, then the coordination mechanism does not always induce a pure Nash equilibrium.*

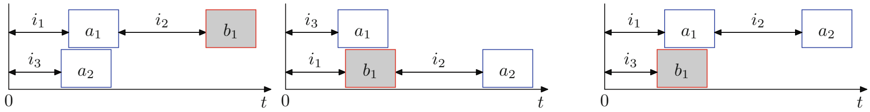
Proof. We provide a instance without pure Nash equilibrium. This instance, depicted in Fig. 1, consists in two machines and two agents A and B. Agent A has two tasks a_1 and a_2, each of length 1, while B has one task b_1 of length 1. We consider tasks a_1, a_2, and b_1 such that $a_1 \prec b_1$ and $b_1 \prec a_2$. Note that given three tasks i, j, k, and any deterministic policy, there always exists a permutation of the tasks such that $i \prec j$ and $j \prec k$. The configuration which consists of three tasks on the same machine is not a Nash equilibrium since b_1 would have incentive to move on the idle machine. The other configurations are represented in Fig. 1 and are also not Nash equilibria: in Fig. 1(a) Agent A can decrease her cost by assigning task a_1 to M_1; in Fig. 1(b) Agent B has incentive to move her task; in Fig. 1(c) Agent A has incentive to exchange the assignment of her two tasks a_1 and a_2. □

Note that the classical policies LongestFirst and ShortestFirst have this property, and thus they do not always induce pure Nash equilibria (contrary to the case where each agent has only one task [14]). Moreover, the move of only two tasks is needed to show this result. Abed et al. [1] show that when multi-tasks agents are able to move only one task to improve their cost, then the ShortestFirst policy is stable (for each instance there exist a schedule where the agents cannot improve their costs by moving at most one of their tasks). If the agents are able to move at most two tasks to compute their best response, then Proposition 1 shows that there exists instances without stable schedules.

Note also that this result does not depend on the social cost considered, and is thus valid for any social cost.

Proposition 2. *Consider a coordination mechanism in which all the machines have the same deterministic policy which is not based on identification numbers*[1]*. If this coordination mechanism always induces a pure Nash equilibrium, then its price of anarchy is larger than or equal to* 2.

[1] The schedule is constructed by considering only the lengths of the tasks to schedule. Identification numbers are used thereafter to break the ties only, i.e. to assign each task to a slot of its length in the constructed schedule.



(a) not a Nash eq. (b) not a Nash eq. (c) Nash eq. when $i_3 \geq i_1 + i_2 + 1$

Fig. 2. Different configurations where both machines have the same deterministic policy having idle times.

Proof. Let us consider the following instance, with two machines and two agents: Agent A owns two tasks a_1 and a_2, and Agent B owns one task b_1. We consider that these three tasks are all of length one, and are such that $a_1 \prec b_1$ and $b_1 \prec a_2$. Let i_1 (resp. i_2) be the length of the idle time before the first (resp. second) task when a machine schedules two tasks of length 1. Let i_3 be the length of the idle time before the first task when a machine schedules one task of length 1. We proceed by cases analysis. There are four possible schedules. We show that if one of these schedules is a Nash equilibrium then the price of anarchy is at least 2.

- Schedule 1: *Tasks a_1 and b_1 are on the same machine (w.l.o.g. on M_1); task a_2 is alone on M_2*. Figure 2(a) shows this configuration. The completion time of b_1 is i_1+i_2+2. If this task would jump on M_2, then its completion time would be i_1+1: this task has incentive to change machine (because $i_1+i_2+2 > i_1+1$). Thus, this schedule is not a Nash equilibrium.
- Schedule 2: *Tasks b_1 and a_2 are on the same machine (w.l.o.g. on M_2); task a_1 is alone (on M_1).* This configuration is depicted in Fig. 2(b). The completion time of all the tasks of Agent A is $i_1 + i_2 + i_3 + 3$. If Agent A moves task a_2 on M_1 and a_1 on M_2, then the sum of completion times of her tasks will be $i_1 + i_3 + 2$. Since $i_1 + i_2 + i_3 + 3 > i_1 + i_3 + 2$, Agent A has incentive to move her tasks, and this schedule is thus not a Nash equilibrium.
- Schedule 3: *Tasks a_1 and a_2 are on the same machine (w.l.o.g. on M_1); task b_1 is alone (on M_2).* Figure 2(c) shows this configuration. Let us focus on Agent A. The completion time of the tasks of Agent A is $2i_1 + i_2 + 3$ in this schedule. If Agent A would place task a_1 on M_2 and task a_2 on M_1, then the sum of completion times of her tasks would be $i_3 + i_1 + 2$. Thus this schedule is a Nash equilibrium only if $2i_1 + i_2 + 3 \leq i_1 + i_3 + 2$, i.e. if $i_3 \geq i_1 + i_2 + 1$. Thus $i_3 \geq 1$ is a necessary condition for schedule 3 to be a Nash equilibrium. Let us thus consider any policy where $i_3 \geq 1$, and let us consider an instance which consists of only one task of length 1. The completion time of this task is at least 2, whereas the optimal completion time would be 1. Therefore the price of anarchy of a coordination mechanism using such a policy is at least 2.
- Schedule 4: *The three tasks are on the same machine (w.l.o.g. on M_1).* Let us denote this schedule by $\mathcal{S}$. Let us consider that $\mathcal{S}$ is a Nash equilibrium, and that the price of anarchy of the coordination mechanism is smaller than 2. This implies that $i_3 < 1$, otherwise an instance with only one task of length 1 would have a sum of completion times larger than 2, whereas the optimum is 1. Thus task b_1 has to be scheduled first in $\mathcal{S}$, otherwise its completion

time would be at least 2 and this task would decrease its completion time by jumping on the idle machine: $\mathcal{S}$ would not be a Nash equilibrium. Tasks a_1 and a_2 are thus on second and third positions. Since $\mathcal{S}$ is a Nash equilibrium, Agent A has no incentive to move a_2 on M_2. By moving a_2, this agent would let a_1 and b_1 on one machine (a_1 is scheduled first since $a_1 \prec b_1$), and a_2 alone on the other machine. Let us denote by $C_i^{(j)}$ the completion time of the i^{th} task when there are j tasks of length 1 on a machine. Thus we have:

$$C_2^{(3)} + C_3^{(3)} \leq C_1^{(1)} + C_1^{(2)} \tag{1}$$

We saw that $i_3 < 1$, so $C_1^{(1)} < 2$. Moreover, since $C_2^{(3)} \geq 2$, we get: $C_3^{(3)} < C_1^{(2)}$. We now show that with these hypothesis on the policies, there is an instance in which there is no Nash equilibrium.

Let us consider the following instance: three tasks of length 1: a_1', a_2' (belonging to Agent A), and b_1' (belonging to Agent B), such that, when they are together on one machine a_1' is scheduled first. The schedule where the three tasks are together is not a Nash equilibrium, since b_1' has a completion time larger than or equal to 2, whereas it would get a completion time smaller than 2 by going on the other machine. The schedule where b_1' is alone on a machine is also not a Nash equilibrium. Indeed, in this schedule the sum of completion times of a_1' and a_2' is $C_1^{(2)} + C_2^{(2)} \geq 2\,C_1^{(2)} > 2\,C_3^{(3)}$, whereas by going with b_1', tasks a_1' and a_2' would have a sum of completion times smaller than $2\,C_3^{(3)}$: Agent A has incentive to move her tasks. The last possible configuration is when b_1' is with one task of A, the other task of A being on the other machine. In this case the sum of the completion times of the tasks of A is larger than or equal to $C_1^{(1)} + C_1^{(2)} \geq C_2^{(3)} + C_3^{(3)}$ by Eq. 1. By going with b_1', the sum of completion times of A's tasks would be at most $C_1^{(3)} + C_3^{(3)} < C_2^{(3)} + C_3^{(3)}$: these tasks again have incentive to move. Therefore there is no Nash equilibrium in this instance, if we assume that the price of anarchy of the coordination mechanism is smaller than 2. □

We studied the case where the owners of the tasks are not known by the machines: the results are rather negative since we gave strong necessary conditions to get coordination mechanisms which always induce Nash equilibria. Let us now show that the results are more positive if the machines are able to know the owners of the tasks.

3 Coordination Mechanisms in Which the Machines Know the Owners of Their Tasks

If the identification numbers (IDs) of the owners of the tasks are used only to break the ties between the tasks of the same length, then we can extend Proposition 1: in this case, if all the machines have the same deterministic policy, and if this policy does not introduce idle times between the tasks, then the coordination mechanism does not always induce a pure Nash equilibrium[2]. Thus the coordination mechanism which considers the tasks with the ShortestFirst policy and breaks the ties with the IDs of the agents does not always induce Nash equilibria. Since the IDs of the agents should not be considered only to break the ties, let us now consider coordination mechanisms which make a more intensive use of these IDs.

Let us first introduce a simple coordination mechanism, called PrioSPT: each machine schedules the tasks of the same agent together, considering the agents by increasing order of their ID. In other words, each machine schedules the tasks of Agent A_1, and then the tasks of Agent A_2, and so forth. The tasks of a same agent are scheduled with the ShortestFirst policy. This coordination mechanism induces a Nash equilibrium, since each agent A_i has assigned her tasks in order to minimize her cost given the tasks of higher priority agents, and the tasks of lower priority agents will be scheduled after the tasks of A_i and thus will not change the cost of A_i. Note that this coordination mechanism induces Nash equilibria which can be reached in a polynomial time. Indeed, it has been shown [17,18] that the SPT list algorithm[3] is optimal for the minimization of the sum of the completion times, even if some machines are not available at time 0. Each agent will thus use this polynomial time algorithm to schedule her tasks, given the schedule obtained with the tasks of the higher priority agents.

However, this coordination has two main drawbacks: it is unfair (the lower is the ID of an agent, the higher is her priority), and its price of anarchy is unbounded: consider for example an instance where Agent A_1 has m very large tasks, and Agent A_2 has a lot of tiny tasks. Let us now introduce a new coordination mechanism which is fair with the agents and which has a bounded price of anarchy. This coordination mechanism, that we call EqualPrioSPT, works if the number of agents is known and smaller than or equal to the number of machines, which is realistic in many situations, like the one studies in [12], where a few organizations (universities, associations, etc.) share a set of machines.

The idea of EqualPrioSPT is the following one: for each agent A_i, there are $\lfloor \frac{m}{K} \rfloor$ (or $\lfloor \frac{m}{K} \rfloor + 1$) machines on which the tasks of A_i are scheduled first (from the smallest one to the largest one). On these machines, once the tasks of A_i have been scheduled, the tasks of $A_{1+(i \bmod K)}$ are scheduled, from the smallest one to the largest one, and then the tasks of $A_{1+((i+1) \mod K)}$, etc. The latest tasks to be scheduled are the tasks of A_{i-1} (or A_K if $i = 1$).

More formally, to each agent $A_i \in \{A_1, \ldots, A_K\}$, we associate a priority list $L_i = (A_{1+(i \bmod K)}, A_{1+((i+1) \bmod K)}, \ldots, A_{1+((i+K-2) \bmod K)})$ (e.g. the priority list of A_3 is $(A_4, A_5, A_6, A_1, A_2)$ when there are 6 agents). Let q and r be the

[2] The proof is the same as the one of Proposition 1, except that the three considered tasks i, j and k are not of length 1 but of length $1-\varepsilon$, 1 and $1+\varepsilon$ for a small value of ε. For any deterministic policy, there always exists a permutation of the tasks such that $i \prec j$ and $j \prec k$. Tasks i and k are the ones of Agent A, and task j is the one of Agent B.

[3] The *SPT list algorithm* considers the tasks in non-decreasing order of their lengths, and assigns each task to a machine, as soon as a machine is available (idle).

two positive integers such that $m = qK + r$. For $0 \leq i \leq K-1$, machine M_{iq+1} to machine $M_{(i+1)q}$ schedule the tasks of agent A_{i+1} first (using the ShortestFirst policy). If $r \neq 0$, then for $1 \leq i \leq r$ machine M_{Kq+i} schedules the tasks of agent A_i first (using the ShortestFirst policy). Let M_j be one of the machines which schedule first the tasks of A_i. Once M_j has scheduled the tasks of agent A_i, it schedules the tasks of the other agents in the order of the priority list L_i. The tasks belonging to a same agent are scheduled with the ShortestFirst policy.

Proposition 3. EqualPrioSPT *induces a pure Nash equilibrium, and this equilibrium can be reached in* $O(nK)$*, where* $n = \sum_{i=1}^{K} n_i$ *is the number of tasks.*

Proof. We give a constructive proof: we provide a polynomial time algorithm which takes as input an instance of the game (m machines and a set of tasks belonging to K agents), and which returns a Nash equilibrium of this instance. In this algorithm, we say that an agent is *fixed* or not (once an agent is fixed, her tasks won't be moved anymore). We will also say that each agent *owns*, at each step of the algorithm, a set of machines. This algorithm is the following one:

- No agent is fixed. For each agent $A_j \in \{A_1, \ldots, A_K\}$, the machines owned by A_j are the ones on which A_j has the highest priority. Each agent A_j schedules her tasks using the SPT list algorithm on the machines she owns.
- For i from 1 to K:
  - For each agent $A_j \in \{A_1, \ldots, A_K\}$, let D_i^j be the smallest date at which a machine is idle among the machines owned by A_j. Let $D_i = \min_{j \in \{1,\ldots,K\}} \{D_i^j\}$. Let A_{x_i} be an agent such that $D_i^j = D_i$.
  - Agent A_{x_i} is now *fixed* (and will remain fixed in the sequel).
  - Let A_{y_i} be the first agent, among the agents which are not fixed, in the priority list L_{x_i}. Add to the set of machines owned by A_{y_i} the machines previously owned by A_{x_i}. Remove from the schedule all the tasks of A_{y_i} which are started after time D_i, and schedule them again using the SPT list algorithm on the machines that A_{y_i} currently owns (on these machines, the tasks starting before D_i are not moved - note that it includes all the tasks of the agents other than A_{y_i}).

At each step (iteration) one agent is fixed (her tasks won't move anymore) and the only tasks which are moved are the one of a single agent A_{y_i}: the SPT list algorithm used to schedule them takes time $O(n_{y_i}) \subset O(n)$ (once the tasks have been sorted for each agent - which takes time $O(n \log n)$). There are K steps so this algorithm runs in $O(nK)$. Let us now prove the following property: *at the end of each iteration* i *of this algorithm, the agents which are fixed do not have incentive to move their tasks.* The proof is by induction on i.

- This is true when $i = 1$: all the tasks of the only fixed agent, A_{x_1}, start at the latest at time D_i, whereas the first idle time on a machine is D_i. Moreover, A_{x_1} used the SPT list algorithm to schedule her tasks on the machines she owns: this minimizes her sum of completion times.

– Let $i > 1$. Let us now consider that the property is true for each iteration $j < i$, and let us show that it is also true for iteration i. Agent A_{x_i}, which has been fixed at iteration i, has not incentive to move her tasks since all her tasks starts at the latest at time D_i, whereas the first idle time on a machine is D_i. Before this date, all the machines which are not owned by A_{x_i} schedule tasks which have a higher priority than A_{x_i} (otherwise by construction, some tasks of A_{x_i} would have been scheduled instead of the tasks of a lower priority agent). Furthermore, the tasks of A_{x_i} have been scheduled with the SPT list algorithm: this minimizes the cost of A_{x_i}. Likewise, each agent A_{x_j} fixed at a given iteration $j < i$ has not incentive to move her tasks. Indeed, by induction, she had no incentive to move her tasks at the time at which she has been fixed, D_j, and, by construction, the schedule of the tasks scheduled before time D_j does not change after this time.

We have proved that the agents which are fixed do not have incentive to move their tasks once they are fixed. Since at the end of the execution of the algorithm all the agents are fixed, no agent has incentive to move her tasks, and the schedule obtained is thus a Nash equilibrium. □

Let us now show that, contrarily to the coordination mechanism PrioSPT, the price of anarchy of EqualPrioSPT is bounded.

Lemma 1. *Let q and m be two positive integers such that $q < m$. The sum of the completion times of a set of tasks scheduled with the SPT list algorithm on q machines is smaller than or equal to $\frac{m}{q}$ times the sum of completion times of the same tasks scheduled with the SPT list algorithm on m machines.*

Proof. An $OPT_{\sum}$ schedule is a schedule in which the sum of completion times of the tasks is minimized. A schedule obtained by executing the SPT list algorithm (we will call such a schedule a SPT schedule) is thus an $OPT_{\sum}$ schedule. Conway *et al.* [10] show that an $OPT_{\sum}$ schedule of x tasks on m machines can be described as follows. W.l.o.g., we assume that $\ell_1 \geq \ell_2 \geq \cdots \geq \ell_x$, where ℓ_i is the length of task i. We define the following sets: $\pi_1 = \{\ell_1, \ell_2, \ldots, \ell_m\}$, $\pi_2 = \{\ell_{m+1}, \ell_{m+2}, \ldots, \ell_{2m}\}$, $\ldots$, $\pi_k = \{\ell_{(k-1)m}, \ldots, \ell_x\}$, where $k = \lceil \frac{x}{m} \rceil$.

The set π_i is called the i^{th} rank of the tasks. A $OPT_{\sum}$ schedule is a schedule obtained by scheduling the tasks rank by rank, in the order $\pi_k, \pi_{k-1}, \ldots, \pi_1$: the tasks of π_k are scheduled first, each one on a different machine, and the tasks of π_{k-1} are scheduled, also each one on a different machine, and so forth.

By this way, a task in π_i will be followed by $i - 1$ tasks on its machine, and thus it will be counted i times in the sum of the completion times of the tasks: this sum is $\sum_{j=1}^{x} C_j = \sum_{i=1}^{k} \sum_{j \in \pi_i} i\ell_j$.

Let us assume without loss of generality that the number of tasks x is divisible by the number of machines m. If it is not the case, then we can add dummy tasks of length 0. If there are m machines, then task ℓ_i will be in the set $\pi_{\lceil \frac{i}{m} \rceil}$ and thus it will be counted $\lceil \frac{i}{m} \rceil$ times in the sum of the completion times.

Let r_i be the rank of task ℓ_i in a SPT schedule for q machines. For $1 \leq i \leq x$, we have $r_i = \lceil \frac{i}{q} \rceil$, and thus $r_1 \leq r_2 \leq \cdots \leq r_x$.

We will focus on the tasks of rank j in the SPT schedule on m machines. In other words, we will focus on set $\pi_j = \{\ell_{(j-1)m+1}, \ldots, \ell_{jm}\}$. We will prove that

$$\sum_{i=(j-1)m+1}^{jm} r_i \ell_i \leq \frac{m}{q} \sum_{i=(j-1)m+1}^{jm} j\ell_i \tag{2}$$

By definition, we have $r_{jm} = \lceil \frac{jm}{q} \rceil$. We can notice that if $r_{jm} = \frac{jm}{q}$, then Eq. (2) holds. Now, we assume that $jm = q(r_{jm} - 1) + \alpha$ where $q > \alpha > 1$. First, there are α tasks of this rank in π_j. So, we have

$$\sum_{i=jm+1-\alpha}^{jm} r_i \ell_i = r_{jm} \sum_{i=jm+1-\alpha}^{jm} \ell_i = \left(\frac{jm}{q} + \frac{q-\alpha}{q} \right) \sum_{i=jm+1-\alpha}^{jm} \ell_i \tag{3}$$

Second, there are $m - \alpha$ tasks of rank at most $r_{jm} - 1$.

$$\sum_{i=(j-1)m+1}^{jm-\alpha} r_i \ell_i \leq (r_{jm} - 1) \sum_{i=(j-1)m+1}^{jm-\alpha} \ell_i = \left(\frac{jm-\alpha}{q} \right) \sum_{i=(j-1)m+1}^{jm-\alpha} \ell_i \tag{4}$$

Third, we will find an upper bound of the following value $X = \sum_{i=jm+1-\alpha}^{jm} (q - \alpha)\ell_i - \sum_{i=(j-1)m+1}^{jm-\alpha} \alpha\ell_i$. Since $\ell_1 \geq \ell_2 \geq \cdots \geq \ell_x$, we get $\alpha(q-\alpha)\ell_{jm+1-\alpha} \geq (q-\alpha)\sum_{i=jm+1-\alpha}^{jm} \ell_i$ and $\sum_{i=(j-1)m+1}^{jm-\alpha} \alpha\ell_i \geq (m-\alpha)\alpha\ell_{jm+1-\alpha}$. By computation, we obtain $X \leq \alpha(q-m)\ell_{jm+1-\alpha}$. Since $q < m$, we get $X < 0$. From Eqs. (3) and (4), we obtain

$$\sum_{i=(j-1)m+1}^{jm} r_i \ell_i \leq \left(\frac{m}{q} \sum_{i=(j-1)m+1}^{jm} j\ell_i \right) \tag{5}$$

Thus we have: $\sum_{j=1}^{k} \sum_{i \in \pi_j} j\ell_i \leq \frac{m}{q} \sum_{j=1}^{k} \sum_{i \in \pi_j} r_i \ell_i$. Hence the sum of completion times of the tasks scheduled on q machines is at most $\frac{m}{q}$ times larger than the sum of completion times of these tasks scheduled on m machines. □

Proposition 4. *The price of anarchy of* EqualPrioSPT *is at most* $\frac{m}{\lfloor m/K \rfloor}$. *This bound is asymptotically tight.*

Proof. The proof is split into two parts. The first part gives an upper bound on the price of anarchy by finding a relationship between the sum of the completion times in a schedule induced by the EqualPrioSPT coordination mechanism and the sum of the completion times in an optimal schedule, obtained by using the SPT list algorithm. The second part provides a lower bound on the price of anarchy by giving an example. Let q and r be two integers such that $m = qK + r$.

First, we consider the schedule obtained when the tasks of each agent A_i (with $i \in \{1, \ldots, K\}$), are scheduled using the SPT list algorithm on the machines where A_i has the highest priority (there are q or $q + 1$ such machines). Let us denote this schedule by $\mathcal{S}$. In $\mathcal{S}$, the cost of each agent is larger than or equal to her cost in a Nash equilibrium (otherwise an agent would schedule her tasks with the SPT list algorithm on the machines where she has the highest priority and she would decreases her cost). Let us now show that the cost of the sum of completion times in $\mathcal{S}$ is at most $\frac{m}{\lfloor m/K \rfloor} OPT$, where OPT is the optimal sum of completion times. The sum of the completion times of a set of tasks scheduled with the SPT list algorithm on q machines is smaller than or equal to $\frac{m}{q}$ times the sum of completion times of the same tasks scheduled with the SPT list algorithm on m machines (Lemma 1). The SPT list algorithm minimizes the sum of completion times. Thus, the cost of Agent A_i in $\mathcal{S}$ is smaller than or equal to $\frac{m}{q}$ times its cost in any solution (including the optimal solution): the sum of the completion times in $\mathcal{S}$ is thus smaller than or equal to $\frac{m}{q} OPT$. Therefore, the price of anarchy of EqualPrioSPT is at most $\frac{m}{\lfloor m/K \rfloor}$ because $q = \lfloor \frac{m}{K} \rfloor$.

Let us now prove the lower bound by providing a particular instance: there are K agents and $m = K(2K+4)$ machines. Thus $q = 2K + 4$. Agent A_1 has $qm\alpha$ tasks of length 1 and q tasks of length $m\alpha$ where α is an arbitrary integer larger than 1. For $2 \leq i \leq K$, Agent A_i has q tasks of length equal to $m\alpha$. By computation we get that the price of anarchy is at least $\frac{\alpha m}{(\alpha+1)q}$, which for large values of α tends towards to $\frac{m}{q}$. □

4 Conclusion and Future Work

We studied the existence of coordination mechanism for multi-tasks agents. Classical deterministic policies do not always induce pure Nash equilibria in this context. In order to get Nash equilibria, if the machines are not able to identify the owners of the tasks, then we have either to use non deterministic policies (but such policies may be not easy to use in practice); or different policies on the machines (but this may also not be very practical since it may not be easy to add a machine to the system whilst ensuring that the coordination mechanism still induce Nash equilibria); or we should use policies which introduce idle times between the tasks (in this case the price of anarchy is at least 2).

Thus, knowing the owner of each task in the case of multi-tasks agents is a very useful information. In this case there exists coordination mechanisms inducing Nash equilibria. In particular, we have introduced a very simple coordination mechanism which may be used when the number of agents is known and small compared to the number of machines: this mechanism is fair since all the agents are treated equitably, and its price of anarchy is about K (this corresponds to the best we may have for $K = 2$ agents in the case of deterministic identical policies when the owner of the tasks are not known). Note that Lemma 1, introduced to show this result, can also be useful in other contexts: it indeed allows to bound the deterioration of the sum of completion times of a set of tasks when the number of machines to schedule these tasks decreases.

This work is a first step towards the study of coordination mechanism with agents owning several tasks. The main remaining open problem consists in determining whether there exists a coordination mechanism which always induce Nash equilibria with multi-tasks agents when the machines do not know the owners of the tasks.

Acknowledgments. The work of the second author was supported by ANR grant ANR-14-CE24-0007 (project COCORICO).

References

1. Abed, F., Correa, J.R., Huang, C.-C.: Optimal coordination mechanisms for multi-job scheduling games. In: Schulz, A.S., Wagner, D. (eds.) ESA 2014. LNCS, vol. 8737, pp. 13–24. Springer, Heidelberg (2014)
2. Agnetis, A., Billaut, J.C., Gawiejnowicz, S., Pacciarelli, D., Soukhal, A.: Multiagent Scheduling - Models and Algorithms. Springer, Heidelberg (2014)
3. Azar, Y., Jain, K., Mirrokni, V.: (Almost) optimal coordination mechanisms for unrelated machine scheduling. In: SODA 2008, pp. 323–332. SIAM (2008)
4. Caragiannis, I.: Efficient coordination mechanisms for unrelated machine scheduling. Algorithmica **66**(3), 512–540 (2013)
5. Chakravorty, A., Gupta, N., Lawaria, N., Kumar, P., Sabharwal,Y.: Algorithms for the relaxed multiple-organization multiple-machine scheduling problem. In: HiPC 2013, pp. 30–38 (2013)
6. Christodoulou, G., Gourvès, L., Pascual, F.: Scheduling selfish tasks: about the performance of truthful algorithms. In: Lin, G. (ed.) COCOON 2007. LNCS, vol. 4598, pp. 187–197. Springer, Heidelberg (2007)
7. Christodoulou, G., Koutsoupias, E., Nanavati, A.: Coordination mechanisms. Theor. Comput. Sci. **410**(36), 3327–3336 (2009)
8. Cohen, J., Cordeiro, D., Trystram, D., Wagner, F.: Analysis of multi-organization scheduling algorithms. In: D'Ambra, P., Guarracino, M., Talia, D. (eds.) Euro-Par 2010, Part II. LNCS, vol. 6272, pp. 367–379. Springer, Heidelberg (2010)
9. Cole, R., Correa, J.R., Gkatzelis, V., Mirrokni, V., Olver, N.: Inner product spaces for MinSum coordination mechanisms. In: STOC 2011, pp. 539–548. ACM (2011)
10. Conway, R.W., Maxwell, W.L., Miller, L.W.: Theory of Scheduling. Addison-Wesley Publishing Company, Reading (1967)
11. Correa, J.R., Queyranne, M.: Efficiency of equilibria in restricted uniform machine scheduling with total weighted completion time as social cost. Naval Res. Logistics (NRL) **59**(5), 384–395 (2012)
12. Dutot, P.F., Pascual, F., Rzadca, K., Trystram, D.: Approximation algorithms for the multiorganization scheduling problem. IEEE Trans. Parallel Distrib. Syst. **22**(11), 1888–1895 (2011)
13. Hoeksma, R., Uetz, M.: The price of anarchy for minsum related machine scheduling. In: Solis-Oba, R., Persiano, G. (eds.) WAOA 2011. LNCS, vol. 7164, pp. 261–273. Springer, Heidelberg (2012)
14. Immorlica, N., Li, L.E., Mirrokni, V.S., Schulz, A.S.: Coordination mechanisms for selfish scheduling. Theor. Comput. Sci. **410**(17), 1589–1598 (2009)
15. Kollias, K.: Nonpreemptive coordination mechanisms for identical machines. Theory Comput. Syst. **53**(3), 424–440 (2013)
16. Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
17. Liman, S.: Scheduling with capacities and due-dates. Ph.D. thesis, University of Florida (1991)
18. Montreuil, B., Kaspi, M., Ramudhin, A.: Scheduling identical parallel processors with arbitrary initial available times. Université Laval, Faculté des sciences de l'administration (1992)

19. Nash, J.F.: Equilibrium points in n-person games. In: Proceedings of the National Academy of Sciences, vol. 36, no. 1, pp. 48–49 (1950)
20. Pascual, F., Rzadca, K., Trystram, D.: Cooperation in multi-organization scheduling. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 224–233. Springer, Heidelberg (2007)
21. Vöcking, B.: Selfish load balancing. In: Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V. (eds.) Algorithmic Game Theory, pp. 517–542. Cambridge University Press, Cambridge (2007)