Games and Mechanism Design in Machine Scheduling - An Introduction

Birgit Heydenreich* Rudolf Müller Marc Uetz

Maastricht University, Quantitative Economics,
P.O. Box 616, 6200 MD Maastricht, The Netherlands.
E-mail: {b.heydenreich,r.muller,m.uetz}@ke.unimaas.nl

Abstract

In this paper, we survey different models, techniques, and some recent results to tackle machine scheduling problems within a distributed setting. In traditional optimization, a central authority is asked to solve a (computationally hard) optimization problem. In contrast, in distributed settings there are several agents, possibly equipped with private information that is not publicly known, and these agents need to interact in order to derive a solution to the problem. Usually the agents have their individual preferences, which induces them to behave strategically in order to manipulate the resulting solution. Nevertheless, one is often interested in the global performance of such systems. The analysis of such distributed settings requires techniques from classical Optimization, Game Theory, and Economic Theory. The paper therefore briefly introduces the most important of the underlying concepts, and gives a selection of typical research questions and recent results, focussing on applications to machine scheduling problems. This includes the study of the so-called price of anarchy for settings where the agents do not possess private information, as well as the design and analysis of (truthful) mechanisms in settings where the agents do possess private information.

Keywords: Machine Scheduling, Game Theory, Mechanism Design

1 Introduction and Scope

Consider a problem where we are asked to distribute a stream of jobs over several production facilities that are available to process those jobs. In traditional optimization, a central decision maker would be equipped with all relevant data of that problem, asked to derive a solution that fulfills all the necessary side constraints, and optimizing some kind of global performance criterion. However, assuming that decisions are taken by several independent economic units, it might be the case that these individual units aim at optimizing their own objectives rather than the performance of the system as a whole. Such situations call for models and techniques that take the strategic behavior of individual units into account, and simultaneously keep an eye on the global performance of the system.

Strategic situations are traditionally analyzed in Game Theory as well as certain areas of Economic Theory. The game theoretic viewpoint was lately also adopted in the Computer Science

*Supported by NWO grant 2004/03545/MaGW 'Local Decisions in Decentralised Planning Environments'.

and Operations Research communities, motivated by an increasing amount of economic activity that takes place via the internet, such as electronic auctions. In recent years, this resulted in both a (re-)discovery of several classic results from Game Theory and Economic Theory, as well as an amazing wealth of new results in a quickly developing field that combines techniques from classical Optimization, Game Theory, and Economic Theory.

In this paper, we give an—admittedly subjective—introduction into typical research questions that have recently been addressed in the literature. From the application perspective, we focus on specific problems from Production and Operations Management, namely simple and classical scheduling and sequencing problems. In that perspective, the focus is not so much on actual applications in practice, but rather on the underlying game theoretic models and methodologies. We are aware that this focus is quite narrow, yet it serves well to exemplify the most important research questions that arise when addressing optimization problems from a decentralized perspective. In particular, by keeping the focus narrow from the applications point of view, we think that we are able to better highlight the most important underlying theoretical challenges.

In decentralized settings, central coordination of a system is partially replaced by decisions and actions taken by *agents*. Those agents are assumed to act rationally on behalf of their own interest, and it is generally assumed that their selfish behavior results in a situation that can be characterized by some sort of *system equilibrium*. From a global perspective, such an equilibrium may of course lead to suboptimal system performance. The following two issues arise in such settings and will be addressed in this paper.

- Given a fixed decentralized setting in which agents selfishly act on behalf of their own interest, try to *characterize and analyze* the quality of the resulting *system equilibria* from the perspective of the overall system performance.
- Try to *design* the decentralized setting in such a way that selfish agents are induced to behavior that results in system equilibria that nevertheless exhibit a good overall system performance.

Moreover, both issues can be studied in settings where the individual agents *do* or *do not* have private information. The distinction between settings with or without private information leads to different challenges and related research questions. In fact, the paper is structured along this distinction.

In settings *without private information*, also called *complete information* settings, the wealth of the literature is of a descriptive nature and addresses the issue to characterize and analyze equilibria of given systems. Only to a lesser extent the actual design of such settings is addressed. The analysis of system equilibria leads to the definition of the so-called *price of anarchy* or *coordination ratio*: Caused by selfish behavior of agents, by how much does the overall system performance deteriorate due to a lack of central coordination? In the literature on the price of anarchy, it is generally assumed that all data that describes the problem is publicly known. The 'only' complication is caused by the fact that agents act on their own behalf. The agents thus need to take into account the (strategic) behavior of other agents. The underlying equilibrium concept is the Nash equilibrium. Settings without private information and the analysis of the price of anarchy will be addressed in Section 3. The models and results discussed in Section 3 are mainly from the work of Koutsoupias and Papadimitriou (1999), Czumaj and Vöcking (2002), and Immorlica, Li, Mirrokni, and Schulz (2005). When it comes to the design of complete information settings, one is concerned with defining the rules within which the agents may interact. In this paper, we give several examples of system designs for machine scheduling problems, and discuss the resulting price of anarchy.

In order to improve the quality of resulting equilibria in complete information settings, one can augment the system by introducing *payments*. In the context of a network routing problem,

the issues that might arise with introducing payments have been addressed, for example, by Cole, Dodis, and Roughgarden (2006). Another option to improve the quality of system equilibria is to centrally control a certain fraction of the agents, leading to so-called *Stackelberg* games. Such a model was analyzed for example by Roughgarden (2004) in the context of scheduling. We refer to those papers for an introduction into these and related issues. In this paper, we will not further elaborate on such extensions of complete information settings.

In settings *with private information*, we immediately arrive at a research field that is also known as *algorithmic mechanism design*; a term that was coined by Nisan and Ronen (2000). In these settings, the additional complication is that the agents own some piece of private information that is not publicly known. In order to be able to run and evaluate the system, the agents need to reveal this private information to the system. Hence, in addition to their (strategic) behavior within the system itself, agents might be tempted to falsely report their private information if it is beneficial for their own objectives. One important part of the design of such systems is therefore to induce the agents to *truthfully* report their private information; sometimes also called the design of *truthful mechanisms*. Notice that the equilibria concepts in models with private information are more complex, because each agent is faced with the additional uncertainty about the private information of the other agents. Algorithmic mechanism design problems are addressed in Section 4. The specific models and results discussed in Section 4 are based on the work of Archer and Tardos (2001), Nisan and Ronen (2000), Porter (2004), and Heydenreich, Müller, and Uetz (2006).

As mentioned before, the scope of this paper is not to give an exhaustive overview of the field, but to highlight some typical research questions. Hence, we have chosen to discuss only a subjective selection of recent papers. Other references from Computer Science not explicitly discussed here are, for example, Christodoulou, Koutsoupias, and Nanavati (2004) and Angel, Bampis, and Pascual (2005). Related problems are also studied in the literature on Economic Theory, addressing questions on the efficient organization of queues. There are, for example, papers on the existence of mechanisms with more properties than only truthfulness (Mitra 2001, 2005), or where queue disciplines are organized with the help of auctions (Kittsteiner and Moldovanu 2005).

The paper is structured as follows. In Section 2.1, we introduce basic notation and terminology for the scheduling models that will be addressed. In Section 2.2 we give a survey of the most basic concepts and terminology in game theory and mechanism design that will be used throughout the paper. Section 3 then addresses the analysis of the price of anarchy in settings without private information, and Section 4 addresses the design and analysis of (truthful) mechanisms in different settings with private information.

2 Basic Concepts and Notation

2.1 Basic Scheduling Models

We consider machine scheduling problems with the following characteristics. There is a set of jobs $J = \{1, \dots, n\}$, and each job has to be scheduled on any machine out of a set of machines $M = \{1, \dots, m\}$. Unless explicitly stated otherwise, jobs must be scheduled *non-preemptively*, meaning that once the processing of a job has started, it cannot be interrupted until the job is completed. Regarding the machines we distinguish between three different models:

- In *parallel* machine scheduling, each job $j \in J$ has processing time $p_j > 0$, independent of the machine that processes the job.
- In *related* (or *uniform*) machine scheduling, each job j has processing time p_j (on a unit speed machine), each machine $i \in M$ has a speed $s_i > 0$, and the processing time of job j on

machine i equals p_j/s_i.

- In *unrelated* machine scheduling, each job $j \in J$ has processing time $p_{ij} > 0$ when scheduled on machine $i \in M$.

In addition, jobs may have different characteristics depending on the specific model that is addressed. We only mention the two most important characteristics here. A *release date* $r_j \geq 0$ of job j is the time when job j comes into existence or is released for processing. In models with *deadlines*, each job j should be completed by its deadline d_j, and a job which is completed before its deadline is called *early*, otherwise a job is called *late*.

A feasible *schedule* is an assignment of jobs to machines, together with the specification of the time interval(s) when the job is processed. In non-preemptive settings, this reduces to specifying the machine and start time S_j for any job j. The precise definition of *feasibility* clearly depends on the particular model, but always comprises the requirement that each job must be completely processed and no machine can process more than one job at a time. If jobs have release dates, for example, no job must be started before its release date r_j.

With respect to the objective of scheduling, we address several, classical objectives, which will always depend on the completion times of the jobs. Given a schedule, denote by S_j and C_j the start time and completion time of job j, respectively. Then the *makespan* of a schedule is the latest job completion time, denoted by $C_{\max} := \max_j C_j$. Jobs also might have *weights* $w_j \geq 0$, denoting a priority for being processed early. Then the *total weighted completion time* is $\sum_{j \in J} w_j C_j$. These job weights w_j could, for example, be deducted from an inventory value, and they can be interpreted as opportunity costs for delaying job j one unit of time.

Most models that we address represent NP-hard combinatorial optimization problems; for a survey and references, see, for example, the paper by Lawler, Lenstra, Rinnooy Kan, and Shmoys (1993). In addition, we address scheduling models that are *online*, thus the complete problem instance is not given at the outset, but only revealed gradually over time. For example, the existence of jobs might only become known upon their release dates r_j. For an introduction to online scheduling problems and models, see, for example, the paper by Pruhs, Sgall, and Torng (2004).

It should be mentioned that research in scheduling has addressed many more features and models than discussed here. For example, there might be precedence constraints between jobs, saying that the processing of job j may only start after another job i has been finished. Or the processing of jobs might need multiple resources, rather than one machine, and resources may be non-renewable. Also, there are other objectives than those considered here. We have decided to leave these models out of consideration, because—to the best of our knowledge—the combination of optimization and game theory has only been applied to machine scheduling models.

2.2 Concepts in Game Theory and Mechanism Design

We next define some basic notation for game theoretic concepts used throughout the paper. In addition, we introduce problem specific notation and concepts when needed, and indicate when we deviate from the game theoretic notation introduced here.

A crucial element in game theoretic settings is the fact that we have a set of interacting *agents*. In the scheduling models we address, this will either be the set of jobs J or the set of machines M. Let us say we have ℓ agents, then either $\ell = n$ or $\ell = m$. In some settings, an agent $k \in \{1, \ldots, \ell\}$ may own a piece of information that is not publicly known, its *type* t_k. Typical types are, for example, the speed s_i of a machine-agent $i \in M$, or the weight w_j of a job-agent $j \in J$. The possible types for agent k are denoted by T_k. Furthermore, let $T = T_1 \times \cdots \times T_\ell$ denote the type

space of all agents. Next to the private information of agents, there is usually public information, as for example the number of machines or the type space of the agents (though not their actual types).

In a game, agents have to choose between several possible *actions*. An action could be that jobs have to select a machine on which they want to be processed, or that machines have to report their actual speed. We denote by A_k the possible actions of agent k and by $A = A_1 \times \cdots \times A_\ell$ the action space of all agents. The *outcome* of the game depends on the actions of all agents. In the games we consider, the outcome will always be a (feasible) schedule. Therefore, by Y we denote all (feasible) schedules.

Some care is required in order to translate 'problem instances' and 'algorithms' to a game theoretic setting. First, the term 'problem instance' that is used in optimization refers to both the public and the private information of a game. Let us denote by I the public information of a game. Then the equivalent of an algorithm is usually called an *allocation algorithm*, denoted by α; it computes an outcome (a schedule) on the basis of the public information I together with the actions of all agents. More precisely, $\alpha : I \times A \to Y$. Since there is hardly danger of ambiguity, we usually omit the public information I and write $\alpha\colon A \to Y$. To give an example, suppose that the jobs are agents and that their action is to select a machine. Then, $A = M^n$ is the action space, and the public information I consists of m, the number of machines, n, the number of jobs, as well as the set of processing times of all jobs $\{p_j \mid j \in J\}$. Assume that the allocation of jobs to time slots is defined by the *Local SPT rule*: Each machine processes its jobs in the order of non-decreasing processing times (SPT, shortest processing time first). For given actions $a = (a_1, \ldots, a_n)$ of all n jobs, the allocation algorithm α thus assigns job k to machine a_k, in such a way that k is processed after all jobs j with $a_j = a_k$ and with $p_j < p_k$. (To make the game unambiguous, a tie breaking rule would be required for jobs with equal processing times assigned to the same machine. We assume that ties are broken in favor of jobs with smaller index j.) Notice that jobs are informed about the public information I, such as the number of available machines and the processing times of other jobs.

Agents will appreciate different outcomes of the game (i.e., schedules) differently. We express this by the *valuation* of an agent for a certain schedule. Moreover, the valuation of agent k for a schedule $y \in Y$ might depend on its actual type t_k. Therefore, agent k's valuation for outcome y is usually denoted by $v_k(y|t_k)$. Here, schedule $y = \alpha(a)$ depends on the actions a of all agents. If the allocation algorithm α is clear from the context, we also write $v_k(a|t_k)$, for convenience. If schedule y is preferred over y' by agent k (being of type t_k), we assume that $v_k(y|t_k) > v_k(y'|t_k)$. Thus the higher the valuation the better. For a job-agent j, for example, the valuation for a certain schedule might be $-C_j$, meaning that the job-agent wants to be finished as early as possible.

Given the public information, an agent's choice of an action depends on its type. Therefore, we need to define the *strategy* x_k of an agent k as a mapping from the agent's type space into its action space. Let $X_k = \{x_k \,|\, x_k\colon T_k \to A_k\}$ denote the strategy space of agent k and let $X = X_1 \times \cdots \times X_\ell$ be the possible strategies of all agents. For example, suppose a job j has to choose for being processed on one of two machines with different speeds, say machine 1 with speed $s_1 = 1$ and machine 2 with speed $s_2 = 2$. Suppose further that the job could be processed immediately on the slow machine 1, whereas it has to wait one time unit until the fast machine 2 becomes available. Assume that the type t_j of job j is just its processing time p_j (on a unit speed machine), and its valuation for an outcome (a schedule) is $-C_j$. Then the job's preferred strategy would be to choose the slow machine 1 if $p_j \leq 2$, but to wait for the fast machine 2 if $p_j > 2$.

As a central authority, we evaluate the overall quality of a schedule by the objective value that it achieves. Clearly, the agent's choices of actions influence the quality of the schedule. In order to induce agents to choose their actions in a way that is favorable for the overall quality of a schedule,

it is common to manipulate the agents by introducing *payments*. Such payments depend on the actions of all agents and specify for each agent how much (money) is to be payed (or received) by that agent. Given the actions a of all agents, let $\pi_k(a)$ denote the required payment for agent k. This could be both positive or negative. The overall *payment scheme* π is then a mapping from the action space A to the space of all possible payments. Assuming we have ℓ agents, we thus have $\pi : A \to \mathbb{R}^\ell$. (More precisely, we should write $\pi : I \times A \to \mathbb{R}^\ell$.)

Payments clearly change the appreciation of an agent for a certain outcome (i.e., schedule). Therefore, the preferences of agents over the possible schedules will depend on their valuations for the schedule *as well as* the associated payments. In the models we address, we express the relation of valuations to payments by so-called *quasi-linear utilities*. That means that the *utility* u_k that an agent k receives from a schedule is just the valuation minus the payment. More precisely, if a schedule $y = \alpha(a)$ is computed by some allocation algorithm α on the basis of actions a of all agents, with associated payments $\pi(a)$, then the utility of agent k (being of type t_k) is given by $u_k(\alpha(a)|t_k) = v_k(\alpha(a)|t_k) - \pi_k(a)$. Finally, notice that we assume that agents are *rational*; meaning that they aim at maximizing their utilities.

3 Models with Complete Information

When agents do not have any private information, we talk about *games with complete information*. In these settings, a strategy of an agent is simply the choice of an action, and it does not depend on any private information. Therefore, we can identify strategies X_k with actions A_k for every agent. (Recall that in models where agents have private types t_k, a strategy $x_k \in X_k$ maps possible types from T_k to actions in A_k.) As it is common practice in game theory, we will adopt the term strategy for the actions of agents, and we will use X_k instead of A_k. A game is then simply a mapping from the set of strategies of the agents to the set of schedules, coinciding with the allocation algorithm α defined earlier. An agent k's valuation for a schedule $y \in Y$ can be written simply as $v_k(y)$, because it does not depend on a potential type t_k of that agent. Since the schedule y only depends on the agent's strategies, the valuation can also be expressed as the valuation for a certain strategy vector $v_k(x)$ for $x \in X$.

In a game with payments, we can compute the utility of an agent k from its valuation for a certain strategy vector x and its payment given that strategy vector x as $u_k(x) = v_k(x) - \pi_k(x)$. In a game without payments, an agent's utility equals its valuation; we use the term utility also in that case.

In general, agents are also allowed to play *mixed strategies*. A mixed strategy of an agent k is a probability distribution over the set of its *pure strategies* X_k. We denote the set of probability distributions over the pure strategy set X_k by $\Delta(X_k)$. For a given vector of mixed strategies, the utilities for the individual agents as well as the objective function value become random variables. A Nash equilibrium is then defined as follows.

Definition 1. *A strategy vector* $x = (x_1, \dots, x_\ell) \in \Delta(X_1) \times \cdots \times \Delta(X_\ell)$ *is called* Nash equilibrium *if for every agent* $k = 1, \dots, \ell$

$$\mathbb{E}[u_k(x)] \geq \mathbb{E}[u_k(x_1, \dots, x_{k-1}, x'_k, x_{k+1}, \dots, x_\ell)] \qquad \forall x'_k \in \Delta(X_k).$$

Here, $\mathbb{E}[\,\cdot\,]$ denotes the expectation. In a model where only pure strategies are allowed, this definition reduces to the following.

Definition 2. *A strategy vector* $x = (x_1, \dots, x_\ell) \in X_1 \times \cdots \times X_\ell$ *is called* pure strategy Nash equilibrium *if for every agent* $k = 1, \dots, \ell$

$$u_k(x) \geq u_k(x_1, \dots, x_{k-1}, x'_k, x_{k+1}, \dots, x_\ell) \qquad \forall x'_k \in X_k.$$

In general, Nash equilibria in pure strategies do not necessarily exist. Existence of Nash equilibria is only guaranteed if agents are allowed to play mixed strategies. Therefore, an interesting question is the existence of pure strategy Nash equilibria for a given problem. Moreover, one is interested in algorithms to compute pure or mixed strategy Nash equilibria (efficiently).

A third issue that is addressed in the literature that is specific to games with complete information is the following question. How does the objective value that results from a Nash equilibrium—thus a solution induced by utility maximizing selfish agents—compare to the optimal objective value. The latter might just be computed by some central authority. The extent to which the objective value deteriorates due to the *lack of central coordination* is called the *price of anarchy*. It can be defined for pure as well as for mixed strategy settings.

Definition 3. *For a minimization problem, let V_{OPT} be the optimal objective value and let V_{NE} be the worst possible objective value achieved by any (pure-strategy) Nash equilibrium. Then the* price of anarchy (*of pure Nash equilibria*) *is defined as*

$$POA = \frac{V_{NE}}{V_{OPT}}.$$

Accordingly, one defines the price of anarchy as V_{OPT}/V_{NE} in a maximization problem. The study of the price of anarchy as the worst case ratio between the objective value of a Nash equilibrium and that of the overall system optimum was initiated by Koutsoupias and Papadimitriou (1999). They were motivated by the fact that Nash equilibria in general do not optimize the overall performance of the system, the most prominent example being the Prisoner's Dilemma, see e.g. (Owen 1995). In a part of the literature the price of anarchy is also referred to as *coordination ratio*.

The following sections highlight a sample of different scheduling settings, their Nash equilibria, and the corresponding prices of anarchy.

3.1 The Price of Anarchy in Congestion Models

We first define the model as described and analyzed by Koutsoupias and Papadimitriou (1999). Consider n job-agents $j \in J = \{1, \ldots, n\}$ with processing times p_j that have to be processed on m machines $i \in M = \{1, \ldots, m\}$ with possibly different speeds $s_i > 0$. This is the related machine scheduling model, and if all speeds s_i are equal, the parallel machine scheduling model. Each pure strategy of an agent corresponds to the deterministic selection of one of the machines. A mixed strategy of agent j assigns a probability q_i^j to every machine i, such that $\sum_{i=1}^m q_i^j = 1$ for all j.

We call the model *congestion model* due to the following assumption. It is assumed that the valuation of any job for a given schedule is determined by the total processing time of the jobs assigned to the same machine. Stated otherwise, jobs are released from a machine only when the machine has finished *all* the jobs assigned to it. We therefore define the utility of job j with strategy $i \in M$ as follows. For any vector of pure strategies $(i, i_{-j}) := (i_1, \ldots, i_{j-1}, i, i_{j+1}, \ldots, i_n) \in \{1, \ldots, m\}^n$,

$$u_j(i, i_{-j}) = -\frac{1}{s_i} \sum_{k: i_k = i} p_k.$$

The expected utility of agent j when the mixed strategy vector $(q^1, \ldots, q^n)$ is played is then

$$\mathbb{E}[u_j(q^1, \ldots, q^n)] = -\sum_{i=1}^m \frac{q_i^j}{s_i} \left(p_j + \sum_{k \neq j} q_i^k p_k \right).$$

The objective of the central authority is to minimize the makespan of the overall schedule, i.e.

$$V_{OPT} = \min_{i_1,\ldots,i_n} \max_i \frac{1}{s_i} \sum_{j:i_j=i} p_j.$$

The model was originally motivated by regarding the machines as network links and the jobs as traffic that has to be routed via the links. The utility of each agent is then defined by the delay it experiences when being routed via a specific link, caused by the corresponding total congestion of that link. The utilities as defined above are therefore also called *linear cost functions*, as the congestion depends linearly on the total load assigned to that link.

For the case with two identical machines, i.e., machines with equal speeds, it was shown that the price of anarchy (for mixed strategies) is equal to 3/2 (Koutsoupias and Papadimitriou 1999). We present their example showing why the price of anarchy is at least 3/2.

Example 4. *Consider two jobs with $p_1 = p_2 = 1$ and two machines with unit speed. Then a mixed Nash equilibrium is the choice $q_i^j = 1/2$ for $i, j = 1, 2$. In that Nash equilibrium, both jobs choose the same machine with probability 1/2, resulting in makespan 2. With probability 1/2, the jobs are processed by different machines, which gives a makespan of 1. Therefore, the expected objective value is 3/2. The optimum is to assign both jobs to different machines, yielding an objective value of 1. Therefore the price of anarchy for minimizing the schedule makespan in the congestion model is at least 3/2.*

A matching upper bound of 3/2 for the price of anarchy can be derived as well.

Theorem 5 (Koutsoupias and Papadimitriou 1999)**.** *For $m = 2$ identical machines, the price of anarchy for minimizing the makespan in the congestion model is 3/2.*

Let us briefly summarize further (and more general) results by Koutsoupias and Papadimitriou (1999) and Czumaj and Vöcking (2002).

For an arbitrary number m of identical machines, Koutsoupias and Papadimitriou (1999) show that the POA is at least $\Omega(\log m/(\log\log m))$. This result goes back to the classical *bins-and-balls* result: When throwing m balls into m bins uniformly at random, then the expected maximum number of balls in any bin is $\Theta(\log m/(\log\log m))$. To translate this into the given setting, consider the case with m machines and m jobs with unit processing times. One can check that there is a Nash equilibrium where every job randomizes uniformly over all machines. In this Nash equilibrium, the expected makespan is $\Theta(\log m/(\log\log m))$, due to the bins-and-balls result. In the optimal solution, however, each machine processes exactly one job, yielding a makespan of 1. The claimed lower bound on the POA follows. Czumaj and Vöcking (2002) establish a matching upper bound of $\mathcal{O}(\log m/(\log\log m))$ for the POA on m machines; they even give an exact expression for the price of anarchy for that case.

For the case with two *related* machines (two machines with different speeds), the POA is at least as large as the golden ratio $\phi \approx 1.618$ (Koutsoupias and Papadimitriou 1999). They also conjecture that the POA for two machines with different speeds is not more than $\phi \approx 1.618$; yet this remains an open question. For the more general case with m related machines, Czumaj and Vöcking (2002) show that the price of anarchy is in $\Theta(\log m/(\log\log\log m))$. This completes the picture for models with linear cost functions and identical or related machines. For other extensions (e.g., non-linear congestion models), we refer to the survey by Czumaj (2004).

Clearly, the price of anarchy when mixed strategies are allowed is at least as large as the price of anarchy when only pure Nash equilibria are considered. Pure strategy Nash equilibria are not

analyzed by Koutsoupias and Papadimitriou (1999), neither by Czumaj and Vöcking (2002), hence we shortly elaborate on that issue here.

Theorem 6. *For $m = 2$ identical machines, the price of anarchy of pure Nash equilibria for minimizing the schedule makespan in the congestion model is 4/3.*

Proof. To see that the POA is at leat 4/3, consider the following example. There are four jobs with $p_1 = p_2 = 1$ and $p_3 = p_4 = 2$. In an optimal solution, every machine processes one job of length 1 and one of length 2, yielding a makespan of 3. One pure Nash equilibrium is the strategy vector $(1, 1, 2, 2)$, i.e., the two short jobs go on the first machine, whereas the two long jobs go on the second machine. In that situation none of the jobs has an incentive to change the machine unilaterally. The makespan of this Nash equilibrium is 4, which proves that 4/3 is a lower bound on the price of anarchy of pure Nash equilibria.

To prove that the POA is at most 4/3, consider any schedule in (pure) Nash equilibrium. Denote by L_1 and L_2 the total loads of machines 1 and 2, respectively, and assume w.l.o.g. $L_2 \geq L_1$. Let $\delta = L_2 - L_1 \geq 0$. The makespan of the schedule in Nash equilibrium is hence $V_{NE} = L_2 = L_1 + \delta$. If there is only one job on machine 2, then no schedule can have a smaller makespan, and the schedule is optimal. Therefore, we assume that there are at least two jobs on machine 2. Any job on machine 2 must have a processing time at least δ, as any job with smaller processing time would have an incentive to change to machine 1. Therefore, $L_1 + \delta \geq 2\delta$ and thus $L_1 \geq \delta$. Since no schedule can do better than distributing the total processing time equally over both machines, $V_{OPT} \geq L_1 + \delta/2$. Thus we have

$$POA = \frac{V_{NE}}{V_{OPT}} \leq \frac{L_1 + \delta}{L_1 + \frac{\delta}{2}}.$$

This expression is maximized when L_1 is small. Using $L_1 \geq \delta$, we therefore get

$$POA \leq \frac{2\delta}{\frac{3}{2}\delta} = \frac{4}{3}.$$

□

In fact, the same proof technique works for an arbitrary number of machines m. One derives that the price of anarchy is at most $2 - 2/(m + 1)$ (Finn and Horowitz 1979). A matching lower bound was given by Schuurman and Vredeveld (2006).

Theorem 7 (Finn and Horowitz 1979, Schuurman and Vredeveld 2006)**.** *For an arbitrary number of identical machines, the price of anarchy of pure Nash equilibria for minimizing the makespan in the congestion model is $2 - 2/(m + 1)$.*

3.2 The Price of Anarchy in Sequencing Models

In the models of the previous section, the utility of any job assigned to a certain machine does only depend on the *total load* of that machine, but not on the *sequence* of the jobs on that machine. Next we discuss models where each job j's utility depends only on its own completion time C_j, and is independent of the processing that might occur later than C_j on the same machine.

Clearly, different local sequencing policies on the machines will yield different Nash equilibria, and the price of anarchy will depend on the employed local sequencing policies. The analysis of local sequencing policies in such settings was termed *coordination mechanisms* in the paper by Christodoulou, Koutsoupias, and Nanavati (2004). However, we prefer to not use this term in

this context, as we reserve the term "mechanism" for problems where agents have private (type) information; this is not the case here.

In the following, we examine the price of anarchy in different scheduling models and with different local sequencing policies. As in the preceding section, the central authority objective is to minimize the makespan $C_{\max}$ of the overall schedule. Our aim is to only highlight a few phenomena and proof techniques rather than to give a complete survey of the known results. For a more comprehensive overview, we refer to the paper by Immorlica, Li, Mirrokni, and Schulz (2005).

Consider again the setting where n job-agents have to choose one out of m machines to be processed on, thus the jobs' actions are again $(i_1, \ldots, i_n) \in M^n$. Each job seeks to minimize its own completion time C_j, thus

$$u_j(i_1, \ldots, i_n) = -C_j(i_1, \ldots, i_n)\,,$$

where $C_j(i_1, \ldots, i_n)$ is the completion time of job j in dependence on the jobs' actions and the sequencing of the jobs per machine.

We will use the term *local sequencing policy* to denote the sequencing policies implemented locally by the machines. As it turns out, for some local sequencing policies the schedules resulting from (pure strategy) Nash equilibria coincide with the outcome of well-known, classical scheduling algorithms. In such cases, for analyzing the price of anarchy we can just exploit well known results on the performance of those scheduling algorithms. To avoid confusion, note that we use the term 'policy' only for local sequencing policies, while we use the term 'algorithm' only for (centrally coordinated) scheduling algorithms.

We consider the most general of the three scheduling models, namely *unrelated* machine scheduling; thus if job j is scheduled on machine i, its processing time is p_{ij}. In the *Local SPT policy*, every machine processes the jobs that have selected that machine in order of non-decreasing processing times. As it turns out (Immorlica et al. 2005), the pure strategy Nash equilibria of the Local SPT policy coincide with the schedules that result from the *Ibarra-Kim* algorithm (Ibarra and Kim 1977). Notice that we assume in both cases that ties between jobs with equal processing times on one machine are broken in favor of the job with smaller index.

> **Ibarra-Kim algorithm.** In each of the $\tau = 1, \ldots, n$ iterations of the algorithm, select a pair (j, i) where j is an unscheduled job and i is a machine. If $C_j^\tau(i)$ denotes the completion time of job j when scheduled after all jobs already assigned to machine i in iterations 1, ...,$\tau - 1$, we select (j, i) as $\operatorname{argmin}_{i,j} C_j^\tau(i)$. We break ties by choosing a minimal j. In iteration τ, job j is then scheduled on machine i after all jobs already scheduled on i.

Theorem 8. *For unrelated (related, parallel) machines, the set of pure Nash equilibria for the Local SPT policy is precisely the set of solutions of the Ibarra-Kim algorithm.*

The proof of this result by Immorlica et al. (2005) is deferred to the full version of their paper; we therefore give a proof here.

Proof. Consider any job j, and consider the iteration when the Ibarra-Kim algorithm places job j on a machine minimizing j's completion time. At that iteration, for all machines i, and all jobs k already scheduled on machine i, it holds that $p_{ik} < p_{ij}$ (or such job k has the same processing time but a smaller index than j). Thus, in the final schedule, assuming that machines implement the Local SPT policy, j cannot be processed before any of those jobs k either. Given this constraint, however, j is already sitting on a machine that minimizes its completion time. Thus, j cannot

improve its completion time by unilaterally changing to another machine. That means that the Ibarra-Kim schedule is a Nash equilibrium for the Local SPT policy.

Conversely, consider any schedule that is a pure strategy Nash equilibrium for the Local SPT policy. In that schedule, for any job j denote by i_j the machine that hosts job j and let C_j^N be the completion time of job j (N for Nash equilibrium). Sort the jobs in order of non-decreasing completion times C_j^N. Note that jobs with equal completion times must be scheduled on different machines; let their respective order be chosen with respect to their index. We now schedule the jobs in this order on their respective machines i_j, and claim that this coincides with a run of the Ibarra-Kim algorithm. We need to show that whenever the τth job, say job j, is scheduled on its machine i_j, the combination (j, i_j) minimizes the completion time $C_k^\tau(i)$ among all combinations of unscheduled jobs k and machines i, where ties are broken by job index k. Suppose this is not the case and let (j, i_j) be the first job-machine pair for which the claim does not hold, j being the τth job in the given order. Then in iteration τ, there is a different job-machine pair (k, i) with $C_k^\tau(i) < C_j^\tau(i_j)$ or $C_k^\tau(i) = C_j^\tau(i_j)$ and $k < j$. Choose (k, i) such that $C_k^\tau(i)$ is minimum, and break ties according to smallest job index.

First, we argue that $C_k^\tau(i) < C_k^N$. Indeed, since

$$C_k^\tau(i) \leq C_j^\tau(i_j) = C_j^N \leq C_k^N \,, \tag{1}$$

we have $C_k^\tau(i) \leq C_k^N$. Assume that $C_k^\tau(i) = C_k^N$. Then we conclude from (1) that $C_k^\tau(i) = C_j^\tau(i_j)$, thus by the choice of k we must have $k < j$. But by (1) we also have that $C_j^N = C_k^N$. This, together with $k < j$, is a contradiction to the definition of our procedure, since we break ties according to smaller job index. Hence we must have $C_k^\tau(i) < C_k^N$.

Next we claim that all jobs l that are hosted by machine i in the Nash-equilibrium, and that would precede k according to the Local SPT-policy if k would chose machine i, are already present on machine i at iteration τ. To prove this claim, let l be a such a job. Since l would precede k, either $p_{il} < p_{ik}$, or $p_{il} = p_{ik}$, but $l < k$. In both cases, if l is not yet scheduled at iteration τ, its existence contradicts the choice of k.

This claim implies that in the Nash equilibrium, job k could improve its completion time C_k^N to $C_k^\tau(i) < C_k^N$ by choosing machine i, a contradiction. □

Utilizing this result, the price of anarchy of pure Nash equilibria for the Local SPT policy can be derived from known results on the performance of the Ibarra-Kim algorithm. Using such results by Finn and Horowitz (1979) and Schuurman and Vredeveld (2006) for parallel machines, by Aspnes, Azar, Fiat, Plotkin, and Waarts (1997) and Immorlica et al. (2005) for related machines, and by Ibarra and Kim (1977) for unrelated machines, we get the following.

Theorem 9. *The price of anarchy for minimizing the makespan in the sequencing model, when using the Local SPT policy on each machine, is*

- $2 - 2/(m+1)$ *on parallel machines,*
- $\Theta(\log m)$ *on related machines,*
- *and upper bounded by* m *on unrelated machines.*

For the sake of completeness, we mention that for the case of unrelated machines, Immorlica et al. (2005) show that the price of anarchy of the Local SPT policy is at least $\log m$. This result, however, does not follow from the analysis of the Ibarra-Kim algorithm; it is based on other techniques.

For the case of parallel machines, the Ibarra-Kim algorithm is in fact equivalent to the classical greedy SPT algorithm.

> **Greedy SPT algorithm.** Whenever a machine becomes idle, start a job with the shortest processing time among all remaining unscheduled jobs.

Theorem 9 states that this algorithm yields a schedule with makespan no more than $2 - 2/(m+1)$ times the optimal makespan. However, for the parallel machine case, the LPT algorithm yields an even better performance bound of $4/3 - 1/(3m)$ (Graham 1966).

> **Greedy LPT algorithm.** Whenever a machine becomes idle, start a job with the longest processing time among all remaining unscheduled jobs.

This motivates the analysis of the Local LPT policy on parallel machines. Again, it can be shown that pure strategy Nash equilibria correspond to the output of the greedy LPT algorithm (Christodoulou, Koutsoupias, and Nanavati 2004). The well known analysis of the LPT-algorithm by Graham (1966) now yields the following.

Theorem 10 (Christodoulou, Koutsoupias, and Nanavati 2004)**.** *For the parallel machine setting, the price of anarchy for minimizing the makespan in the sequencing model, when using the Local LPT policy on each machine, is* $4/3 - 1/(3m)$.

Clearly, from the above result it follows that the price of anarchy is at least $4/3 - 1/(3m)$ also for related (or unrelated machines). We mention the following result without a proof.

Theorem 11 (Graham 1966, Immorlica et al. 2005)**.** *For the related machine setting, the price of anarchy for minimizing the makespan in the sequencing model, when using the Local LPT policy on each machine, is bounded as follows:* $4/3 - 1/(3m) \leq POA \leq 2 - 2/m$.

Consider now the case of unrelated machines. In contrast to the price of anarchy of the Local SPT policy, which is at most m by Theorem 9, it is *unbounded* for the Local LPT policy. To that end, consider the following example.

Example 12. *Consider two machines* 1 *and* 2 *and two jobs* 1 *and* 2*. Let* $p_{11} = p_{22} = 1$ *and* $p_{12} = p_{21} = K$ *for some constant* $K > 0$*. Then in one Nash equilibrium, job 1 is processed by machine 1 and job 2 by machine 2. The makespan of the resulting schedule is 1. In the other Nash equilibrium, job 1 is processed on machine 2 and job 2 on machine 1. Because longer jobs are processed before shorter ones on every machine, unilaterally changing the machine is not beneficial for either job. The makespan is* K *in this case. Therefore, the price of anarchy is equal to* K *and is hence unbounded.*

The question of *existence* of pure strategy Nash equilibria in the above mentioned settings is often answered by showing that a certain local sequencing policy constitutes a so-called *potential game*. This method is used by Immorlica et al. (2005) for analyzing the Local SPT policy for the case of related machines. Potential games have a *potential function* mapping strategy vectors to real numbers such that the potential function decreases whenever an agent unilaterally changes its strategy in such a way that its own utility increases. Minima of the potential function then correspond to pure strategy Nash equilibria of the game. Potential functions where first used by Rosenthal (1973) and formally introduced by Monderer and Shapley (1996). We refer to those references for further reading.

Notice that the link to potential games also establishes a close relationship to *local search algorithms* in optimization. One can define a *local search neighborhood* of a given strategy vector

by considering all strategy vectors where only one agent has changed its strategy to the best response against the given strategies of the other agents. The potential function takes the role of the objective function of the local search. Local optima of those neighborhoods then correspond to pure strategy Nash equilibria. The analysis of the quality of local optima is therefore closely related to the analysis of (pure strategy) Nash equilibria. The quality of local optimal of several neighborhoods in machine scheduling was analyzed, for example, by Schuurman and Vredeveld (2006).

3.3 The Price of Anarchy for Other Objective Functions

To our knowledge, the price of anarchy in scheduling has almost exclusively been studied with respect to the makespan objective. However, the strategic behavior of a job-agent that seeks to minimize its own completion time is not affected by the objective function of the central authority. Therefore, the Nash equilibria for the different models discussed in the previous section remain Nash equilibria if only the central authorities' objective function is modified.

As an example, we consider a model that differs only slightly from the models in the previous section. Each job-agent now has a weight w_j additionally to its processing time p_j, and as before, seeks to be finished as early as possible. The strategy of each job remains the choice of a machine. As central objective we consider the minimization of the total weighted completion time $\sum_{j\in J} w_j\, C_j$.

The most natural (because optimal) local sequencing policy on the machines is then the well known Local WSPT policy, also known as Smith's rule: each machine processes its jobs in the order of non-increasing ratios w_j/p_j. For each machine individually, this yields the minimum total weighted completion time $\sum_{j\in J} w_j\, C_j$ (Smith 1956). Consider now the following algorithm.

> **(Ibarra-Kim version of) WSPT algorithm.** Sort the jobs in order of their weight over processing time ratios w_j/p_j, largest first. In that order, schedule each job on the machine that minimizes its completion time when scheduled after all jobs already scheduled on that machine.

Notice that for parallel machines, this algorithm reduces to the classical WSPT algorithm that just scheduled the jobs greedily in order of non-increasing ratios w_j/p_j. The worst case behavior of this algorithm has been analyzed by Kawaguchi and Kyan (1986). Similar to the proof of Theorem 8, one can show that the set of Nash equilibria of the Local WSPT policy is equal to the set of all possible outputs of this WSPT algorithm.

Theorem 13. *For related (parallel) machines, the set of pure Nash equilibria for the Local WSPT policy is precisely the set of solutions of the above WSPT algorithm.*

Consequently, the price of anarchy of the Local WSPT policy follows from the analysis of that algorithm. For parallel machines, the work of Kawaguchi and Kyan (1986) thus yields the following.

Theorem 14. *For parallel machines, the price of anarchy of the Local WSPT policy for minimizing* $\sum w_j C_j$ *in the sequencing model is* $(\sqrt{2}+1)/2 \approx 1.207$.

For the case of related machines, we are not aware of non-trivial bounds on the price of anarchy. For the unrelated machine case, it is not even clear whether or not a pure strategy Nash equilibrium exists. However, if all jobs have the same weights, then the Local WSPT policy is equivalent to the Local SPT policy. The existence of pure strategy Nash equilibria then follows from Theorem 8. Also for this case, however, we are not aware of any non-trivial bound on the price of anarchy.

4 Models with Private Information

In the previous section we addressed the question what happens if in a scheduling application a part of the decisions is left to selfish job agents. Given various policies that determine how jobs are scheduled on the selected machines, we compared the objective in equilibrium with the objective of the optimal solution. In this section we additionally assume that the agents own some private information, namely their types, and these are not publicly known. For any given agent, its type will influence its action in the game. Since agents do not know other agents' types, they do not know which actions are beneficial for the other agents and therefore which actions other agents are likely to chose. This additional uncertainty results in more complicated equilibrium concepts than in the previous section. Our general goal is to design allocation algorithms and associated payments that discourage agents from gaming by pretending false types, because wrong information about types makes it often impossible to select an allocation that optimizes the objective of the original optimization problem. We start with some general notation, then discuss general techniques and key results, and finally review a couple of interesting models related to scheduling.

4.1 Mechanism Design

An allocation algorithm α together with a payment scheme π is usually referred to as a *mechanism*, denoted by $\mu = (\alpha, \pi)$. We will present several examples for mechanisms in the following sections. Next, we introduce the equilibrium that is most robust towards the information uncertainty described above and that is at the same time the one that is best studied in the algorithmically oriented literature in mechanism design.

Recall that for agent k, a strategy x_k is a mapping from types t_k to actions a_k. We denote by t_{-k}, x_{-k} and a_{-k} the vectors of types, strategies and actions respectively of all agents other than k. For the type, strategy and action vector of all the agents, we then write (t_k, t_{-k}), (x_k, x_{-k}), and (a_k, a_{-k}).

Definition 15 (dominant strategy equilibrium)**.** *Let $\mu = (\alpha, \pi)$ be a mechanism. A strategy vector $x \in X$ is called a dominant strategy equilibrium, if for all agents k, for all types t_k of agent k, for all actions a_{-k} of the other agents and all alternative actions a_k of agent k it holds that*

$$v_k(\alpha(x_k(t_k), a_{-k})|t_k) - \pi_k(x_k(t_k), a_{-k}) \geq v_k(\alpha(a_k, a_{-k})|t_k) - \pi_k(a_k, a_{-k}).$$

Remarkably, this means that independent of which actions the other agents take, it never pays off for any agent k, to deviate from its strategy x_k.

A class of mechanisms that receive great attention in the literature (Briest, Krysta, and Vöcking 2005; Gui, Müller, and Vohra 2004; Lavi, Mu'alem, and Nisan 2003; Saks and Yu 2005) are *direct revelation* mechanisms . In a direct revelation mechanism, the only action that an agent is required to take is reporting its type, thus $x_k \colon T_k \to T_k$. This class of mechanisms is of particular interest for the following reason. Assume we have a scheduling problem where part of the instance is private information of the agents. Given reports about that private information, we can define the allocation algorithm that merely chooses an optimal solution. (For the time being, we are not addressing the question *how* this optimal solution is derived.) Let us denote it by the *exact allocation algorithm*. Without payments, utility maximizing agents might misreport their private information in such settings in order to achieve a more favorable outcome. With well-designed payments, however, agents may get incentives to report their private information truthfully in the following sense.

Definition 16 (dominant strategy incentive compatible, truthful)**.** *A direct revelation mechanism is called* dominant strategy incentive compatible*, or* truthful*, if the strategy vector x in which each agent truthfully reports its type, that is, $x_k = id$ for all k, is a dominant strategy equilibrium. An allocation algorithm α is said to be* truthfully implementable*, if we can find a payment rule π such that the mechanism $\mu = (\alpha, \pi)$ is truthful.*

Given some optimization problem, if the exact allocation algorithm is truthfully implementable, there is of course still the issue whether the algorithm runs in polynomial time, and whether the payments can be computed in polynomial time. If we leave this algorithmic problem out of consideration, however, it is often surprisingly easy to provide a truthful implementation, because many optimization problems are special cases of the setting in which we can apply so-called Vickrey-Clarke-Groves (VCG) payments. In what follows we use the notation by Roberts (1979).

Definition 17. *Given ℓ agents, their types $t_1, \ldots, t_\ell$, valuation functions $v_1, \ldots, v_\ell$, strictly positive weights $\gamma_1, \ldots, \gamma_\ell$, and constants β_y for every $y \in Y$, an allocation algorithm α is called an* affine maximizer*, if it chooses a schedule $y \in Y$ that maximizes $\beta_y + \sum_{k=1}^{\ell} \gamma_k v_k(y|t_k)$.*

The following theorem is in this generality due to Roberts (1979). For all weights equal to 1, it has been proven by Clarke (1971) and Groves (1973), while for the special case of single-item auctions it has been proven by Vickrey (1961).

Theorem 18. *Let an allocation algorithm α be an affine maximizer, and let for every agent k, h_k be an arbitrary function mapping type reports t_{-k} of the other agents to real numbers. Then the mechanism $\mu = (\alpha, \pi)$ is truthful if the payments are defined as follows.*

$$\pi_k(t) = h_k(t_{-k}) - \frac{1}{\gamma_k}\beta_{\alpha(t)} - \sum_{k' \neq k} \frac{\gamma_{k'}}{\gamma_k}\, v_{k'}(\alpha(t)|t_{k'})\,.$$

It is intuitive to give the very short proof of this important theorem.

Proof. Let us assume that agent k reports $\hat{t}_k$ instead of its true type t_k, and let $\hat{t} = (\hat{t}_k, t_{-k})$. Since α is an affine maximizer we have:

$$\beta_{\alpha(t)} + \sum_k \gamma_k v_k(\alpha(t)|t_k) \geq \beta_{\alpha(\hat{t})} + \sum_k \gamma_k v_k(\alpha(\hat{t})|t_k),$$

which implies:

$$v_k(\alpha(t)|t_k) + \frac{1}{\gamma_k}\beta_{\alpha(t)} + \sum_{k' \neq k} \frac{\gamma_{k'}}{\gamma_k}\, v_{k'}(\alpha(t)|t_{k'}) \geq v_k(\alpha(\hat{t})|t_k) + \frac{1}{\gamma_k}\beta_{\alpha(\hat{t})} + \sum_{k' \neq k} \frac{\gamma_{k'}}{\gamma_k}\, v_{k'}(\alpha(\hat{t})|t_{k'}).$$

If we subtract $h_k(t_{-k})$ on both sides of this inequality, we get on the left hand side the utility of agent k for truth-telling, and on the right hand side its utility when reporting $\hat{t}_k$. □

Note that the generality of the functions h_k gives some flexibility to define payments. In auctions, for example, one uses this flexibility to adjust prices such that agents who do not win any object pay 0. Note further that, for fixed type report t_{-k}, agent k pays a price that depends only on the allocation that is selected by the affine maximizer, and not on its particular type report by which this allocation is achieved. It is easy to see that this so-called *taxation principle* does not only hold for affine maximizers and their VCG payments, but it must hold for any truthful mechanism.

Let us provide an example application of VCG payments in the context of scheduling.

Example 19. *Suppose there is a single machine, and there are job-agents whose characteristics —weights w_j and processing times p_j— are private information. The valuation of an agent is the negative of its weighted completion time (in order to make the agents utility maximizers). Let α be the exact allocation algorithm, i.e., α chooses a schedule minimizing the weighted sum of completion times. Note that this is an affine maximizer, since it maximizes the sum of agent valuations. Let us use the notation $C_k(t)$ for the completion time of agent k in the optimal solution of the instance with all agents, and $C_k(t_{-j})$ as the completion of agent k in the optimal solution of the instance in which j is not present. Now choose $h_j(t_{-j})$ as the negative of the optimal weighted sum of completion times if agent j is not present. We get the following VCG payments.*

$$\begin{aligned}\pi_j(t) &= -\sum_{k\neq j} w_k C_k(t_{-j}) + \sum_{k\neq j} w_k C_k(t) \\ &= \sum_{k\neq j} w_k (C_k(t) - C_k(t_{-j})) = \sum_{k \text{ delayed by } j} w_k p_j.\end{aligned}$$

Here, the last sum is restricted to those jobs that are delayed due to the presence of j. In other words, job j pays for the decrease in utility of other agents. By Theorem 18, with these payments agents maximize their utility by reporting their types truthfully.

Notice that the scheduling problem of Example 19 does not only allow for a truthful implementation, but at the same time is the allocation algorithm a polynomial time algorithm: The exact allocation algorithm α just schedules the jobs in the order of non-increasing ratios w_j/p_j (Smith 1956). Given the reports of all agents, also the payments can be computed efficiently.

It is often the case, however, that an exact allocation algorithm is not that easily obtainable, for example because the underlying optimization problem is NP-hard. If instead of an exponential time exact allocation algorithm, we use an allocation algorithm that is a (suboptimal) heuristic, this algorithm is generally not an affine maximizer, and computing payments with the VCG formula on the basis of the solutions computed by the heuristic does not necessarily yield a truthful mechanism (Nisan and Ronen 2000; Ronen 2006).

On the other hand, even if an allocation algorithm is *not* an affine maximizer, it might be truthfully implementable. In order to verify whether a given allocation algorithm is truthfully implementable, and in order to determine the required payments, a characterization of truthfully implementable allocation algorithms is of great importance. Such a characterization has recently been given by Saks and Yu (2005), generalizing earlier results by Bikhchandani, Chatterjee, and Sen (2004), Gui, Müller, and Vohra (2004), Lavi, Mu'alem, and Nisan (2003), and Roberts (1979). We start with a definition.

Definition 20 (Weak monotonicity)**.** *An allocation algorithm α is said to satisfy* weak monotonicity *if for all agents k, for all types t_k, $\hat{t}_k$ of agent k, and for all types t_{-k} of other agents:*

$$v(\alpha(t_k, t_{-k})|t_k) - v(\alpha(t_k, t_{-k})|\hat{t}_k) \geq v(\alpha(\hat{t}_k, t_{-k})|t_k) - v(\alpha(\hat{t}_k, t_{-k})|\hat{t}_k).$$

Weak monotonicity is a necessary condition for a truthful implementation of an allocation algorithm. Indeed, using the types as given in the definition, and assuming that truthful payments exist, we get the following two inequalities, from which weak monotonicity follows:

$$\begin{aligned} v(\alpha(t_k, t_{-k})|t_k) - \pi(t_k, t_{-k}) &\geq v(\alpha(\hat{t}_k, t_{-k})|t_k) - \pi(\hat{t}_k, t_{-k}) \\ v(\alpha(\hat{t}_k, t_{-k})|\hat{t}_k) - \pi(\hat{t}_k, t_{-k}) &\geq v(\alpha(t_k, t_{-k})|\hat{t}_k) - \pi(t_k, t_{-k}). \end{aligned}$$

For some settings, weak monotonicity of an allocation rule is even a sufficient condition for truthful implementability, see Theorem 24 below. The most general result of this type has been obtained by Saks and Yu. They have shown that weak monotonicity is sufficient for convex domains:

Theorem 21 (Saks and Yu 2005)**.** *Let the set of outcomes Y be finite, and let for all agents k the type be represented as a valuation vector with a valuation for every possible outcome $y \in Y$. That is, $T_k \subseteq \mathbb{R}^Y$ and $v_k(y|t_k) = t_{ky}$, $y \in Y$. Furthermore, assume that all T_k are convex. Then an allocation algorithm $\alpha : A \to Y$ is truthfully implementable if and only if α satisfies weak monotonicity.*

The proof of Theorem 21 is based on a link that can be made between the question whether an allocation algorithm is truthfully implementable, and the question whether particular networks have no cycles of negative length. If the latter is the case, we can compute shortest paths in such networks, and the network is defined such that shortest path lengths give us payments that make the allocation algorithm truthful. This construction of truthful payments has first been given by Rochet (1987) in a slightly different context, and also Gui et al. (2004) showed how the link to certain networks can be used to characterize truthfully implementable allocation algorithms.

The network related to an allocation algorithm α contains a node for each outcome, in our case for each schedule $y \in Y$. There is a directed edge (arc) from every y to every other $\hat{y}$. Now fix an agent k and a type report t_{-k} of the other agents, and consider the sub-network induced by those outcomes that can be achieved by a report of agent k, fixing the report t_{-k} of the other agents. That is, the subnetwork contains exactly those y such that $y = \alpha(t_k, t_{-k})$ for some $t_k \in T_k$. On this subnetwork we define arc lengths by:

$$\ell(y, \hat{y}) = \inf_{t_k : \alpha(t_k, t_{-k}) = \hat{y}} [v_k(\hat{y}|t_k) - v_k(y|t_k)].$$

By replacing $\hat{y}$ by $\alpha(\hat{t}_k, t_{-k})$ in the term $v_k(\hat{y}|t) - v_k(y|t)$, and y by $\alpha(t_k, t_{-k})$ for some t_k such that $\alpha(t_k, t_{-k}) = y$, one can easily verify that the network has no negative 2-cycles if and only if weak-monotonicity is satisfied. Furthermore, by construction, all node potentials π_y satisfying

$$\pi_{\hat{y}} \leq \pi_y + \ell(y, \hat{y})$$

correspond to payment schemes that make the allocation algorithm truthful. (Recall that due to the taxation principle, given t_{-k}, payments may only depend on the outcome y, and not on the type t_k by which this outcome is achieved.) Indeed:

$$\begin{array}{lrcll} & \pi_{\hat{y}} & \leq & \pi_y + \ell(y, \hat{y}) & \text{for all } y, \hat{y} \\ \Leftrightarrow & \pi_{\hat{y}} & \leq & \pi_y + \inf_{t_k : \alpha(t_k, t_{-k}) = \hat{y}} [v_k(\hat{y}|t_k) - v_k(y|t_k)] & \text{for all } y, \hat{y} \\ \Leftrightarrow & v_k(\hat{y}|t_k) - \pi_{\hat{y}} & \geq & v_k(y|t_k) - \pi_y & \text{for all } y, \hat{y}, t_k : \alpha(t_k, t_{-k}) = \hat{y}. \end{array}$$

The left hand side of the last inequality is the utility from truth-telling, the right hand side is the utility from reporting a type that leads to any other schedule y.

It is a basic result in combinatorial optimization that such node potentials exist if and only if the network does not have any negative length cycle. Furthermore, in such a case, the potentials can be computed by giving an arbitrary node y a potential $\pi_y = 0$, and then computing shortest paths from this node to any other node.

It turns out that in many settings the constructed networks have no negative cycle if and only if they have no negative 2-cycles. Theorem 21 provides an example of such a setting, other examples were given by Bikhchandani et al. (2004) and Gui et al. (2004). In Section 4.2.2 we show how the network can be used to provide an alternative proof for a result given originally by Archer and Tardos (2001).

4.2 Performance of Truthful Mechanisms in Machine Scheduling

In this section we regard specific scheduling models from a mechanism design perspective and show how the techniques described in the previous section can be used to analyze them. We investigate the trade-off between mechanism design goals (truthfulness) and optimization objectives, such as exact optimization, approximation and competitiveness (in the case of online algorithms). The first two models studied in the following refer to off-line situations where the machines are the agents. The two models regarded thereafter are online scheduling models with job-agents.

By the *performance guarantee* of an (off-line) allocation algorithm for a minimization problem, we refer to an upper bound on the ratio between the worst possible objective value that can be achieved by the allocation algorithm and the optimal objective value. For a mechanism, the performance guarantee is defined with respect to the worst possible objective value that can occur when all agents report their types truthfully. Note that we do not demand any performance guarantee for non-truthful agents.

In an online optimization problem, the instance is not known entirely beforehand, but part of it is only revealed over time. Therefore, any online algorithm has to make decisions on the basis of incomplete information. In our strategic mechanism design setting, the incompleteness of information is due to two sources—the online setting and the fact that agents have private information. The goal is to design online mechanisms that have good properties with respect to truthfulness and performance. In the two online models we describe, the objective is equivalent to affine maximization. However, both problems do not allow for exact allocation algorithms due to the online situation. We say that an online algorithm with minimization objective has performance guarantee ϱ for $\varrho \geq 1$ if the schedule resulting from the online algorithm has an objective value no more than ϱ times the optimal off-line objective value. That is, we compare the online algorithm to the best solution that could have been achieved if the entire instance had been known in advance. For an online mechanism, we demand the performance guarantee only with respect to truthful agents.

4.2.1 Unrelated Machine Scheduling with Machine Agents

In this section, we illustrate the conflict between optimizing the objective function of a given problem and obtaining a truthful mechanism. In the setting that we discuss, it turns out to be impossible to design a mechanism that is at the same time truthful and optimizing the objective function. This section is based on the work of Nisan and Ronen (2001).

Consider the following strategic version of unrelated machine scheduling. The agents are the m machines, on which n jobs have to be scheduled. The type of each machine i is the vector $t_i = (t_{i1}, \ldots, t_{in})$, where t_{ij} denotes the time that machine i needs to process job j. Hence, the type spaces are n-dimensional. The valuation of a machine for a certain schedule is the negative of the total time it needs to process all the jobs assigned to it. The objective of the optimization problem is to minimize the makespan. Obviously, the allocation algorithm that chooses the optimal schedule for every instance does not belong to the class of affine maximizers.

In fact, Nisan and Ronen (2001) show that no truthful mechanism for the regarded problem can approximate the optimal solution with an approximation factor better than 2. We simplify their example and use the theory introduced in Section 4.1 to show the weaker result that no truthful mechanism can exactly optimize the objective function.

Theorem 22. *There does not exist a truthful mechanism that minimizes the makespan in the strategic version of the unrelated machine scheduling problem.*

Proof. As we have seen in the previous section, an allocation algorithm has to satisfy weak monotonicity as a necessary condition in order to be truthfully implementable. Consider an instance with two machines and four jobs. Let the second machine's type be fixed as $(1,1,1,1)$. Consider first the type $t = (1,1,1,1)$ for machine 1. The allocation algorithm that minimizes the makespan has to allocate two jobs to each machine. Let w.l.o.g. jobs 1 and 2 be allocated to machine 1, i.e., for type $t = (1,1,1,1)$ the set $T = \{1,2\}$ of jobs is assigned to machine 1. Consider now type $t' = (\varepsilon, \varepsilon, 1+\varepsilon, 1+\varepsilon)$ of machine 1 for $\varepsilon > 0$, but close to zero. Now, an optimal allocation algorithm has to assign job 1,2 and either 3 or 4 to machine 1, and the remaining job to machine 2. W.l.o.g. let $T' = \{1,2,3\}$ be the set of jobs assigned to machine 1. Then weak monotonicity reads as follows:

$$\begin{aligned} & v_1(T'|t') - v_1(T'|t) + v_1(T|t) - v_1(T|t') \geq 0 \\ \Leftrightarrow \quad & -\sum_{j\in T'} t'_j + \sum_{j\in T'} t_j - \sum_{j\in T} t_j + \sum_{j\in T} t'_j \geq 0 \\ \Leftrightarrow \quad & \sum_{j\in T\setminus T'} (t'_j - t_j) + \sum_{j\in T'\setminus T} (t_j - t'_j) \geq 0. \end{aligned}$$

In our example, the left hand side of the last inequality is equal to $t_3 - t'_3 = 1 - (1+\varepsilon) = -\varepsilon < 0$. Therefore, weak monotonicity is not satisfied and the optimal allocation algorithm cannot be extended to a truthful mechanism. The example can easily be modified to a larger number of jobs and machines. □

In view of this negative result, the question arises which performance guarantee a truthful mechanism can achieve. Nisan and Ronen (2001) suggest the following MinWork mechanism, which can be viewed as auctioning each task separately in a Vickrey auction.

> **MinWork mechanism.**
> *Allocation algorithm:* After each machine has declared its type, assign each job to the machine that has declared the lowest processing time for that job. Ties are broken arbitrarily. For a vector $t = (t_1, \ldots, t_m) \in T^m$ of machine declarations, the set of jobs allocated to machine i is denoted by $\alpha_i(t)$.
> *Payment scheme:* For a vector $t = (t_1, \ldots, t_m) \in T^m$ of machine declarations, the payment for machine i is defined as $\pi_i(t) = -\sum_{j\in\alpha_i(t)} \min_{i'\neq i} t_{i'j}$. That is, each machine receives for each job that it processes a payment that equals the second lowest declaration of any machine for that job.

Theorem 23 (Nisan and Ronen 2001)**.** *MinWork is truthful and an m-approximation.*

Proof. The MinWork mechanism minimizes the total work done and therefore maximizes the sum of the valuations of all machine-agents. Therefore, the allocation algorithm of the mechanism is an affine maximizer. Set $h_i(t_{-i}) := -\sum_{j=1}^{n} \min_{i'\neq i} t_{i'j}$ to see that the payment scheme of MinWork is a VCG payment scheme. Therefore, MinWork is truthful according to Theorem 18.

For the performance guarantee with respect to the makespan objective, note that the optimum makespan V_{OPT} is lower bounded by

$$V_{OPT} \geq \frac{1}{m} \sum_{j=1}^{n} \min_{i=1,\ldots,m} t_{ij}.$$

The makespan V_{MW} resulting from the allocation algorithm of the MinWork mechanism is upper bounded by

$$V_{MW} \leq \sum_{j=1}^{n} \min_{i=1,\ldots,m} t_{ij},$$

i.e., $V_{MW} \leq mV_{OPT}$, assuming that all agents report their true types. □

In fact, Nisan and Ronen prove that the mechanism is strongly truthful, i.e., truthtelling is the only dominant strategy for every agent.

As mentioned before, no truthful mechanism for the regarded problem can approximate the optimal solution better than a factor of 2. Therefore, MinWork is best possible for two machines. Moreover, the authors conjecture that also for the general case with m machines, the upper bound of m is tight, yet this remains an open question.

4.2.2 Related Machine Scheduling with Machine Agents

Archer and Tardos (2001) consider a similar model for related machine scheduling. Again, the agents are the machines. In contrast to the model from the previous section, the processing times of different jobs on one machine are not independent and the type spaces of the machine-agents are one-dimensional. More precisely, each machine i has a speed s_i and the type is defined to be the inverse of this speed $t_i := 1/s_i$. Each job j has a (unit-speed) processing time p_j. The time that is needed to process job j on machine i is $t_i p_j = p_j/s_i$. The action of each machine is to declare its type. If machine i is assigned the set of jobs $\alpha_i(t) \subseteq J$ for a vector of declarations of all machines $t = (t_1, \ldots, t_m) \in T^m$, then its valuation is $v_i(t|t_i) = -\sum_{j \in \alpha_i(t)} t_i p_j$. The objective is again to minimize the makespan.

For this setting, Archer and Tardos derive a necessary and sufficient condition for an allocation algorithm to be truthfully implementable. For an allocation algorithm, denote by $L_i(t) = L_i(t_i, t_{-i})$ the total workload assigned to machine i. Then an agent's valuation can be written as $v_i(t|t_i) = -t_i \cdot L_i(t_i, t_{-i})$.

Theorem 24 (Decreasing Work Curves, Archer and Tardos 2001)**.** *An allocation algorithm is truthfully implementable if and only if for all agents i and all $t_{-i} \in T^{m-1}$ the function $L_i(t_i, t_{-i})$ is a decreasing function of t_i. If this is the case, then the following payments yield a truthful mechanism*

$$\pi_i(t_i, t_{-i}) = -\left(h_i(t_{-i}) + t_i L_i(t_i, t_{-i}) - \int_0^{t_i} L_i(u, t_{-i}) du \right).$$

Here, h_i are arbitrary functions that depend on the declarations of all agents except i.

Instead of giving the original proof by Archer and Tardos (2001), we show how Theorem 24 is implied by the results of Saks and Yu (2005) and Gui et al. (2004). We note, however, that the original proof by Archer and Tardos (2001) even shows that the above payments are the only payments that yield a truthful mechanism.

Proof. Convexity of the type spaces in the sense of Theorem 21 can be easily verified. It is due to the fact that the speed and therefore its inverse can be an arbitrary positive real number and the valuation of an agent for a certain schedule depends linearly on the inverse of the speed.

From Theorem 21, we know that weak monotonicity is a necessary and sufficient condition for the allocation algorithm to be truthfully implementable. To verify weak monotonicity, let t_i and $\hat{t}_i$

be different types of an agent i with $t_i < \hat{t}_i$, and let the reports of the other agents be fixed as t_{-i}. Then weak monotonicity is equivalent to

$$\begin{aligned} & v_i((t_i, t_{-i})|t_i) - v_i((t_i, t_{-i})|\hat{t}_i) \geq v_i((\hat{t}_i, t_{-i})|t_i) - v_i((\hat{t}_i, t_{-i})|\hat{t}_i) \\ \Leftrightarrow \quad & -t_i L_i(t_i, t_{-i}) + \hat{t}_i L_i(t_i, t_{-i}) \geq -t_i L_i(\hat{t}_i, t_{-i}) + \hat{t}_i L_i(\hat{t}_i, t_{-i}) \\ \Leftrightarrow \quad & (\hat{t}_i - t_i)(L_i(t_i, t_{-i}) - L_i(\hat{t}_i, t_{-i})) \geq 0. \end{aligned}$$

This condition is satisfied whenever $L_i(t_i, t_{-i}) - L_i(\hat{t}_i, t_{-i}) \geq 0$, i.e., if and only if the function L_i is decreasing in the report of agent i.

For the second part of the theorem, we will use the results of Gui et al. (2004) to derive the payment scheme given above. Let α be an allocation algorithm that satisfies the decreasing work curves condition, let agent i and the report of the other agents t_{-i} be fixed and let for simplicity of notation $L(t_i) := L_i(t_i, t_{-i})$ denote the workload assigned by α to i when reporting t_i. First, we observe that there are only finitely many possible schedules that can yield only finitely many different values of $L(t_i)$. We denote those values by $L_{max} = \sum_{j \in J} p_j > \cdots > L_1 > L_0 = 0$. Using that α satisfies the decreasing work curve condition, we get the picture in Figure 1 for the graph of $L(t_i)$. In order to determine the payments according to the method of Gui et al. (2004), we have

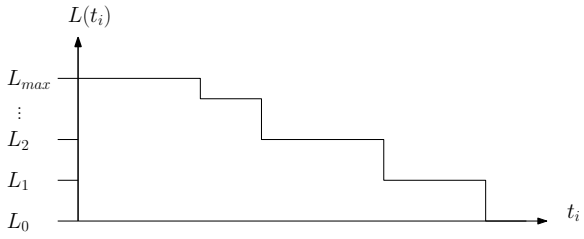

Figure 1: decreasing work curve

to determine shortest paths in the network described in Section 4.1. Let π_r denote the payment that machine i has to make if it is assigned workload L_r by the allocation algorithm. We define $\pi_0 = 0$ and determine the shortest path from the node L_0 to node L_r. It can easily be shown that if $L_{r_3} > L_{r_2} > L_{r_1}$ then the arc lengths satisfy $\ell(L_{r_1}, L_{r_3}) \geq \ell(L_{r_1}, L_{r_2}) + \ell(L_{r_2}, L_{r_3})$. Therefore, $[L_0, L_1, \ldots, L_r]$ is a shortest path from L_0 to L_r. Hence, the payments can be written as:

$$\pi_r = \sum_{i=1}^{r} \ell(L_{i-1}, L_i) = \sum_{i=1}^{r} \inf_{t_i : L(t_i) = L_i} (-t_i L_i + t_i L_{i-1}) = -\sum_{i=1}^{r} \left(\sup_{t_i : L(t_i) = L_i} t_i \right) (L_i - L_{i-1}).$$

Thus, if machine i is assigned a total workload of L_r, it receives a payment that is equal to the area under the graph of $\min(L(t_i), L_r)$, as depicted in Figure 2. Thus the payment to machine i is $t_i L(t_i) + \int_{t_i}^{\infty} L(u)du$. If we now let $h_i(t_{-i}) = \int_0^{\infty} L(u)du$, then the payment that machine i receives equals $t_i L(t_i) + \int_{t_i}^{\infty} L(u)du = h_i(t_{-i}) + t_i L(t_i) - \int_0^{t_i} L(u)du$, which proves the claim. □

Archer and Tardos (2001) give a randomized polynomial time allocation algorithm, which is based on bin-packing and rounding fractional assignments of jobs to bins in a random fashion. The allocation algorithm fulfills the decreasing work curves condition with respect to the expected utilities of the agents. The following result is obtained.

Theorem 25 (Archer and Tardos 2001)**.** *For the strategic version of related machine scheduling with machine agents whose private information is their speed, there exists a mechanism with the following properties:*

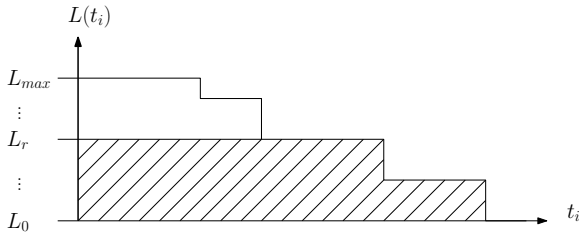

Figure 2: payment for workload L_r

- *the mechanism is truthful when agents maximize expected utilities,*
- *it yields a 3-approximation for the makespan of the schedule independent of the random choices,*
- *the payments can be computed in polynomial time.*

There also exists a deterministic allocation algorithm that fulfills the monotonicity condition of Theorem 24 due to Kovacs (2005). The allocation algorithm given by Kovacs (2005) runs in polynomial time and yields a 3-approximation as well. Thus, the existence of a truthful payment scheme is guaranteed by Theorem 24. However, it is not clear whether the associated payments can be computed in polynomial time. Other papers that have improved some of the results of Archer and Tardos (2001) are by Auletta, De Prisco, Penna, and Persiano (2004), Ambrosio and Auletta (2005), and Andelman, Azar, and Sorani (2005).

4.2.3 Preemptive Online Scheduling on a Single Machine with Job Agents

We now turn to online scheduling models with job agents. The following single machine model was analyzed by Porter (2004). There is one machine that has to process n jobs, where n is not known beforehand. Preemption of jobs is allowed. Each job j has a release date r_j, a processing time p_j, a deadline d_j and a weight w_j. Those four values are private information. The type of agent j is thus $t_j = (r_j, p_j, d_j, w_j)$. The aim is to design a direct revelation mechanism, that is, jobs have to report their types to the mechanism. We assume that a job can declare a release date $\hat{r}_j \geq r_j$ and a processing time $\hat{p}_j \geq p_j$, while it can declare an arbitrary deadline $\hat{d}_j$ and an arbitrary weight $\hat{w}_j$. The reason that we do not admit declaring a shorter processing time is that this could be easily detected and punished by the mechanism. If job j is completed before its deadline, then its valuation is w_j, otherwise its valuation is zero. Jobs have to pay for being processed. The payments have to be determined online as well, i.e., the payment for a job must be determined at the latest when the job leaves the system. The central objective is to maximize the sum of the weights of all jobs that are completed by their deadline, i.e., the goal is affine maximization. However, an exact allocation algorithm does not exist due to the online nature of the problem. Lower bounds on the performance guarantee of any online algorithm for the problem are given by Baruah, Koren, Mao, Mishra, Raghunathan, Rosier, Shasha, and Wang (1992). Those bounds imply in particular that there is no exact allocation algorithm. Porter (2004) shows the following.

Theorem 26 (Porter 2004)**.** *For the described single machine model, there exists a truthful mechanism with performance guarantee* $((1+\sqrt{k})^2+1)$*, where* k *is an upper bound on* $\max_{j,\ell}(w_\ell p_j)/(p_\ell w_j)$ *which is known to the mechanism.*

Notice that the mechanism is assumed to know an upper bound k on the maximum ratio $\max_{j,\ell}(w_\ell p_j)/(p_\ell w_j)$. In addition, the mechanism needs to know the value $\delta_{min} := \min_j w_j/p_j$. The single machine processes at any point in time a job that is chosen among the available jobs that still have a chance to be completed before their declared deadline. The choice depends on the declared weights of the jobs, the time already spent processing each job, on k and on δ_{min}. Jobs are only returned at their declared deadlines. The payment that a job has to make is zero if it is not completed before its declared deadline and equal to the minimum weight that the job could have declared such that it still would have been finished in time, given the declarations of the other jobs and given its own declarations on release date, deadline and processing time.

The proof of the truthfulness of the mechanism given by Porter (2004) is quite technical. Intuitively, the payments can be seen as VCG-payments and are chosen such that the mechanism is truthful with respect to the weights. By returning the job not until the declared deadline it is achieved that a job has no incentive to declare a larger deadline than the true one. Therefore, no job has an incentive to declare a "better" type. It can be shown that also declaring a "worse" type does not pay off.

Interestingly, Porters mechanism is essentially best possible.

Theorem 27 (Porter 2004)**.** *Under a number of (weak) conditions and assuming that $k > 1$, no deterministic truthful online mechanism can have a performance guarantee better than $((1+\sqrt{k})^2+1)$ for the described single machine model.*

This is of special interest in view of the existence of an algorithm with performance guarantee $(1 + \sqrt{k})^2$ for the non-strategic online setting, due to Koren and Shasha (1995).

4.2.4 Online Scheduling on Parallel Machines with Job Agents

In this section, we consider an online scheduling problem on m parallel machines, discussed by Heydenreich, Müller, and Uetz (2006). In this model, job-agents with private types do not only have to report their types, but also they have to choose a machine. The fact that jobs choose a machine themselves requires the design of mechanisms that are no direct revelation mechanisms anymore. The job-agents are released online over time. Each job j has a release date r_j, a processing time p_j and a weight w_j. The weight w_j can be interpreted as the cost to agent j of one unit of time spent waiting. Preemption is not allowed. A job j's valuation for a schedule that yields completion time C_j is $-w_jC_j$. Each job j has to make a report $(\hat{r}_j, \hat{p}_j, \hat{w}_j)$ about its type $t_j = (r_j, p_j, w_j)$. We assume $\hat{r}_j \geq r_j$ and $\hat{p}_j \geq p_j$ for the same reasons as in the previous section. The objective function that we seek to minimize is the weighted sum of completion times $\sum_{j\in J} w_jC_j$. Hence, the exact allocation algorithm for this problem is an affine maximizer. However, note that we are in an online situation. Vestjens (1997) has proven a lower bound of 1.309 for the performance guarantee of any online (allocation) algorithm for this model. Hence, as in the previous section there is no exact allocation algorithm.

Heydenreich et al. (2006) introduce and analyze the Decentralized Local Greedy mechanism, where the Local WSPT policy as defined in Section 3.3 is used as local sequencing policy on the machines. The motivation to use the WSPT policy locally is the fact that it yields the optimal schedule on a single machine, given that no release dates are present (Smith 1956). The mechanism works as follows:

> **Decentralized Local Greedy mechanism.**
> *Local Sequencing:* Whenever a machine becomes idle, it starts processing the job j with highest ratio w_j/p_j among all available jobs that have chosen this machine.

Assignment: Job j arrives at time $\hat{r}_j$ and communicates $\hat{p}_j$ and $\hat{w}_j$ to all machines. The machines compute a tentative completion time and a tentative payment, on the basis of which the job selects a machine and pays the respective tentative payment. The payment is distributed over the already present jobs in a way described next.

The payment π_j for each job j is inspired by the VCG-payments (see also Example 19). Intuitively, each agent pays for the loss in the tentative utility that other agents experience due to j's presence. That is, when j selects machine i and there is a job k on machine i that is displaced by j according to the local sequencing policy, then the tentative completion time of k increases by $\hat{p}_j$. Therefore, k's utility decreases by $w_k\hat{p}_j$. Job j compensates k for this loss by paying $\hat{w_k}\hat{p}_j$.

As we are not dealing with a direct revelation mechanism, the notion of truthfulness does not apply. Instead, we would desire a dominant strategy equilibrium in which all jobs report truthfully and select a machine that results in a good performance. It turns out that such an equilibrium does not exist for the Decentralized Local Greedy mechanism. But, given the online setting and the lack of information about future job releases, it seems reasonable to look at so-called *myopic best responses*. A myopic best response is a strategy that maximizes a job's tentative utility at arrival or —alternatively— that maximizes a job's utility under the assumption that it was the last job to arrive.

Theorem 28 (Heydenreich, Müller, and Uetz 2006)**.** *For the* Decentralized Local Greedy *mechanism, the following is true:*

- *It is a* myopic best response *for every job to report truthfully about its type and to select a machine that maximizes the job's tentative utility at arrival.*
- *In the restricted strategy space where all jobs j report their true weight $\hat{w}_j = w_j$, truthful reporting r_j and p_j and selecting a machine that maximizes the tentative utility at arrival is a dominant strategy equilibrium.*
- *It is not possible to modify only the payment scheme while keeping the local sequencing policy such that the resulting mechanism has a dominant strategy equilibrium in which all jobs tell the truth.*
- *Given that all jobs play their myopic best response, the mechanism is 3.281-competitive.*

The performance guarantee can be derived by adopting some ideas introduced by Megow, Uetz, and Vredeveld (2006) for a slightly different (and non-strategic) setting. Indeed, one can observe that if jobs play their myopic best responses, then the distribution of jobs over machines is almost identical to that of the MinIncrease algorithm of Megow et al. (2006).

For this model and the described mechanism, it is not clear whether the competitive ratio of 3.281 is best possible. For the non-strategic setting, there exists an algorithm with a better competitive ratio of 2.62 due to Correa and Wagner (2005). However, the distribution of jobs over machines in their algorithm is strongly based on central coordination and does not seem suitable in a decentralized setting.

5 Conclusion

In this paper we have given an introduction to the application of game theory and mechanism design to scheduling models. Thereby we have chosen to limit the scope of models and techniques

in order to be able to present the most important techniques in detail. We see several avenues of research departing from this introduction.

Within the narrow scope of this paper the most promising research questions seem to us related to mechanism design in the presence of multi-dimensional types, exemplified by the conjecture by Nisan and Ronen mentioned at the end of Section 4.2.1. Roberts (1979) has shown that in cases where the type space is completely unrestricted, meaning that every agent can have any valuation for each of the outcomes, only affine maximizers are truthfully implementable. For more restrictive type spaces, Lavi, Mu'alem, and Nisan (2003) could show a similar characterization only for allocation algorithms that satisfy some additional properties. If the conjecture by Nisan and Ronen is true, it would show that in this particular case we cannot do better than using an affine maximizer, if we want to guarantee truthfulness. This would indicate that also in this case affine maximizers are the only truthfully implementable algorithms.

A second avenue is related to the discussion in Section 4.2.4, which illustrates that in decentralized models where job agents' strategies are a mix of machine choices and type revelation, and in addition agents arrive online, it seems to be difficult to provide mechanisms that are truthful and simultaneously have a good performance. It would be interesting to explore how far we can get with truthful mechanisms.

Third, we want to emphasize that the area of game theory and mechanism design has established theoretical models that cover many more issues than those treated in this introduction. For example, game theory knows a plentitude of refinements of equilibrium notions, which might be of practical relevance, and mechanism design is also interested in other criteria than only efficiency and truthfulness. For example, various definitions of fairness can be found in the literature. As a point of reference we mention here the literature on matching markets, e.g., Roth, Sotomayor, and Chesher (1990).

Finally, it is likely that also in scheduling we can benefit from the wealth of results that have recently been derived in the context of combinatorial auctions. We recommend the book edited by Cramton, Shoham, and Steinberg (2006) as an excellent reference. While from an optimization point of view combinatorial auctions deal "only" with a set packing problem, many of the game theoretic issues in their design are also relevant when the allocation rule is of a different nature, like assigning jobs to machines, and determining their start time on the machines. In combinatorial auctions those issues are, for example, revenue for the seller, collusion, bidding under multiple identities (shill-bidding), information revelation, and communication complexity. It turns out that a mechanism like the VCG mechanism, when applied to combinatorial auctions, performs badly in terms of such additional criteria, unless rather restrictive assumptions on the bidders valuations are fulfilled (Ausubel and Milgrom 2006). Also in scheduling, such considerations may yield interesting insights and may be of practical importance.

Glossary

action agents have to choose between possible actions in a game or mechanism, 4

algorithm in this paper a centralized device that computes a schedule based on public information and agents actions, in contrast to *local sequencing policies* used to denote by decentralized machines to compute the sequencing of jobs, 10

allocation algorithm algorithm that computes an outcome/schedule on the basis of all public information and all agents actions, 5

direct revelation mechanism A *mechanism* in which an agent's only allowed action is to report a type, which does not have to coincide with its true type, 14

dominant strategy equilibrium vector of strategies of all agents such that the strategy for each agent is a utility-maximizing strategy independent of the actions that the other agents chose, 14

dominant strategy incentive compatible see *truthful*, 14

local sequencing policy the rule by which machines sequence jobs allocated to them based on job characteristics. Examples are shortest processing time first (SPT), longest processing time first (LPT), and largest ratio of weight over processing time first (WSPT), 10

makespan latest completion time of any job in a certain schedule, 4

mechanism in this paper an *allocation algorithm* together with a *payment scheme*, 14

Nash equilibrium vector of strategies, one for each agent, such that each of them is a best response to the strategies of the other agents, 6

outcome in this paper the schedule resulting from a game or mechanism, 4

parallel machine scheduling scheduling problem with several machines, where each job has the same processing time on each machine, 3

payment scheme determines payments for all agents on the basis of the agents' actions, 5

performance guarantee provable bound on the ratio between the objective value of a solution produced by a particular algorithm/mechanism and the objective value of the optimal solution, 18

price of anarchy given an optimization problem with decisions taken partly by selfish agents, the maximum factor by which the objective value in equilibrium deviates from the optimal objective value, 7

related machine scheduling scheduling problem with several machines, each of which having their own speed, the processing time of a job on a certain machine is its unit processing time divided by the speed of the machine, 3

schedule assignment of jobs to machines, together with the specification of the time interval(s) when the job is processed, 4

strategy maps an agents type to an action, 5

truthful property of a *direct revelation mechanism*, where truth-telling is a utility maximizing strategy for each agent independent of the actions that the other agents choose, 14

truthfully implementable property of an *allocation algorithm*, if there exists a *payment scheme*, such that the resulting mechanism is truthful, 14

type private information of an agent, 4

uniform machine scheduling see *related machine scheduling*, 3

unrelated machine scheduling scheduling problem with several machines, where each machine has a separate processing time for each job, 3

utility in this paper, the valuation of an agent for a particular schedule, minus the payment the agent has to make (quasi-linear utility), 6

valuation expresses how much an agent appreciates a certain schedule (depending on its *type*), 5

weak monotonicity property of an allocation algorithm stating that the marginal benefit from reporting type t instead of type s is larger if the true type is t than if the true type is s, 16

References

Ambrosio, P. and V. Auletta (2005). Deterministic monotone algorithms for scheduling on related machines. In G. Persiano and R. Solis-Oba (Eds.), *Approximation and Online Algorithms*, Volume 3351 of *Lecture Notes in Computer Science*, pp. 267–280. Springer.

Andelman, N., Y. Azar, and M. Sorani (2005). Truthful approximation mechanisms for scheduling selfish related machines. In V. Diekert and B. Durand (Eds.), *Theoretical Aspects of Computer Science - STACS 2005*, Volume 3404 of *Lecture Notes in Computer Science*, pp. 69–82. Springer.

Angel, E., E. Bampis, and F. Pascual (2005). Truthful algorithms for scheduling selfish tasks on parallel machines. In X. Deng and Y. Ye (Eds.), *Internet and Network Economics*, Volume 3828 of *Lecture Notes in Computer Science*, pp. 698–707. Springer.

Archer, A. and E. Tardos (2001). Truthful mechanisms for one-parameter agents. In *Proc. 42nd Annual Symposium on Foundations of Computer Science*, pp. 482–491. IEEE Computer Society.

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

Auletta, V., R. De Prisco, P. Penna, and G. Persiano (2004). Deterministic truthful approximation mechanisms for scheduling related machines. In V. Diekert and M. Habib (Eds.), *Theoretical Aspects of Computer Science - STACS 2004*, Volume 2996 of *Lecture Notes in Computer Science*, pp. 608–619. Springer.

Ausubel, L. and P. Milgrom (2006). The lovely but lonely vickrey auction. In P. Cramton, Y. Shoham, and R. Steinberg (Eds.), *Combinatorial Auctions*, pp. 17–40. MIT Press.

Baruah, S., G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang (1992). On the competitiveness of on-line real-time task scheduling. *Journal of Real-Time Systems 4*(2), 125–144.

Bikhchandani, S., S. Chatterjee, and A. Sen (2004, September). Incentive compatibility in multi-unit auctions. Working Paper.

Briest, P., P. Krysta, and B. Vöcking (2005). Approximation techniques for utilitarian mechanism design. In *Proc. 37th Annual ACM Symposium on Theory of Computing*, pp. 39–48. ACM.

Christodoulou, G., E. Koutsoupias, and A. Nanavati (2004). Coordination mechanisms. In J. Díaz, J. Karhumäki, A. Lepistö, and D. Sannella (Eds.), *Automata, Languages and Programming*, Volume 3142 of *Lecture Notes in Computer Science*, pp. 345–357. Berlin: Springer.

Clarke, E. H. (1971). Multipart pricing of public goods. *Public Choice 11*, 17–33.

Cole, R., Y. Dodis, and T. Roughgarden (2006). How much can taxes help selfish routing? *Journal of Computer and System Sciences 72*(3), 444–467.

Correa, J. R. and M. R. Wagner (2005). LP-based online scheduling: from single to parallel machines. In M. Jünger and V. Kaibel (Eds.), *Integer Programming and Combinatorial Optimization*, Volume 3509 of *Lecture Notes in Computer Science*, pp. 196–209. Springer.

Cramton, P., Y. Shoham, and R. Steinberg (Eds.) (2006). *Combinatorial Auctions*. MIT Press.

Czumaj, A. (2004). Selfish routing on the internet. In J. Leung (Ed.), *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, Chapter 42. CRC Press.

Czumaj, A. and B. Vöcking (2002). Tight bounds for the worst-case equilibria. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 413–420. ACM-SIAM.

Finn, G. and E. Horowitz (1979). A linear time approximation algorithm for multiprocessor scheduling. *BIT Numerical Mathematics 19*(3), 312–320.

Graham, R. (1966). Bounds for certain multiprocessing anomalies. *Bell System Technical Journal 45*, 1563–1581.

Groves, T. (1973). Incentives in teams. *Econometrica 41*, 617–631.

Gui, H., R. Müller, and R. Vohra (2004, October). Dominant strategy mechanisms with multidimensional types. Discussion Paper 1392, The Center for Mathematical Studies in Economics & Management Sciences, Northwestern University, Evanston, IL.

Heydenreich, B., R. Müller, and M. Uetz (2006). Decentralization and mechanism design for online machine scheduling. In L. Arge and R. Freivalds (Eds.), *Algorithm Theory - SWAT 2006*, Volume 4059 of *Lecture Notes in Computer Science*, pp. 136–147. Springer.

Ibarra, O. and C. Kim (1977). Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM 24*(2), 280–289.

Immorlica, N., L. Li, V. S. Mirrokni, and A. Schulz (2005). Coordination mechanisms for selfish scheduling. In X. Deng and Y. Ye (Eds.), *Internet and Network Economics*, Volume 3828 of *Lecture Notes in Computer Science*, pp. 55–69. Springer.

Kawaguchi, T. and S. Kyan (1986). Worst case bound of an LRF schedule for the mean weighted flow-time problem. *SIAM Journal on Computing 15*(4), 1119–1129.

Kittsteiner, T. and B. Moldovanu (2005). Priority auctions and queue disciplines that depend on processing time. *Management Science 51*, 236–248.

Koren, G. and D. Shasha (1995). D-over: An optimal on-line scheduling algorithm for overloaded real-time systems. *SIAM Journal on Computing 24*(2), 318–339.

Koutsoupias, E. and C. Papadimitriou (1999). Worst-case equilibria. In C. Meinel and S. Tison (Eds.), *Theoretical Aspects of Computer Science*, Volume 1563 of *Lecture Notes in Computer Science*, pp. 404–413. Springer.

Kovacs, A. (2005). Fast monotone 3-approximation algorithm for scheduling related machines. In G. S. Brodal and S. Leonardi (Eds.), *Algorithms - ESA 2005*, Volume 3669 of *Lecture Notes in Computer Science*, pp. 616–627. Springer.

Lavi, R., A. Mu'alem, and N. Nisan (2003). Towards a characterization of truthful combinatorial auctions. In *Proc. 44th Annual Symposium on Foundations of Computer Science*, pp. 574–583. IEEE Computer Society.

Lawler, E. L., J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys (1993). Sequencing and scheduling: Algorithms and complexity. In *Logistics of Production and Inventory*, Volume 4 of *Handbooks in Operations Research and Management Science*, pp. 445–522. Amsterdam: North-Holland.

Megow, N., M. Uetz, and T. Vredeveld (2006). Models and algorithms for stochastic online scheduling. *Mathematics of Operations Research*. to appear.

Mitra, M. (2001). Mechanism design in queueing problems. *Economic Theory 17*, 277–305.

Mitra, M. (2005). Incomplete information and multiple machine queueing problems. *European Journal of Operational Research 165*, 251–266.

Monderer, D. and L. S. Shapley (1996). Potential games. *Games and Economic Behavior 14*(1), 124–143.

Nisan, N. and A. Ronen (2000). Computationally feasible VCG mechanisms. In *Proc. 2nd ACM Conference on Electronic Commerce*, pp. 242–252. ACM.

Nisan, N. and A. Ronen (2001). Algorithmic mechanism design. *Games and Economic Behavior 35*, 166–196.

Owen, G. (1995). *Game theory* (Third ed.). San Diego, CA: Academic Press Inc.

Porter, R. (2004). Mechanism design for online real-time scheduling. In J. S. Breese, J. Feigenbaum, and M. I. Seltzer (Eds.), *Proc. 5th ACM Conference on Electronic Commerce*, pp. 61–70. ACM.

Pruhs, K., J. Sgall, and E. Torng (2004). Online scheduling. In J. Y.-T. Leung (Ed.), *Handbook of Scheduling*, Chapter 15. CRC Press LLC.

Roberts, K. (1979). The characterization of implementable choice rules. In J.-J. Laffont (Ed.), *Aggregation and Revelation of Preferences*. North Holland Publishing Company.

Rochet, J.-C. (1987). A condition for rationalizability in a quasi-linear context. *Journal of Mathematical Economics 16*, 191–200.

Ronen, A. (2006). Incentive compatibility in computationally feasible combinatorial auctions. In P. Cramton, Y. Shoham, and R. Steinberg (Eds.), *Combinatorial Auctions*, pp. 369–394. MIT Press.

Rosenthal, R. W. (1973). A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory 2*, 65–67.

Roth, A., M. A. O. Sotomayor, and A. Chesher (Eds.) (1990). *Two-Sided Matching : A Study in Game-Theoretic Modeling and Analysis*. Cambridge University Press.

Roughgarden, T. (2004). Stackelberg scheduling strategies. *SIAM Journal on Computing 33*(2), 332–350.

Saks, M. and L. Yu (2005). Weak monotonicity suffices for truthfulness on convex domains. In *Proc. 6th ACM conference on Electronic commerce*, pp. 286 – 293. ACM.

Schuurman, P. and T. Vredeveld (2006). Performance guarantees of local search for multiprocessor scheduling. *INFORMS Journal on Computing*. to appear.

Smith, W. (1956). Various optimizers for single stage production. *Naval Research Logistics Quarterly 3*, 59–66.

Vestjens, A. P. A. (1997). *On-line Machine Scheduling*. Ph. D. thesis, Eindhoven University of Technology, Eindhoven, The Netherlands.

Vickrey, W. (1961). Counterspeculation, auctions and competitive sealed tenders. *J. Finance 19*, 8–37.