Algorithmic game theory and scheduling

Eric Angel, Evripidis Bampis, Fanny Pascual

Algorithmic Game Theory and Scheduling

Eric Angel, Evripidis Bampis, Fanny Pascual

University of Evry Val d'Essonne

81.1 Introduction

Nowadays, understanding the Internet is one of the major issues in theoretical computer science [1]. Toward this direction, a number of researchers consider the Internet as a system of rational selfish agents (or users), each acting his/her own profit. This simplification makes the use of many notions introduced in *noncooperative game theory* possible [2,3], where every agent determines her behavior (strategy) based on the other agents' strategies. The aim of every agent is to maximize her own individual profit, without taking into account the consequences of her choice to the other agents or to the system's performances. A central notion in this field is the notion of *Nash equilibrium* [2] defined as a combination of (deterministic or randomized) choices (strategies), one for each agent, from which no agent has an incentive to unilaterally change her strategy. The existence of such an equilibrium is assured for every finite game from the famous theorem of Nash [2]. However, even if, in the last few years, some progress has been made for particular classes of games [4,5], it remains an *open question* to decide whether or not the problem of computing Nash equilibrium is in general solvable in polynomial time. This question is actually among the most challenging open questions in theoretical computer science [1]. However, a much progress has been made in the related question of evaluating the *impact of the selfishness* of the agents on the efficient use of the system, or, in other words, on the *social welfare* measured in terms of an appropriate global objective function. Given that for a finite game there are, in general, a number of different Nash equilibria that may have different objective function values, this question is very similar to the one of evaluating the quality of a feasible solution for a combinatorial optimization problem. Exploiting this relation, Koutsoupias and Papadimitriou [6] adopted a *worst-case* approach and they introduced the notion of the *price of anarchy* (also known as *coordination ratio*), which is defined as the ratio of the value of the objective function in the *worst* Nash equilibrium and its value at the optimum. In fact, the price of anarchy evaluates the cost of the lack of *coordination*—as opposed to the *competitive ratio*, that evaluates the lack of *information*

in the context of the *on-line* computation or the *approximation ratio*, that is used to evaluate the lack of *unbounded computational resources* in the context of the *off-line* computation.

In this chapter, we present some directions of research related to the notion of the price of anarchy and we illustrate them on three scheduling models involving *selfish agents*: for two of them, namely the KP (Koutsoupias–Papadimitriou) model [6] and the CKN (Christodoulou–Koutsoupias–Nanavati) model of Ref. [7], the agents are the *tasks*, while for the third one, the AT (Archer–Tardos) model [8], the agents are the *machines* (links of the network). Our aim is to offer a starting point for researchers who are interested in this area.

The chapter is organized as follows: in the remaining part of the introduction we present the three scheduling models that we use in the sequel. Section 81.2 is devoted to the notions of the price of anarchy and of the coordination mechanism that we illustrate for the KP and CKN models. In Section 81.3, we deal with the price of stability and the related topic of nashification. An example is given for the CKN model. Section 81.4 concerns the design of truthful algorithms. In the first part, we show how this can be done for the CKN model. In the second part, we give a general result characterizing the algorithms that do admit truthful payment schemes in the context of mechanism design and we illustrate it for the AT model. In Section 81.5, we give a brief state of the art and we point out some related references.

81.1.1 Selfish Tasks: The KP and CKN Models

In the KP and CKN models, n rational agents (tasks) $T_1, \ldots, T_n$ have to choose, among m available machines (also called links or processors) $P_1, \ldots, P_m$, the machine on which they will be scheduled. Each task T_i is characterized by a positive length (execution or processing time) l_i and by an identification number i, and each machine P_j is characterized by its identification number j. The strategy of each agent is either *pure*, that is, decides to be processed on a particular machine (with a probability one), or *mixed*, that is, the strategy is a probability distribution over the machines (goes on each machine with a given probability).

The choice of the machine on which the agent will be processed is based on the *cost* that is associated with each strategy. Let c_i^j be the cost of agent i if the agent chooses to be processed on machine j. In both models, the aim of each agent is to minimize her cost. Therefore, in a Nash equilibrium, the *cost* of agent i is

$$c_i = \min_j c_i^j$$

In the first model [6], called the **KP model**, the aim of each task T_i, $i = 1, \ldots, n$, is to minimize the *load* of the machine j on which she is executed, that is, the sum of the execution times of all the tasks that have been assigned to the same machine as T_i. If p_k^j denotes the probability of task T_k goes on machine j, then the (expected) cost of agent i on machine j is the expected load on machine j, that is,

$$c_i^j = l_i + \sum_{k \neq i} p_k^j l_k$$

In the second model [7], called the **CKN model**, every machine has a *public local policy* (known to all agents) that determines the order on which the tasks that are allocated to this machine will be scheduled. This policy may be a function of the lengths of the tasks, or of their identification numbers, or some other characteristics of the tasks (e.g., such a policy may consist in scheduling the tasks in decreasing order of their lengths). Tasks know the policies of the machines, and the aim of each task is to minimize her own completion time. The (expected) cost of agent i on machine j is then

$$c_i^j = l_i + \sum_{k \prec_j i} p_k^j l_k$$

where $k \prec_j i$ means that task T_k is scheduled before task T_i according to the policy of machine j.

In the sequel, we will consider for this latter model only pure strategies: the probability for a task to go on a given machine will be either 1 or 0.

81.1.2 Selfish Machines: The AT Model

In the AT model [8] there are m machines of different speeds and n tasks with lengths $l_1, \ldots, l_n$. The amount of time to complete task j on machine i is $\frac{l_j}{s_i}$, where s_i is the speed of machine i. Each machine is owned by an agent and the value of s_i is known only by agent i. Each agent has to report her speed, and based on the *reported* speeds, the mechanism $M = (A, \mathcal{P})$ constructs a schedule (i.e., an allocation of the tasks on the machines) using an algorithm A, and then pays the agents, according to a payment function $\mathcal{P}$, to compensate the cost induced by the processing of the tasks. Let $\mathcal{P}_i$ denote the amount of money (as computed by the payment function $\mathcal{P}$) given to agent i. The *profit* of agent i is defined as

$$profit_i = \mathcal{P}_i - \frac{w_i}{s_i}$$

that is, her payment minus the cost incurred by the agent in being assigned work w_i (w_i is the sum of the processing times of all the tasks that have been allocated to machine i). The *cost* for each agent is the opposite of her profit. The aim of each agent is to maximize her profit (i.e., to minimize her cost), and an agent may *lie* and bid a false value (a value different from her real speed) if this can increase her profit. The (social) cost of a schedule is the maximum load over all machines, that is, its *makespan*. The goal in this model is to design a mechanism (i.e., a schedule and a payment function), which is truthful (i.e., it assures that every agent has no incentive to lie), and which, at the same time, minimizes the makespan.

81.2 Price of Anarchy and Coordination Mechanisms

In many networks, users and service providers act selfishly, without an authority that monitors and regulates network operation to achieve some "social optimum." *How much performance is lost because of this?* This question, raised by Papadimitriou and Koutsoupias [6], opens many theoretical problems where the main question concerns the cost of the lack of *coordination*, called *price of anarchy*. This notion is studied in the next section and is illustrated in the case of the KP model. However, from an algorithmic point of view what is more interesting is to find ways for decreasing the price of anarchy in such a context. Such an approach has been introduced by Christodoulou et al. [7], where the notion of *coordination mechanism* is proposed. We discuss this notion in Section 81.2.2 and we illustrate it for the CKN model.

81.2.1 Price of Anarchy for the KP Model

In this section we consider the KP model and show that the price of anarchy on two identical machines is 3/2.

Let us consider an arbitrary task i and let us assume that she decides to go to machine j with a probability 1, that is, $p_i^j = 1$, then its cost, that is, the average load of machine j, is $c_i^j = l_i + \sum_{k \neq i} p_t^j l_k$. In the general case, the expected cost of task i is $\sum_j p_i^j c_i^j$. In a Nash equilibrium there is no incentive for task i to change its strategy, therefore it assigns nonzero probabilities only to machines j that minimize c_i^j. We note $c_i = \min_j c_i^j$.

For example, consider two tasks, each one with a length 1. The probabilities $p_i^j = 1/2$ for $i = 1, 2$ and $j = 1, 2$ give rise to a Nash equilibrium. Indeed, we have $c_1^1 = 1 + 1/2 \times 1 = 3/2$ and also $c_1^2 = c_2^1 = c_2^2 = 3/2$. What is the expected makespan of this Nash equilibrium? With a probability 1/4 (resp. 1/4) all tasks go to machine 1 (resp. 2) giving rise to a makespan of 2, with a probability 1/4 (resp. 1/4) task 1 goes to machine 1 (resp. 2) and task 2 goes to machine 2 (resp. 1), giving rise to a makespan of 1. The expected makespan is thus $1/4 \times 2 + 1/4 \times 2 + 1/4 \times 1 + 1/4 \times 1 = 3/2$. Note that the expected makespan is not the maximum over all machines of their average load. The average load of machine j is $M^j = \sum_i p_i^j l_i$, and it is 1 in this example. Of course, the optimum makespan of 1 is obtained when task 1 (resp. 2) goes to machine 1 (resp. 2). Recall that the price of anarchy is defined as the ratio between the (worst) expected makespan in a worst Nash equilibrium over the optimal makespan (achievable over the set of all schedules, i.e., not necessarily Nash equilibria). Therefore, this example shows that the price of anarchy on two identical machines is at least 3/2.

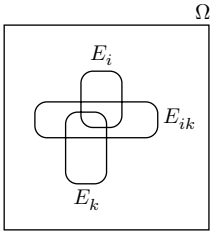

FIGURE 81.1 Let E_i (resp. E_k) denote the event "Task i (resp. k) is scheduled on the machine with the maximum load," and E_{ik} denote the event "Tasks i and k collide." One has $E_i \cap E_k \subseteq E_{ik}$. Hence, $q_i + q_k = \text{Proba}(E_i) + \text{Proba}(E_k) = \text{Proba}(E_i \cup E_k) + \text{Proba}(E_i \cap E_k) \leq 1 + \text{Proba}(E_{ik}) = 1 + t_{ik}$.

In the sequel we consider any Nash equilibrium, and we want to upper bound its expected makespan with respect to the optimum makespan, to show that this ratio 3/2 is tight.

The *contribution probability* q_i of task i is equal to the probability that this task is scheduled on the machine of maximum load (ties are broken using the machine with the smallest identification number). Therefore, the expected makespan is $\sum_i q_i l_i$.

Lemma 81.1 (Koutsoupias and Papadimitriou [6])
For any task i, one has $\sum_{k\neq i}(q_i + q_k)l_k \leq \frac{3}{2}\sum_{k\neq i} l_k$.

Proof
Let t_{ik} be the *collision probability* of tasks i and k, that is, the probability that they are scheduled on the same machine. Observe that if both tasks i and k contribute to the expected makespan, that is they are scheduled on the machine with the maximum load, then they must collide. This is illustrated in Figure 81.1.

Therefore, one has the following inequality:

$$q_i + q_k \leq 1 + t_{ik} \tag{81.1}$$

Moreover, we will show that for any task i

$$\sum_{k\neq i} t_{ik} l_k = c_i - l_i \tag{81.2}$$

We have $t_{ik} = \sum_j p_i^j p_k^j$, then $\sum_{k\neq i} t_{ik} l_k = \sum_j p_i^j \sum_{k\neq i} p_k^j l_k = \sum_j p_i^j (c_i^j - l_i) = \sum_j p_i^j (c_i - l_i)$. The last equality comes from the fact that since in a Nash equilibrium task i assigns nonzero probabilities only to machines j that minimize c_i^j, we have $p_i^j \neq 0$ when $c_i^j = c_i$ and $p_i^j = 0$ otherwise. Finally, since $\sum_j p_i^j = 1$ we obtain equality (81.2).

Now we will show that

$$c_i \leq \frac{\sum_k l_k}{2} + \frac{1}{2} l_i \tag{81.3}$$

Recall that $c_i^j = l_i + \sum_{t\neq i} p_t^j l_t = M^j + (1 - p_i^j) l_i$, with $M^j = \sum_i p_i^j l_i$ the expected load on machine j. Note that

$$M^1 + M^2 = \sum_k l_k \tag{81.4}$$

since $p_i^1 + p_i^2 = 1$ for all tasks i.

Therefore, we have

$$\begin{aligned} c_i &= \min_{j=1,2} c_i^j \\ &\leq \frac{1}{2}(c_i^1 + c_i^2) \end{aligned}$$

$$
\begin{aligned}
&\leq \frac{1}{2}\sum_{j=1}^{2}(M^j + (1-p_i^j)l_i) \\
&= \frac{M^1+M^2}{2} + \frac{1}{2}l_i \text{ since } p_i^1 + p_i^2 = 1 \\
&= \frac{\sum_k l_k}{2} + \frac{1}{2}l_i \quad \text{using Eq. (81.4)} \\
&\quad \text{and this proves Eq. (81.3).}
\end{aligned}
$$

Finally, we can write

$$
\begin{aligned}
\sum_{k\neq i}(q_i+q_k)l_k &\leq \sum_{k\neq i}(1+t_{ik})l_k \quad \text{using (81.1)} \\
&\leq \sum_{k\neq i} l_k + \sum_{k\neq i} t_{ik}l_k \\
&= \sum_{k\neq i} l_k + c_i - l_i \quad \text{using (81.2)} \\
&\leq \sum_{k\neq i} l_k + \frac{\sum_k l_k}{2} - \frac{1}{2}l_i \quad \text{using (81.3)} \\
&= \frac{3}{2}\sum_{k\neq i} l_k
\end{aligned}
$$

□

Theorem 81.1 (**Koutsoupias and Papadimitriou [6]**)

The price of anarchy for any number of tasks and two machines is 3/2.

Proof

We have already seen, with the example given in the beginning of this section, that the price of anarchy is at least 3/2. We show that is at most 3/2.

Let *OPT* be the optimum makespan. We will to compare the expected makespan in any Nash equilibrium versus the optimum makespan *OPT*.

Let us assume that $q_i \leq 3/4$ for all tasks i. Then the expected makespan is $\sum_k q_k l_k \leq \frac{3}{4}\sum_k l_k \leq \frac{3}{2}\,OPT$, since $OPT \geq \frac{1}{2}\sum_k l_k$.

Otherwise there exists a task i such that $q_i \geq 3/4$. In that case,

$$
\begin{aligned}
\sum_k q_k l_k &= \sum_{k\neq i} q_k l_k + q_i l_i \\
&\leq \frac{3}{2}\sum_{k\neq i} l_k - \sum_{k\neq i} q_i l_k + q_i l_i \quad \text{using Lemma 81.1} \\
&= \frac{3}{2}\sum_k l_k - \frac{3}{2}l_i - \sum_k q_i l_k + 2q_i l_i \\
&= \left(\frac{3}{2} - q_i\right)\sum_k l_k + \left(2q_i - \frac{3}{2}\right)l_i \\
&\leq \left(\frac{3}{2} - q_i\right)2\,OPT + \left(2q_i - \frac{3}{2}\right)OPT \quad \text{since } 2q_i - \tfrac{3}{2} \geq 0, \text{ and using } OPT \\
&\geq \max\left\{\tfrac{1}{2}\textstyle\sum_k l_k, l_i\right\} \\
&= \frac{3}{2}\,OPT
\end{aligned}
$$

□

81.2.2 Coordination Mechanisms for the CKN Model

From an *algorithmic* point of view, an important question concerns possible ways to reduce the price of anarchy. One of the proposed approaches is due to Christodoulou et al. [7], who introduced the notion of *coordination mechanism*. A coordination mechanism is a set of local policies, one for each facility, that have to be based *only* on the characteristics of the local agents and not require any additional resources or alter the distributed nature of the system (the local policies are fixed once and for all before having any knowledge of the input).[1]

For instance, the local policy of a facility may give priorities to the agents or introduce delays. Knowing the coordination mechanism and the characteristics of the other agents, each agent chooses on which facility she goes, and she is then allocated to it, according to the policy of the facility.

The *price of anarchy* of a coordination mechanism is defined as the ratio between the objective function in the worst Nash equilibrium that we can obtain with this coordination mechanism, and the optimal value of the objective function (obtained in a centralized way, and which is not necessarily a Nash equilibrium).

Let us consider the following coordination mechanism for the CKN model on two machines: on each machine the tasks are sorted in order of increasing lengths: a task T_i is said larger than a task T_j if and only if the length of T_i is larger than the one of T_j ($l_i > l_j$) or the tasks have the same lengths and the identification number of T_i is smaller than the one of T_j ($i < j$). The first machine, denoted by P_{SPT}, schedules the tasks in order of increasing lengths, and the second machine, denoted by P_{LPT}, schedules the tasks in order of decreasing lengths.

With this mechanism, denoted by LPT-SPT, every task knows on which machine it will be scheduled first: a task T_i will go on P_{SPT} if the total length of the tasks that are smaller than T_i is smaller than or equal to the total length of the tasks that are larger than T_i; otherwise T_i will have incentive to go on P_{LPT}.

Theorem 81.2 (Christodoulou et al. [7])

The coordination ratio of the LPT-SPT mechanism over two machines is $4/3$.

Proof

Let us consider any schedule S obtained by this coordination mechanism, and let us show that the makespan of this schedule is smaller than or equal to $\frac{4}{3}\,OPT$, where OPT is the smallest makespan of any schedule involving the same tasks. Let T_r be the last task to be completed in S, and let C_{max} denote its completion time (i.e., C_{max} is the makespan of S).

Let L_1 (resp. L_2) be the sum of lengths of tasks scheduled on machine 1 (resp. 2), without taking into account task T_r. One has $C_{max} = \min\{L_1, L_2\} + l_r \leq (L_1 + L_2)/2 + l_r$. Therefore, $C_{max} \leq \frac{1}{2}\sum_{i\neq r} l_i + l_r \leq \frac{1}{2}\sum_{i=1}^{n} l_i + (1-\frac{1}{2})l_r \leq OPT + \frac{1}{2}l_r$. If $l_r \leq \frac{2}{3}\,OPT$ then $C_{max} \leq \frac{4}{3}\,OPT$.

Let us now consider the case $l_r > \frac{2}{3}\,OPT$.

If T_r is scheduled on P_{LPT}, then the schedule is optimal. Indeed, if T_r is the only task of P_{LPT}, then S is an optimal schedule. In the other cases, there is at least one task before T_r on P_{LPT} and this task has a length larger than $\frac{2}{3}\,OPT$. Likewise, since task T_r has no incentive to go on P_{SPT}, there are on P_{SPT} tasks whose sum of lengths is larger than $\frac{2}{3}\,OPT$. Thus $\sum_{i=1}^{n} l_i > \frac{6\,OPT}{3} = 2\,OPT$, which is impossible.

If T_r is scheduled on P_{SPT}, then there exists a task T_1 of length larger than or equal to l_r on P_{LPT}. Let x be the sum of the lengths of the tasks scheduled before T_r on P_{SPT}. Since $l_1 + x + l_r \leq 2\,OPT$, and since $l_1 > \frac{2}{3}\,OPT$, we deduce that $l_r + x \leq \frac{4}{3}\,OPT$, that is, $C_{max} \leq \frac{4}{3}\,OPT$. □

We have shown that the coordination ratio of this mechanism is at most $\frac{4}{3}$. We can notice that this bound is tight by considering the following instance: two tasks of lengths 1 and two tasks of lengths 2. With this coordination mechanism the makespan will be 4, whereas the makespan of an optimal solution (where one task of length 1 and one task of length 2 are scheduled on each machine) is 3.

[1] This approach is related to the classical approach of game theory, the approach of *Mechanism Design*, where the agents are "paid" to cooperate [3].

However, this coordination mechanism cannot be generalized for m machines. Hence, Christodoulou et al. [7] proposed in the following coordination mechanism, that has a price of anarchy of $\frac{4}{3} - \frac{1}{3m}$: all the machines use a Longest Processing Time (LPT) local policy (i.e., every machine schedules its tasks in order of decreasing lengths) and introduce small (negligible) delays to break ties. In this way, it can be shown that the tasks have incentive to choose the machine on which they would have been affected by a centralized LPT algorithm (i.e., an algorithm which schedules greedily the tasks in the decreasing order of their lengths without leaving a machine idle whenever there is an unscheduled task). Thus, the price of anarchy of this mechanism is equal to the approximation ratio of the LPT algorithm (delays can be as small as we wish) [9]. In Ref. [7], the authors conjecture that there is no coordination mechanism with a better coordination ratio.

81.3 Price of Stability

In many applications, it is not true that the users (or agents) are interacting directly with each other. In reality, they interact with a protocol, which proposes a collective solution to all of them and the users are free to accept or reject. Hence, in these applications it is necessary to design protocols (algorithms) producing the *best* (or a near optimal) Nash equilibrium, that is, a *stable* (near) optimal solution such that no agent has incentive to defect from it [10]. A new measure for evaluating the impact of searching a solution under the constraint that the returned solution must be stable (i.e., a Nash equilibrium) has been introduced by Anshelevich et al. [10]: the *price of stability* is defined as the ratio of the objective function in the best Nash equilibrium and the global optimum (this maximum is taken over all instances). This measure can be viewed as the *optimistic* price of anarchy.

For the KP model there is always a pure Nash equilibrium with minimum makespan, so the price of stability is equal to 1 for this model. This result is a direct corollary of the results of Ref. [11] stating that for the KP model it is always possible to modify (*nashify*) an arbitrary initial solution to a pure Nash equilibrium without increasing the value of the makespan. However, for the CKN model this is not possible and the price of stability depends on the policies of the machines. Indeed, if for instance, we consider that the policies of the machines are LPT (each machine schedules the tasks in order of decreasing lengths), then the only pure Nash equilibrium is the schedule obtained by a centralized LPT schedule. Given that the approximation ratio of such an algorithm is $\frac{4}{3} - \frac{1}{3m}$ [9], the price of stability for the CKN model with LPT local policies is $\frac{4}{3} - \frac{1}{3m}$.

81.3.1 Approximate Stability for the CKN Model

In this section, we relax the definition of "stable" schedule: A solution is said to be *stable* if it corresponds to an α*-approximate Nash equilibrium*, that is, to a situation in which no agent has *sufficient incentive* to unilaterally change her behavior. We say that an agent does not have *sufficient incentive* to unilaterally leave the machine on which she is scheduled, if and only if this change does not increase her profit by more than α times its current profit, where α is a given threshold ($\alpha \geq 1$). If in a solution no agent has sufficient incentive to change strategy, then this solution is an α-approximate-Nash equilibrium. If $\alpha = 1$ then the schedule is an (exact) Nash equilibrium. Thus, we can define *the price of* α*-approximate stability* as the maximum ratio between the value of the objective function in the *best* α-approximate Nash equilibrium, and the value of the objective function in the global optimum.

Let us illustrate this notion for the CKN model with two identical machines whose policies are LPT. In the sequel, a task is said to be α-approximate if it will not reduce its completion time by a factor greater than α by changing machine.

We prove that this relaxation allows to reduce the price of stability to $(1 + \epsilon)$, while preserving an α-approximate Nash equilibrium, with α bounded by a constant. To do so, we consider the classical polynomial-time approximation scheme (PTAS) of Graham [9], modified slightly:

(1) Let k be some specified and fixed integer.

(2) Obtain an optimal schedule for the k largest tasks, such that:
   - Once tasks are assigned to each machine, they are scheduled on their machines in order of decreasing lengths (i.e., for a given machine, tasks are scheduled from the largest to the smallest one).
   - If two tasks have the same length, the one which has the smallest identification number is scheduled first.

(3) Schedule the remaining $n - k$ tasks using the LPT algorithm.

Note that since k is a constant, step (2) of this algorithm takes a polynomial time using an exhaustive enumeration search. This algorithm is a PTAS, and its approximation ratio is $1 + \varepsilon$, where ε is equal to $\frac{1}{2+2\lfloor \frac{k}{2} \rfloor}$, if the k largest tasks of the schedule are optimally scheduled [9]. Let us now show that this algorithm, denoted by OPT-LPT(k), always returns an α-approximate-Nash equilibrium, with $\alpha < k-2$, for the CKN model with two machines P_1 and P_2 whose policies are LPT.

Theorem 81.3 (Angel et al. [12])

The schedule returned by algorithm OPT-LPT(k) is an α-approximate-Nash equilibrium, with $\alpha < k - 2$, for two machines whose policies are LPT.

Proof

Let us show that each task of an OPT-LPT(k) schedule either does not have incentive to change machine (because she would increase its completion time by going on the other machine), or does not decrease her completion time by a factor larger than or equal to $k-2$, by going on the other machine. The $n-k$ smallest tasks of the schedule are scheduled using the LPT rule, so they do not have incentive to change machine. Thus we consider the k largest tasks. Let *OPT* be the optimal solution of these tasks, such as computed by OPT-LPT(k). We will now consider three cases. In the sequel we will denote the ith task of P_1 by x_i, the ith task of P_2 by y_i, and $l(t)$ will denote the length of task t. We will say that task t_1 is larger than task t_2 if $l(t_1) > l(t_2)$ or if $l(t_1) = l(t_2)$, and the identification number of t_1 is smaller than the identification number of t_2. Likewise t_1 is said smaller than t_2 if t_2 is larger than t_1.

- In the first case, there is, in *OPT*, only one task on a machine (w.l.o.g. on P_1), and $k-1$ tasks on the other machine. Since this schedule is an optimal solution, the task on P_1 is necessarily the largest task on the schedule, and this schedule is an LPT schedule. So no task has incentive to change machine in this case.
- Let us now consider the case where there are exactly two tasks on a machine (w.l.o.g. on P_1) in *OPT*. The others $k-2$ tasks are then on P_2.

We first show that no task scheduled on P_2 has incentive to go on P_1. By construction, we know that $l(x_1)+l(x_2)$ is larger than or equal to the sum of the lengths of the $k-3$ first tasks of P_2, $\sum_{j=1}^{k-3} y_j$. Let i be the largest number such that $l(x_1) \geq \sum_{j=1}^{i} y_j$: the $i+1$ first tasks of P_2 (i.e., the tasks who start at the latest at the end of x_1) do not have incentive to go on P_1, otherwise they would be scheduled after task x_1 and would not decrease their completion times. Moreover, we know that $l(x_2) \geq \sum_{j=i+2}^{k-3} y_j$: thus the tasks from y_{i+2} to y_{k-3} do not have incentive to change machine. Likewise, y_{k-2} does not have incentive to change: if it is smaller than x_2, then she would be scheduled on P_1 after x_2, and would not decrease her completion time, since *OPT* is an optimal solution. If y_{k-2} is larger than x_2, then y_{k-2} starts to be executed before (or at the same time as) x_2, otherwise by switching x_2 with y_{k-2} we could obtain a better solution than *OPT*. Thus, if it goes on P_1, y_{k-2} will be scheduled after x_1, and then will not decrease her completion time (note that y_{k-2} is smaller than x_1, otherwise *OPT* would not be an optimal solution).

The only task which may have incentive to change machine is x_2. If x_2 is smaller than all the other tasks, then she does not have incentive to change. Otherwise, since *OPT* is an optimal solution, we know that at least a task of P_2 starts at the same time or after x_2. In the best case, x_2 can go to the first position on P_2: by doing this, she starts on P_2 before at most $k-3$ tasks, which started before her when she was on P_1. These $k-3$ tasks are smaller than x_2: the sum of their completion times, S, is thus smaller than $(k-3)\,l(x_2)$.

The completion time of x_2 decreases with this change, from $S + l(x_2) < (k-2)\,l(x_2)$ to $l(x_2)$. Thus x_2 is, in *OPT*, α-Nash-approximate, with $\alpha < k-2$.

- Let us now consider the case where there are exactly $a < k-2$ tasks on P_1, and $b < k-2$ tasks on P_2. Let t be a task on P_1 (resp. P_2), who has incentive to change machine. When she changes machine, t overtakes p tasks of P_2 (resp. P_1), that is, she starts to be executed before p tasks, which started to be executed before t, that is before the change. We know that p is smaller than $k-2$ because there are less than $k-2$ tasks on each machine. Moreover, these tasks have a length smaller than the one of t, otherwise t would not overtake them. Thus, in the best case, t overtakes $k-3$ tasks of length almost equal to $l(t)$, and the completion time of t decreases from a value smaller than $(k-2)\,l(t)$ to $l(t)$. Thus t is α-Nash-approximate, with $\alpha < k-2$. □

We saw that OPT-LPT(k) returns α-approximate Nash equilibria, with $\alpha < k-2$. Let us now show that this bound is tight.

Theorem 81.4 (Angel et al. [12])

Let ε be any small number such that $0 < \varepsilon < 1$. OPT-LPT(k) can return α-approximate Nash equilibria, with $\alpha \geq k-2-\varepsilon$ and $k \geq 5$.

Proof

Let $\varepsilon' = \frac{\varepsilon}{k-2-\varepsilon}$, and let us consider the following instance: a task of length $k-3-\varepsilon'$, a task of length $1+\varepsilon'$, and $k-2$ tasks of length 1. The only optimal solution for this instance is the schedule where the tasks of length 1 are on the same machine (w.l.o.g. on P_2), and the two other tasks on the other machine. Let t denote the task of length $1+\varepsilon'$: t is completed on P_1 at time $k-2$. Note that $1+\varepsilon' < k-3-\varepsilon'$ for any $0 < \varepsilon < 1$ and $k \geq 5$, thus by going on the other machine, t would end at time $1+\varepsilon'$, and then decrease her completion time by $\frac{k-2}{1+\varepsilon'} = k-2-\varepsilon$. Thus t is $(k-2-\varepsilon)$-approximate. The schedule returned by OPT-LPT(k) on this instance is an α-approximate Nash equilibrium, with $\alpha \geq k-2-\varepsilon$. □

We can deduce from Theorem 81.3, and from the fact that the approximation ratio of OPT-LPT(k) is $\frac{1}{2+2\lfloor \frac{k}{2} \rfloor}$, the following result:

Corollary 81.1

Let k be any integer larger than or equal to 5. For the two machine scheduling game, if the local policies of the links are LPT, then the price of α-approximate stability is at most $1+\varepsilon$, where $\varepsilon = \frac{1}{4+2\lfloor \frac{k}{2} \rfloor} < \frac{1}{k}$, for all $\alpha \geq k$.

81.4 Truthful Algorithms

Another important aspect of systems based on the selfish behavior of a set of agents is the notion of *truthfulness*. Consider a system with selfish agents, where the network is organized by a protocol whose aim is the maximization of the social welfare. The underlying assumption, in such a context, is that the agents on whom the protocols are applied are trustworthy. However, this assumption is unrealistic in some settings as the agents might try to manipulate the protocol by reporting false information to get some advantages. With false information, even the most efficient protocol may lead to unreasonable solutions if it is not designed to cope with the selfish behavior of the single entities. Thus, if every agent has a value (for a task, its length; for a machine, its speed; etc.), which is known only by herself, it is useful to design algorithms that are able to give incentive to the agents to bid their real secret values. Such algorithms are called *truthful*.

The field of *mechanism design* provides a theory to get truthful algorithms. The main idea of this theory is to pay the agents to convince them to declare their true values, thus helping the system to solve the

optimization problem correctly. This is possible if every agent incurs some *monetary* cost. It is common to allow side payments to the agents, and to assume that each agent tries to maximize the sum of her payment and her intrinsic cost of the outcome. Thus a mechanism $M = (A, \mathcal{P})$ is a couple, where A is an algorithm that computes an allocation of the resources (in our case which affects tasks to machines), and $\mathcal{P}$ is a payment function that affects to every agent a certain amount of money. A mechanism is *truthful* if and only if the profit of every agent is maximized when the agent reveals her true value. The main general result in this direction is the Vickrey–Clarke–Groves (VCG) mechanism [13–15]. It handles arbitrary agents' cost functions and guarantees the truthfulness under the hypothesis that the global objective function is utilitarian (it optimizes the sum of the agents' costs or profits), and that the mechanism is able to compute the optimal solution. This is the case, for instance, when the agents are the tasks and they want to reduce their individual completion times, and the global objective function is the sum of completion times. However, if the objective function is the makespan, then the VCG mechanism cannot be used anymore since the minimization of the makespan is an NP-hard problem (and thus it is not possible to compute the optimum in polynomial time unless P = NP), and in addition the problem is not utilitarian since the goal is the minimization of the *maximum* completion time over all the machines.

In this section, we consider problems where the objective function is the makespan. In Section 81.4.1, we study the case where the agents are the tasks (CKN model): every agent is a task, which is the only one to know its real length and may lie on the value of this length. We present a (randomized) truthful mechanism without payment where the incentive of every agent is based only on her cost which is her (expected) completion time.

In Section 81.4.2, we consider the case where the agents are the machines (AT model): each agent is a machine that may lie on the value of its speed. We first show a fundamental result, due to Archer and Tardos [8], for the following type of problems: a set of loads have to be allocated by the mechanism on a set of machines (agents), and every agent has a secret data, which is a single positive real number that represents the cost incurred per unit of load assigned to this agent. These problems are known as *One-Parameter-Agent* problems. Archer and Tardos [8] showed that a mechanism $M = (A, \mathcal{P})$ for a problem with one-parameter agents is truthful if and only if algorithm A is *monotone*: an algorithm is monotone if, given the secret data $b_1, \ldots, b_n$ of the agents, then, for any i and fixed b_j ($j = 1, \ldots, n$ with $j \neq i$), the load assigned to agent i is nondecreasing with respect to b_i. Moreover, they show how to construct a payment function $\mathcal{P}_A$ such that if A is monotone then $(A, \mathcal{P}_A)$ is a truthful mechanism.

It is interesting to see that our selfish machines problem is a problem involving one-parameter agents: the agents are the machines, the loads are the tasks, and the agent's secret data is the inverse of her speed. Thus, an algorithm will be truthful for this problem if and only if it is monotone. Intuitively, monotonicity here means that increasing the speed of exactly one machine does not make the algorithm decrease the load assigned to that machine. We use this result in Section 81.4.2.2 to show that the LPT algorithm is truthful if the speeds of the machines are powers of a constant higher than or equal to 2.

81.4.1 Truthful Algorithm for Scheduling Selfish Tasks

We consider in this section selfish tasks whose aim is to reduce their completion times (CKN model). We assume that the length of each task (agent) is a private value, known only by herself, and that each agent bids a value representing her length. We focus on the following process: at first the agents declare their lengths; then given these bids the system allocates the tasks to the machines. The objective of the system is to minimize the makespan. The aim of each agent is to minimize her completion time and an agent may lie on this value if this can improve her (expected) completion time. We assume that the tasks cannot shrink their lengths and thus they will not bid values smaller than their real lengths, but they may bid values larger than their real lengths. There is a natural way to get a truthful deterministic algorithm, it is sufficient to schedule the tasks according to the *Shortest Processing Time* (SPT) algorithm, in which tasks are greedily scheduled from the smallest task to the largest task. The approximation ratio of this algorithm is $2 - \frac{1}{m}$ (see Ref. [9]). Here we will present a (randomized) truthful algorithm, which has an (expected) approximation ratio better than the one of SPT.

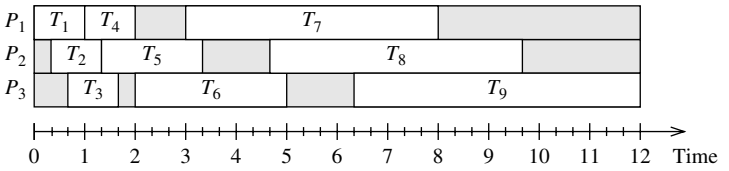

FIGURE 81.2 Example of an SPT_δ schedule.

Since SPT is truthful and LPT has a good approximation ratio, it is natural to wonder whether a randomized algorithm that returns the SPT schedule with a probability p and the LPT schedule with a probability $1 - p$, is truthful for some values of p ($0 \leq p \leq 1$). We can easily see that this algorithm is not truthful for any p smaller than 1. Indeed, consider the following instance on two machines: a task T_1 of length 1, a task T_2 of length 2, and a task T_3 of length 3. If T_1 bids its true value, then it will be scheduled first in the SPT schedule, and then finish at time 1; and it will be scheduled in the LPT schedule after task T_2, and then finish in this schedule at time 3. Thus, the expected completion time of T_1 if it bids its true value is $p + 3(1 - p) = 3 - 2p$. If T_1 bids 2.5 (instead of its true value 1), then it will be on the first position in both the SPT and the LPT schedules. In both cases its completion time will be 1. Since the expected completion time of T_1 is smaller if it bids a false value rather than if it bids its true value, this algorithm is not truthful. We will now see that if we slightly modify the SPT algorithm, then a randomized algorithm of this type is truthful.

Let us consider the following algorithm, denoted by SPT_δ in the sequel:
Let $\{T_1, T_2, \ldots, T_n\}$ be n tasks to be scheduled on $m \geq 2$ identical machines, $\{P_1, P_2, \ldots, P_m\}$. Let us suppose that $l_1 \leq l_2 \leq \cdots \leq l_n$. Tasks are scheduled alternatively on $P_1, P_2, \ldots, P_m$, in order of increasing lengths, and T_{i+1} starts to be executed when exactly $\frac{1}{m}$ of task T_i has been executed. Thus T_1 starts to be scheduled on P_1 at time 0, T_2 is scheduled on P_2 at time $\frac{l_1}{m}$, T_3 is scheduled on P_3 (on P_1 if $m = 2$) when $\frac{1}{m}$ of T_2 has been executed, that is, at time $\frac{l_1}{m} + \frac{l_2}{m}$, and so forth.

The schedule returned by SPT_δ will be called an SPT_δ schedule in the sequel. Figure 81.2 shows an SPT_δ schedule, where $m = 3$.

Let us now consider the following algorithm, denoted by LS_δ in the sequel:
Let m be the number of machines. With a probability of $\frac{m}{m+1}$, the output schedule is an SPT_δ schedule, and with a probability $\frac{1}{m+1}$, the output schedule is an LPT schedule.

We will now show that this algorithm has an approximation ratio better than the one of SPT (cf. Theorem 81.6) and that it is truthful (cf. Theorem 81.7). But first, we need to find the approximation ratio of SPT_δ.

Theorem 81.5 (Angel et al. [16])

The algorithm SPT_δ is $(2 - \frac{1}{m})$-approximate: the makespan of an SPT_δ schedule is smaller than or equal to $(2 - \frac{1}{m})$ OPT, where OPT is the makespan of an optimal schedule for the same tasks.

Proof

We have n tasks $T_1, \ldots, T_n$, such that $l_1 \leq \cdots \leq l_n$, to schedule on m machines. Each task T_i starts to be executed exactly when $\frac{1}{m}$ of T_{i-1} has been executed. So, if $n \leq m$, then the makespan of the SPT_δ schedule is $\frac{1}{m}(l_1 + \cdots + l_{n-1}) + l_n \leq \frac{1}{m}(n-1)\, l_n + l_n \leq \frac{(2m-1)\, l_n}{m} \leq (2 - \frac{1}{m})\, l_n \leq (2 - \frac{1}{m})\, OPT$, since $l_n \leq OPT$.

Let us now consider the case $n > m$. Let $i \in \{m+1, \ldots, n\}$. Task T_i starts to be executed when $\frac{1}{m}$ of T_{i-1} is executed, and T_{i-1} started to be executed when $\frac{1}{m}$ of T_{i-2} was executed, etc., $T_{(i-m)+1}$ started to be executed when $\frac{1}{m}$ of T_{i-m} was executed. So the idle time between T_i and T_{i-m} is $idle(i) = \frac{1}{m}(l_{i-m} + l_{i-m+1} + \cdots + l_{i-1}) - l_{i-m}$.

Let $i \in \{2, \ldots, m\}$. The idle time before T_i is equal to $idle(i) = \frac{1}{m}(l_1 + \cdots + l_{i-1})$, and there is no idle time before T_1, which starts to be executed at time 0. Thus, the sum of the idle times between tasks is $\sum_{i=2}^{n} idle(i) = \frac{1}{m}((m-1)l_{n-m+1} + (m-2)l_{n-m+2} + \cdots + l_{n-1})$.

Let $j \in \{n-m+1, \ldots, n-1\}$. Let $end(j)$ be the idle time in the schedule after the end of task T_j and before the end of T_n: $end(j) = l_{j+1} - \frac{m-1}{m} l_j + end(j+1)$, where $end(n) = 0$. So the sum of the idle times after the last tasks and before the end of the schedule is $\sum_{j=n-m+1}^{n-1} end(j) = (m-1)\left(l_n - \frac{m-1}{m} l_{n-1}\right) + (m-2)\left(l_{n-1} - \frac{m-1}{m} l_{n-2}\right) + \cdots + \left(l_{n-m+2} - \frac{m-1}{m} l_{n-m+1}\right)$.

The sum of the idle times on the machines, from the beginning of the schedule until the makespan, is the sum of the idle times between tasks (and before the first tasks), plus the sum of the idle times after the end of the last task of a machine and before the makespan. It is equal to $\sum_{i=2}^{n} idle(i) + \sum_{j=n-m+1}^{n-1} end(j) = \frac{1}{m}\left((m-1)l_{n-m+1} + (m-2)l_{n-m+2} + \cdots + l_{n-1}\right) + (m-1)\left(l_n - \frac{m-1}{m} l_{n-1}\right) + (m-2)\left(l_{n-1} - \frac{m-1}{m} l_{n-2}\right) + \cdots + \left(l_{n-m+2} - \frac{m-1}{m} l_{n-m+1}\right) = (m-1)\, l_n$.

Let ξ be the makespan of an SPT_δ schedule. ξ is the sum of the tasks plus the sum of the idle times, divided by m: $\xi = \frac{\left(\sum_{i=1}^{n} l_i\right) + (m-1)\, l_n}{m} = \frac{\sum_{i=1}^{n} l_i}{m} + \frac{(m-1)\, l_n}{m}$. Since $\frac{\sum_{i=1}^{n} l_i}{m} \leq OPT$ and $l_n \leq OPT$, we have $\xi \leq (2 - \frac{1}{m})\, OPT$. □

Theorem 81.6 (**Angel et al. [16]**)

The expected approximation ratio of LS_δ is $2 - \frac{1}{m+1}\left(\frac{5}{3} + \frac{1}{3m}\right)$.

Proof

The approximation ratio of an SPT_δ schedule is $2 - \frac{1}{m}$ (see Theorem 81.5), and the approximation ratio of an LPT schedule is $\frac{4}{3} - \frac{1}{3m}$ (see Ref. [9]). Thus the expected approximation ratio of LS_δ is $\frac{m}{m+1}\left(2 - \frac{1}{m}\right) + \frac{1}{m+1}\left(\frac{4}{3} - \frac{1}{3m}\right) = \frac{1}{m+1}\left(2m - 1 + \frac{4}{3} - \frac{1}{3m}\right) = \frac{1}{m+1}\left(2(m+1) - \frac{5}{3} - \frac{1}{3m}\right) = 2 - \frac{1}{m+1}\left(\frac{5}{3} + \frac{1}{3m}\right)$. □

For example, in the case where we have two machines, the expected approximation ratio is $\frac{25}{18} < 1.39$, whereas the approximation ratio of SPT is $2 - \frac{1}{m} = 1.5$ in this case.

Let us now show that LS_δ is truthful.

Theorem 81.7 (**Angel et al. [16]**)

The algorithm LS_δ is truthful.

Proof

Let us suppose that we have n tasks $T_1, \ldots, T_n$, ordered by increasing lengths, to schedule on m machines. Let us show that any task T_i does not have incentive to bid a length higher than her true length. Let us suppose that task T_i bids $b > l_i$, and that, by bidding b, T_i is now larger than all the tasks $T_1, \ldots, T_x$, and smaller than T_{x+1}. In the LPT schedule, the tasks T_{x+1} to T_n are scheduled in the same way, whatever T_i bids (l_i or b). By bidding b, T_i can, at best, start $(l_{i+1} + \cdots + l_x)$ time units before she had bided l_i. Thus the expected completion time of T_i in LS_δ decreases by at most $\frac{1}{m+1}(l_{i+1} + \cdots + l_x)$ time units when T_i bids b instead of l_i.

However, by bidding b instead of l_i, T_i will end later in the SPT_δ schedule: in this schedule, tasks from T_{i+1} to T_x will be started before T_i. Since a task T_j starts to be scheduled when $\frac{1}{m}$ of her predecessor T_{j-1} is executed, by bidding b, T_i starts $\frac{1}{m}(l_{i+1} + \cdots + l_x)$ time units later than she had bided l_i. Thus, the expected completion time of T_i in LS_δ is increased by $\frac{m}{m+1}(\frac{1}{m}(l_{i+1} + \cdots + l_x)) = \frac{1}{m+1}(l_{i+1} + \cdots + l_x)$. Thus, as a whole, the expected completion time of T_i cannot decrease when T_i bids a higher value than l_i, and we can deduce that LS_δ is truthful. □

81.4.2 Truthful Algorithms for Scheduling with Selfish Machines

We start this section by stating a general result, from Ref. [8], for one-parameter-agent problems whose application is illustrated in the case of the AT model.

81.4.2.1 Monotonicity

We characterize the algorithms that do and do not admit truthful payment schemes for mechanism design problems where the cost of agent i is of the form $t_i\, w_i(o)$, where t_i is her privately known cost per unit work

and $w_i(o)$ is the amount of work—or load—assigned to her. This is, for instance, the case of scheduling problems where the agents are the machines that have to bid their speeds (see Section 81.4.2.2).

The private data t_i of agent i is only known by herself, whereas everything else is public knowledge. Every agent reports some value b_i to the mechanism: this value is called the agent's bid. Let b_{-i} denote the vector of bids, not including agent i. We write the vector of bids b as (b_{-i}, b_i). The mechanism's output algorithm computes a function $o(b)$ according to the agents' bids, and tries to optimize a global objective function without knowing t directly. Each agent i incurs some monetary cost, $cost_i(t_i, o) = t_i\, w_i(o)$. To offset these costs, the mechanism makes a payment $\mathcal{P}_i(b)$ to agent i. We assume that agent i always attempts to maximize her profit: $profit_i(t_i, b) = \mathcal{P}_i(b) - cost_i(t_i, o(b))$.

Theorem 81.8 (Archer and Tardos [8])

The output function $o(b)$ admits a truthful payment scheme if and only if it is decreasing. In this case, the mechanism is truthful if and only if the payments $\mathcal{P}_i(b_{-i}, b_i)$ are of the form

$$h_i(b_{-i}) + b_i\, w_i(b_{-i}, b_i) - \int_0^{b_i} w_i(b_{-i}, u)\, du \tag{81.5}$$

where the h_i's are arbitrary functions.

Proof

Let us show this theorem with the pictorial proof of Figure 81.3.

In Figure 81.3 *(left)*, A, B, C, and D denote the areas of the rectangles they label. The cost of agent i is her privately known cost per unit work (t_i), times the amount of work assigned ($w_i(b_{-i}, b_i)$). If i's true value is y, her cost will be $B + D$ if she bids x, and $A + B + C + D$ if she bids y. Thus she would save cost $A + C$ by bidding x. If her true value is x, her cost will be B if she bids x, and $A + B$ if she bids y: she would incur an extra cost of A by bidding y. To motivate truth-telling, the extra payment for bidding y instead of x should then be at least $A + C$ and at most A, which is impossible since $C > 0$. Therefore, the work curve must decrease.

In Figure 81.3 *(right)*, the work curve is decreasing and the payments are given by Eq. (81.5). Geometrically, the payment to i if she bids x is a constant *cste* minus the area between the work curve and the horizontal line at height $w_i(x)$. If agent i's true value is t_i and she bids $x > t_i$, then her cost decreases by A (this cost is $A + C$ if the agent bids t_i whereas it is only C if she bids x), but her payment decreases by $A + B$ (this payment is *cste* $- D$ if the agent bids t_i whereas it is *cste* $- (D + A + B)$ if she bids x). Since $B > 0$, the agent never benefits from overbidding. Similarly, we can show that she never benefits from underbidding. □

81.4.2.2 A Truthful Algorithm for the AT Model

We now consider the AT model in which each machine is an agent whose secret data is her speed. We consider the case where we have two machines, and we wish to know if the LPT algorithm is truthful. The

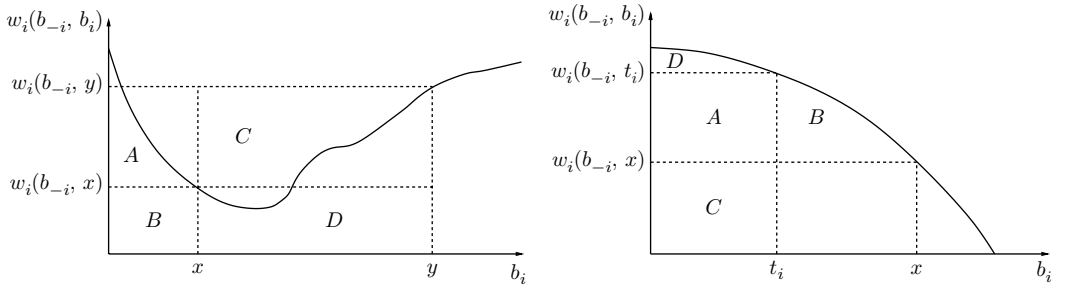

FIGURE 81.3 *Left:* The graph shows why the work curve must be decreasing. *Right:* The graph shows why agent i never gains by overbidding.

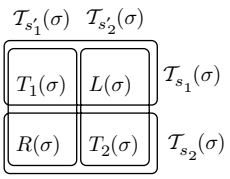

FIGURE 81.4 The set of tasks σ is partitioned into two subsets $\mathcal{T}_{s_1}(\sigma)$ and $\mathcal{T}_{s_2}(\sigma)$ (resp. $\mathcal{T}_{s'_1}(\sigma)$ and $\mathcal{T}_{s'_2}(\sigma)$) according the machine (1 or 2) on which each task is scheduled when the speed vector is s (resp. s').

LPT algorithm, in the case where the machines do not necessarily have the same speeds, is the following one: tasks are scheduled in order of decreasing lengths and each task is scheduled on the machine on which it will be completed first. Theorem 81.9 shows that LPT is truthful if the speeds of the machines are c-divisible, with $c \geq 2$, that is, if the speeds of the machines are restricted to be powers of c, where c is a constant greater than or equal to 2. It has also be shown in Ref. [17] that LPT is not truthful in the case of two machines with c-divisible speeds if $c \leq 1.78$. To prove Theorem 81.9, we will show that LPT is monotone when the speeds are c-divisible, with $c \geq 2$. By Theorem 81.8, shown in the previous section, this result implies that LPT is truthful in that case. Let us first start with the following lemma:

Lemma 81.2 (Ambrosio and Auletta [17])

For each speed vector s, where s_i is the speed of machine i, and for each sequence of tasks σ, the schedule computed by any list scheduling algorithm[2] *on input s and σ is such that for any i, j, if $s_j \geq 2\,s_i$ then $w_j \geq w_i$, where w_i is the load assigned to machine i by the algorithm.*

Proof

Suppose, by contradiction, that $w_j < w_i$ and consider the last task t assigned to machine i. We have $\frac{w_j + t}{s_j} < \frac{2\,w_i}{s_j} \leq \frac{w_i}{s_i}$, which contradicts the hypothesis that the list scheduling algorithm assigned task t to machine i. □

Consider two speed vectors $s = \langle s_1, s_2 \rangle$ and $s' = \langle s'_1, s'_2 \rangle$, where s' differs from s only on the speed of machine i and $s_i \leq s'_i$.

Let $\mathcal{T}_{s_1}(\sigma)$ be the set of tasks of σ that are assigned to machine 1 when the speed vector is s. We define $\mathcal{T}_{s_2}(\sigma), \mathcal{T}_{s'_1}(\sigma)$ and $\mathcal{T}_{s'_2}(\sigma)$ in a similar way. We define $L(\sigma) = |\mathcal{T}_{s_1}(\sigma) \cap \mathcal{T}_{s'_2}(\sigma)|$, $R(\sigma) = |\mathcal{T}_{s_2}(\sigma) \cap \mathcal{T}_{s'_1}(\sigma)|$, $T_1(\sigma) = |\mathcal{T}_{s_1}(\sigma) \cap \mathcal{T}_{s'_1}(\sigma)|$ and $T_2(\sigma) = |\mathcal{T}_{s_2}(\sigma) \cap \mathcal{T}_{s'_2}(\sigma)|$. In other words, $L(\sigma)$ is, for example, the sum of the lengths of the tasks of σ that are assigned to machine 1 with respect to s (i.e., when the speed vector is s), and to machine 2 with respect to s' (i.e., when the speed vector is s'). These notations are depicted in Figure 81.4.

In the sequel we omit the argument σ when it is clear from the context.

Theorem 81.9 (Ambrosio and Auletta [17])

For any $c \geq 2$, the algorithm LPT is monotone when restricted to the case of two machines with c-divisible speeds.

Proof

Suppose, by contradiction, that LPT is not monotone for c-divisible speeds. Then, there exists two speed vectors $s = \langle s_1, s_2 \rangle$, and $s' = \langle s'_1, s'_2 \rangle$, where s' has been obtained from s by increasing only one speed (i.e., $s'_i \geq c\,s_i$ and $s'_j = s_j$ for $i, j \in \{1, 2\}$ and $i \neq j$), and a sequence of tasks $\sigma = \langle \sigma', t \rangle$ such that the scheduling of the tasks in σ computed by LPT with respect to s and s' is not monotone. Without

[2] A *list scheduling algorithm* is an algorithm, which schedules the tasks following the order of an arbitrary priority list without leaving any unnecessary idle time.

loss of generality, assume that σ is the shortest sequence that LPT schedules in a not monotone way: the schedule of σ' is monotone while the allocation of t destroys the monotonicity. We distinguish three cases.

• Let us first consider the case where $s'_1 = s_1$ and $s'_2 \geq c\, s_2$. Since the schedule of σ' is monotone, we know that $w_2(\sigma', s) \leq w_2(\sigma', s')$, that is, $|\mathcal{T}_{s_2}(\sigma')| \leq |\mathcal{T}_{s'_2}(\sigma')|$, and so $R(\sigma') \leq L(\sigma')$ (see Figure 81.4).

Likewise, since the schedule of σ is not monotone, $w_2(\sigma, s) > w_2(\sigma, s')$, and so we know that t is on machine 2 with respect to s, and t is not on machine 2 with respect to s', meaning that the length of task t contributes to the quantity $R(\sigma)$. One can deduce that $L(\sigma') < R(\sigma') + t$. Thus,

$$R(\sigma') \leq L(\sigma') < R(\sigma') + t \tag{81.6}$$

Since t is the smallest task of σ, $R(\sigma')$ and $L(\sigma')$ are either nil, or greater than or equal to t. Hence,

$$R(\sigma') \geq \frac{L(\sigma')}{2} \tag{81.7}$$

Since the schedule of σ' is monotone, while the one of σ is not monotone, we know that t is on machine 2 with respect to s, and t is on machine 1 with respect to s'. By definition of LPT, since LPT on input s assigns task t to machine 2, we know that $\frac{T_1(\sigma')+L(\sigma')+t}{s_1} \leq \frac{T_2(\sigma')+R(\sigma')+t}{s_2}$, and so

$$T_2(\sigma') \leq \frac{s_2}{s_1}(T_1(\sigma') + L(\sigma') + t) - R(\sigma') - t \tag{81.8}$$

Similarly, since LPT on input s' assigns t to machine 1, $\frac{T_1(\sigma')+R(\sigma')+t}{s'_1} \leq \frac{T_2(\sigma')+L(\sigma')+t}{s'_2}$, from which we obtain

$$T_2(\sigma') + L(\sigma') + t \geq \frac{s'_2}{s_1}(T_1(\sigma') + R(\sigma') + t) \tag{81.9}$$

since $s_1 = s'_1$. We deduce from inequalities (81.8) and (81.9) that

$$\begin{aligned}
L(\sigma') &\geq \frac{s'_2}{s_1}(T_1(\sigma') + R(\sigma') + t) - T_2(\sigma') - t \\
L(\sigma') &\geq \frac{s'_2}{s_1}(T_1(\sigma') + R(\sigma') + t) - \frac{s_2}{s_1}(T_1(\sigma') + L(\sigma') + t) + (R(\sigma') + t) - t \\
L(\sigma') &\geq \frac{2\, s_2}{s_1}(T_1(\sigma') + R(\sigma') + t) - \frac{s_2}{s_1}(T_1(\sigma') + L(\sigma') + t) + R(\sigma') \\
L(\sigma') &\geq \frac{s_2}{s_1} T_1(\sigma') + \frac{s_2}{s_1}(2\, R(\sigma') + 2\, t - L(\sigma') - t) + R(\sigma') \\
L(\sigma') &\geq \frac{s_2}{s_1} T_1(\sigma') + \frac{s_2}{s_1}(2\, R(\sigma') - L(\sigma')) + t\,(\frac{s_2}{s_1} - 1) + (R(\sigma') + t) \\
L(\sigma') &\geq (R(\sigma') + t) \qquad \text{since } s_2 \geq s_1 \text{ and using inequality (81.7)}
\end{aligned}$$

This last inequality contradicts Eq. (81.6), and thus there is no instance σ for which the schedule computed by LPT is not monotone.

• The case $s_2 \geq s'_1 \geq c\, s_1$ can be reduced to the previous case by observing that since the scheduling of σ with respect to s and s' is not monotone, $w_1(\sigma, s) > w_1(\sigma, s')$ and therefore $w_2(\sigma, s) < w_2(\sigma, s')$. The schedules computed by LPT with respect to s and s' are equal to the schedules computed with respect to speed vectors $\langle 1, \frac{s_2}{s_1}\rangle$ and $\langle 1, \frac{s_2}{s'_1}\rangle$, where $\frac{s_2}{s'_1} \leq c\, \frac{s_2}{s_1}$.

• Case where $s'_1 > s_2$: since s_2 and s'_1 are by hypothesis powers of $c \geq 2$, and since $s'_1 > s_2$, then $s'_1 \geq 2\, s_2$, and this case follows directly from Lemma 81.2. □

This result together with Theorem 81.8 shows that LPT is truthful for the AT model.

81.5 Other Results

In this section, we give some of the numerous results in this area classified by model. Notice the existence of other surveys [18,19] and of a recent book dealing with the price of anarchy in the context of selfish routing [20].

81.5.1 Results for the KP Model and Variants

The proof of Theorem 81.1, stating that the price of anarchy for the KP model in the case of two identical parallel machines is equal to 3/2, is due to Koutsoupias and Papadimitriou [6]. They have also pointed out the relation between the problem of computing the price of anarchy for the KP model and the BINS AND BALLS problem (see, e.g., Ref. [21]). In this way, they were able to show that the price of anarchy for the KP model with m identical machines is at least $\Omega(\frac{\log m}{\log\log m})$ and is at most $3 + \sqrt{4m \log m}$. They also obtained that for the case of the KP model with uniform machines (where the machines can have different speeds) the price of anarchy is at least $\phi = \frac{1+\sqrt{5}}{2} = 1.618$ in the case of two machines, and $O(\sqrt{\frac{s_1}{s_m}\sum_{j=1}^{m}\frac{s_j}{s_m}}\sqrt{\log m})$ in the case of m uniform machines with speeds $s_1 \geq s_2 \geq \cdots \geq s_m$. They have also conjectured that the price of anarchy for the KP model is $\Theta(\frac{\log m}{\log\log m})$ [6].

The first advance in the direction of settling the validity of this conjecture has been made by Mavronicolas and Spirakis [22], who considered a special class of Nash equilibria, the so-called *fully mixed* equilibria where for every pair of task i and machine j, p_i^j is nonzero. They proved that for this class of equilibria the conjecture of Koutsoupias and Papadimitriou is valid for the KP model with identical machines. They further considered the case of the KP model with m uniform machines and n tasks of the same length (with $m \leq n$) and they established that the price of anarchy is $\Theta(\frac{\log n}{\log\log n})$. Koutsoupias et al. [23] showed that the price of anarchy is $O(\frac{\log m}{\log\log m})$ for the KP model with identical machines. But the conjecture has been completely settled for the more general KP model with uniform machines by Czumaj and Vöcking [24], who showed that the price of anarchy is indeed $\Theta(\frac{\log m}{\log\log\log m})$.

Two important classes of Nash equilibria have been extensively studied for the KP model, namely the *pure* and the *fully mixed* Nash equilibria. The obtained results concern the evaluation of the price of anarchy, as well as the computation of the *best*, *worst*, or of *just a* Nash equilibrium within these classes.

Pure Equilibria

In the case where the agents are not allowed to randomize their strategies, the set of solutions for the KP model is the set of all *pure* Nash equilibria, that is where for every agent i and machine j, p_i^j is either 0 or 1. Several questions have been treated concerning this kind of equilibria: the first one asks if such an equilibrium always exists. It has been shown in Ref. [25] that this is true for the KP model. Another question concerns the price of anarchy when restricted to pure equilibria: Czumaj and Vöcking [24] proved two upper bounds of $\Gamma^{-1}(m) + 1 = O(\frac{\log m}{\log\log m})$ for the case of the KP model with identical machines and $O(\log \frac{s_1}{s_m})$ for the same model with uniform machines of speeds $s_1 \geq s_2 \geq \cdots \geq s_m$. They have also showed that these bounds are tight up to a constant factor. From an algorithmic point of view, the main question is how to find a Nash equilibrium: Fotakis et al. [25] showed that it is NP-hard to find the

best and worst equilibria, but, they proved that it is possible to compute a Nash equilibrium for the KP model with identical machines whose price of anarchy is $\frac{4}{3} - \frac{1}{3m}$ using the LPT algorithm of Graham [9]. This result has been strengthened by the results of Refs. [26–28]. In these papers, the authors consider the *nashification problem* for the KP model, that is, the problem of designing a polynomial-time algorithm that starting from an arbitrary schedule computes a Nash equilibrium whose social cost is not greater from the one of the original schedule. Hence, in Refs. [26,27], an $O(n \log n)$ algorithm is presented for the KP model with identical machines. In Ref. [28], an $O(m^2 n)$ algorithm is presented for the KP model with uniform machines. These results show that there is a PTAS for the problem of finding a Nash equilibrium of minimum social cost for the KP model (in both the identical and uniform machine cases). This is true since it is sufficient to start from a schedule computed by the PTAS of Hochbaum and Shmoys [29] that has to be nashified using one of the above-mentioned nashification algorithms.

Fully Mixed Equilibria

As mentioned above the first to study the class of *fully mixed Nash equilibria* were Mavronicolas and Spirakis [22]. Many researchers followed this direction with the hope that the techniques elaborated for the analysis of fully mixed Nash equilibria could be appropriately extended to the general case. Fotakis et al. [25] proposed a polynomial-time algorithm, which computes in $O(n \log n)$ time a *fully mixed Nash equilibrium*. Gairing et al. [26] conjectured that the worst Nash equilibrium, that is, the Nash equilibrium with the highest social cost, is a fully mixed Nash equilibrium. This conjecture is known as the *Fully Mixed Nash Equilibrium* conjecture. The motivation to study this conjecture is that if it was true, then computing the worst Nash equilibrium would be trivial: indeed, for the KP model there is a unique fully mixed Nash equilibrium in which each task is scheduled with probability $\frac{1}{m}$ to each machine [22]. Fotakis et al. [25] studied this conjecture and gave some partial results. They proved that the conjecture is true in the special case where there are only two tasks for the KP model with identical machines. They also proved that the social cost of the worst Nash equilibrium is less than or equal to 49.02 times the cost of any generalized fully mixed Nash equilibrium for the KP model with uniform machines. They have also studied the computational complexity of computing the social cost of a Nash equilibrium, and they showed that it is #P-complete when restricted to mixed equilibria. Furthermore, they proposed a fully polynomial randomized approximation scheme (FPRAS) to compute it for the KP model with identical machines. If the fully mixed Nash equilibrium conjecture was true, this scheme would help to approximate within any accuracy the cost of the worst Nash equilibrium. Thus the efforts continued in this direction and the conjecture was shown to be true for some other special cases: the case where the number of machines is two [11] and the case where the comparison is limited to *pure* Nash equilibria [30]. Unfortunately, Fischer and Vöcking [31] showed recently that the *fully mixed Nash equilibrium* conjecture is not true and that the ratio between the social cost of a fully mixed Nash equilibrium and the worst Nash equilibrium can be almost as bad as the price of anarchy.

Various extensions of the KP model have been considered in the literature [11,33]. Among them, we can cite the *restricted* KP model where a task is allowed to be executed only to subset of the machines [11,33], the KP model with *unrelated machines* [33] or the *Web server farm* model [34]. The main results concern, as for the KP model, the evaluation of the price of anarchy or the nashification problem.

81.5.2 Results for the CKN Model

Coordination Mechanisms

The notion of coordination mechanism, the CKN model in the context of selfish scheduling, and the results of Section 81.2.2 are from Ref. [7]. In Ref. [35], the authors have recently extended the results of Ref. [7]: they noticed the close relation between coordination mechanisms and *local search algorithms*, as well as, between the price of anarchy and the *locality gap* of a neighborhood and by doing so they were able to obtain the price of anarchy for some of the variants of the CKN model using some older results from the local search literature [36]. They also studied the price of anarchy of *pure* Nash equilibria for various

coordination mechanisms for the CKN model. They proved that the price of anarchy of any deterministic coordination mechanism for the CKN model with uniform machines and the CKN model where each task can be executed to a *restricted* subset of machines is at most $O(\log m)$. They also proved that the price of anarchy of a coordination mechanism based on a *universal policy* (like LPT or SPT) is $\Omega(\log m)$ for the later model. Furthermore, they showed that the price of anarchy of a coordination mechanism based on LPT is $2 - \frac{2}{m}$ for the CKN model with uniform machines. They also proved that a coordination mechanism based on a *randomized* policy has a price of anarchy of $\Theta(m)$ for the CKN model with unrelated machines. They also studied the convergence and existence of pure Nash equilibria for coordination mechanisms based on SPT and LPT policies.

Price of Stability

The results in Section 81.3.1 are from Ref. [12] and can be extended to the m machines case. The authors studied the trade-off between α and the quality of the proposed solution. More precisely, they showed that the *price of α-approximate stability* is at most $\frac{8}{7}$ for all $\alpha \geq 3$, and they gave an algorithm that achieves this bound. They also provided a relation between α and the best possible price of α-approximate stability, by showing, for example, that the price of α-approximate stability is at least $\frac{8}{7}$ for all $\alpha < 2.1$, and that it is larger than or equal to $1 + \varepsilon$ if α is smaller than a certain constant k in $\Theta(\varepsilon^{-1/2})$.

Truthfulness

The results of Section 81.4.1 are from Ref. [16]. The authors considered also the design of truthful coordination mechanisms for the CKN model with two identical machines. They first studied a *coordination mechanism* where the first machine always schedules its tasks in order of increasing lengths (its policy is SPT), and the second machine schedules its tasks with a probability $p > \frac{2}{3}$ in order of increasing lengths and with probability $(1 - p)$ in order of decreasing lengths. The expected price of anarchy of this *(randomized)* coordination mechanism, that they proved to be $\frac{4}{3} + \frac{p}{6}$, is better than the one of SPT (whose price of anarchy is $\frac{3}{2}$). They also showed that this coordination mechanism is truthful if the tasks are powers of a constant greater than or equal to $\frac{4-3p}{2-p}$, but not if the values of the task lengths are not restricted. Moreover, they showed that if $p < \frac{1}{2}$ then this coordination mechanism is not truthful even if the tasks are powers of any integer larger than 1. They also considered the other randomized coordination mechanisms that combine deterministic coordination mechanisms in which the tasks are scheduled in order of increasing or decreasing lengths (and thus which have expected price of anarchy better than the one of SPT), and gave negative results on their truthfulness.

81.5.3 Results for the AT Model

The results of Theorem 1.8 are from Ref. [8] and that of Theorem 1.9 from Ref. [17]. The first results in this direction were given by Nisan and Ronen [37], but for the model of selfish *unrelated* machines. For this model they proved that the former known approximation algorithms are not truthful. For the AT model (with uniform machines) it was Archer and Tardos who introduced a truthful *randomized* mechanism, which gives 3-approximate solutions. The first deterministic result is due to Auletta et al. [38] who proposed a deterministic polynomial-time $(2 + \varepsilon)$-approximation algorithm and suitable payments functions that yield truthful mechanisms for the following restrictions of the problem: (i) the speeds of the machines are integer and the largest is bounded from above by a constant, and (ii) the speeds are divisible, that is, s_{i+1} is a multiple of s_i. The proposed mechanisms compute the payments in polynomial time and satisfy *voluntary participation*, that is, a truthfully behaving agent never incurs in a loss. They were also able to deduce a deterministic truthful $(4 + \varepsilon)$-approximate mechanism for the case of *arbitrary speeds* and for any, but *fixed*, number of machines. More recently, Azar and Sorani [39] improved these results: They provided a deterministic 12-approximation truthful mechanism for the AT model with an *arbitrary* number of processors. They also proposed a deterministic truthful PTAS for the AT model with a *fixed* number of machines. Auletta et al. [40] studied the difficulty of translating approximation/competitive algorithms into equivalent approximation/competitive truthful mechanisms. They proposed "translation" technique and studied its limits.

References

[1] Papadimitriou, C., Algorithms, games and the Internet, *Proc. STOC*, 2001, p. 749.

[2] Nash, J. F., Non cooperative games, *Ann. Math.*, 54, 286, 1951.

[3] Osborne, J. O. and Rubinstein, A., A *Course in Game Theory*, The MIT Press, Cambridge, MA, 1994.

[4] Lipton, R. J., Markakis, E., and Mehta, A., Playing large games using simple strategies, *Proc. 4th ACM Conf. on Electronic Commerce,* 2003, p. 36.

[5] Papadimitriou, C. and Roughgarden, T., Computing equilibria in multi-player games, *Proc. SODA*, 2005, p. 82.

[6] Koutsoupias, E. and Papadimitriou, C., Worst-case equilibria, *Proc. STACS*, Lecture Notes in Computer Science, Vol. 1563, 1999, Springer, Berlin, p. 404.

[7] Christodoulou, G., Koutsoupias, E., and Nanavati, A., Coordination mechanisms, *Proc. ICALP*, Lecture Notes in Computer Science, Vol. 3142, Springer, Berlin, 2004, p. 345.

[8] Archer, A. and Tardos, E., Truthful mechanisms for one-parameter agents, *Proc. FOCS*, 2001, p. 482.

[9] Graham, R., Bounds on multiprocessor timing anomalies, *SIAM J. Appl. Math.*, 17(2), 416, 1969.

[10] Anshelevich, E., Dasgupta, A., Kleinberg, J., Tardos, É., Wexler, T., and Roughgarden, T., The price of stability for network design with fair cost allocation, *Proc. FOCS*, 2004, p. 295.

[11] Gairing, M., Lücking, T., Mavronicolas, M., and Monien, B., Computing Nash equilibria for scheduling on restricted parallel links, *Proc. STOC,* 2004, p. 613.

[12] Angel, E., Bampis, E., and Pascual, F., The price of approximate stability for a scheduling game problem, Euro-Par, LNCS, 2006.

[13] Clarke, E., Multipart pricing of public goods, *Public Choices*, 11, 17, 1971.

[14] Groves, T., Incentive in teams, *Econometrica*, 41(4), 617, 1973.

[15] Vickrey, W., Counterspeculation, auctions and competitive sealed tenders, *J. Finance*, 16, 8, 1961.

[16] Angel, E., Bampis, E., and Pascual, F., Truthful algorithms for scheduling selfish tasks on parallel machines, *Proc. WINE*, Lecture Notes in Computer Science, Vol. 3828, Springer, Berlin, 2005, p. 698.

[17] Ambrosio, P. and Auletta, V., Deterministic monotone algorithms for scheduling on related machines, *Proc. WAOA*, Lecture Notes in Computer Science, Vol. 3351, Springer, Berlin, 2004, p. 267.

[18] Czumaj, A., Selfish routing on the Internet, in *Handbook of Scheduling: Algorithms, Models and Performance Analysis,* Leung, J., Ed., CRC Press, Boca Raton, FL, 2004, chap. 42.

[19] Feldmann, R., Gairing, M., Lücking, T., Monien, B., and Rode, M., Selfish routing in non-cooperative networks: a survey, *Proc. MFCS*, Lecture Notes in Computer Science, Vol. 2747, Springer, Berlin, 2003, p. 21.

[20] Roughgarden, T., Selfish Routing and the Price of Anarchy, The MIT Press, Cambridge, MA, 2005.

[21] Gonnet, G., Expected length of the longest probe sequence in hash code searching, *JACM*, 28(2), 289, 1981.

[22] Mavronicolas, M. and Spirakis, P., The price of selfish routing, *Proc. STOC*, 2001, p. 510.

[23] Koutsoupias, E., Mavronicolas, M., and Spirakis, P., Approximate equilibria and ball fusion, *Theor. Comput. Syst.*, 36(6), 683, 2003.

[24] Czumaj, A. and Vöcking, B., Tight bounds for worst-case equilibria, *Proc. SODA*, 2002, p. 413.

[25] Fotakis, D., Kontogiannis, S., Koutsoupias, E., Mavronicolas, M., and Spirakis, P., The structure and complexity of Nash equilibria for a selfish routing game, *Proc. ICALP*, Lecture Notes in Computer Science, Vol. 2380, Springer, Berlin, 2002, p. 123.

[26] Gairing, M., Lücking, T., Mavronicolas, M., Monien, B., and Spirakis, P., Extreme Nash equilibria, *Proc. ICTCS*, Lecture Notes in Computer Science, Vol. 2841, Springer, Berlin, 2003, p. 1.

[27] Even-Dar, E., Kesselman, A., and Mansour, Y., Convergence time to Nash equilibria, *Proc. ICALP*, Lecture Notes in Computer Science, Vol. 2719, Springer, Berlin, 2003, p. 502.

[28] Feldmann, R., Gairing, M., Lücking, T., Monien, B., and Rode, M., Nashification and the coordination ratio for a selfish routing game, *Proc. ICALP*, Lecture Notes in Computer Science, Vol. 2719, Springer, Berlin, 2003, p. 514.

[29] Hochbaum, D. and Shmoys, D., A polynomial approximation scheme for scheduling in uniform processors: using the dual approximation scheme, *SIAM J. Comput.*, 17(3), 539, 1998.
[30] Gairing, M., Lücking, T., Mavronicolas, M., Monien, B., and Spirakis, P., Structure and complexity of extreme Nash equilibria, *Theor. Comp. Sci.*, 343(1–2), 133, 2005.
[31] Fischer, S. and Vöcking, B., On the structure and complexity of worst-case equilibria, *Proc. WINE,* Lecture Notes in Computer Science, Vol. 3828, Springer, Berlin, 2005, p. 151.
[32] Ferrante, A. and Parente, M., Existence of Nash equilibria in selfish routing problems, *Proc. SIROCCO,* Lecture Notes in Computer Science, Vol. 3104, Springer, Berlin, 2004, p. 149.
[33] Awerbuch, B., Azar, Y., Richter, Y., and Tsur, D., Tradeoffs in worst-case equilibria, *Proc. WAOA*, Lecture Notes in Computer Science, Vol. 2909, Springer, Berlin, 2003, p. 64.
[34] Czumaj, A., Krysta, P., and Vöcking, B., Selfish traffic allocation for server farms, *Proc. STOC*, 2002, p. 287.
[35] Immorlica, N., Li, L., Mirrokni, V. S., and Schulz, A., Coordination mechanisms for selfish scheduling, *Proc. WINE*, Lecture Notes in Computer Science, Vol. 3828, Springer, Berlin, 2005, p. 55.
[36] Vredeveld, T., Combinatorial Approximation Algorithms. Guaranteed versus Experimental Performance, Ph.D. thesis, Technische Universiteit Eindhoven, The Netherlands, 2002.
[37] Nisan, N. and Ronen, A., Algorithmic mechanism design, *Proc. STOC*, 1999, p. 129.
[38] Auletta, V., De Prisco, R., Penna, P., and Persiano, P., Deterministic truthful approximation mechanisms for scheduling related machines, *Proc. STACS*, Lecture Notes in Computer Science, Vol. 2996, Springer, Berlin, 2004, p. 608.
[39] Azar, Y. and Sorani, M., Truthful approximation mechanisms for scheduling selfish related machines, *Proc. STACS*, Lecture Notes in Computer Science, Vol. 3404, Springer, Berlin, 2005, p. 69.
[40] Auletta, V., De Prisco, R., Penna, P., and Persiano, P., On designing truthful mechanisms for online scheduling, *Proc. SIROCCO*, Lecture Notes in Computer Science, Vol. 3499, Springer, Berlin, 2005, p. 3.