Sorbonne Université
Faculté des Sciences et Ingénierie

Thesis entitled

Models and algorithms for combinatorial optimization problems of resource sharing between agents.

by

Fanny Pascual

for the diploma

Habilitation à Diriger des Recherches

Contents

Introduction

In a combinatorial optimization problem, the aim is to find an optimal solution, given a predefined objective function, among a finite set of possible solutions. The set of possible solutions is generally defined by a set of restrictions, and is usually of exponential size, and thus too large for exhaustive search. Since the invention of computers during the last century, numerous researchers in computer science and in applied mathematics have aimed to design exact or approximate polynomial time algorithms for such problems. For a long time, these problems have been regarded as problems involving one decision maker, who gives the objective function. For example, a decision maker aimed to schedule his/her tasks on his/her machines in a way that minimizes the maximum completion time of the tasks, or a decision maker aimed to compute a minimum weight Hamiltonian cycle (traveler salesman problem).

Researchers then considered combinatorial optimization problem involving several objective functions, but still one decision maker (the objective functions represent conflicting objectives of the decision maker): this lead to the *multi-criteria decision making* field at the end of the 1970s. This field is still very active today. At the end of the 1990s and in the beginning of the 21st century, with the development of Internet, computer scientists considered systems where people interact together, each one aiming to optimize his or her own objective function. This led to the fields of *algorithmic game theory* and *algorithmic mechanism design*. The field of *computational social choice* also emerged in this context, in the beginning of the 2000s.

The aim of this manuscript is to present the main concepts and techniques in the domains I have worked on since the end of my PhD thesis, and to present my contributions. In this document, we will consider problems involving different independent users, each one having his/her own objective function (or his/her preferences) – these users will be called *agents*. A social choice function maps a set of individual preferences to a solution. We will most of the time consider the utilitarian social choice function (which aims to minimize the sum of the costs of the agents), or the egalitarian social choice function (which aims to minimize the maximum cost of an agent), to analyse the quality of a solution.

We will usually consider that the agents share resources. These resources can for example be machines, when agents have tasks to schedule on a set of shared machines, or these resources can be goods when agents have to share a set of goods. We will follow several approaches:

- In the first chapter, we will consider that the agents do not have power on the algorithm that assigns the resources to the agents: a decision maker – the one that designs the algorithm – has thus to compute a solution which is good for the considered social choice function. In this chapter, we will present the approach that we will take to tackle a combinatorial optimization problem (this approach will also be used in the following chapters). It is also a pretext to present a work that we have done on the assignment of tasks of different agents to machines in data-centers.

- In the second chapter, we will consider that the agents have a possibility of action (a set of possible strategies) that impacts the solution obtained. Each agent will behave in order to optimize his or her own objective function. Such situations can be modeled as games (strategic settings where agents act rationally), and fall in the domain of algorithmic game theory. We will be interested in the dynamics obtained in such situations, and by the quality of equilibria (if any) obtained. The quality of an equilibrium, as in the previous chapter, is usually measured with respect to the egalitarian social choice function or the utilitarian social choice function.

- In the third chapter, we will consider that the agents still have a possibility of action in the system (a set of possible strategies), but that there is a trusted third party that can propose a solution to the agents. The aim of the trusted third party – our aim – will be to return a good solution with respect to the social choice function, while returning a solution that the agents will have incentive to accept.

- In the fourth chapter, we will consider situations in which the agents have private information (data that the system designer and the other agents do not know). Given the values bidden by the agents, an algorithm will compute a solution (for example an allocation of the resources to the agents). Since the objective of each agent is to optimize his/her own objective function, an agent can bid false information to the algorithm if this allow him/her to get a smaller cost or a larger profit. These situations fall in the domain of algorithmic mechanism design. Our aim will be to design algorithms that are resistant to the manipulation of the agents.

- Whereas in the first four chapters we usually consider that the agents wish to optimize their access to the shared resources, in the fifth and last chapter, we will consider that the resources are not shared, but are *collective*. A decision has to be done on these collective resources, and each agent has his/her own preferences on the solutions which can possibly be returned. For example, a schedule of collective tasks has to be returned (the agents having their preferences on the order of the tasks), or an Hamiltonian cycle has to be returned (the agents having their preferences on the edges chosen). The aim is to return a "good" solution given these preferences. These situations will be considered under the multi-criteria decision making field, and the computational social choice field.

The main applications that we will consider in this manuscript are scheduling and assignment problems. When we will introduce a new scheduling problem, we will use *this police* to define it, and to output its difference with standard scheduling problems.

Finally, the bibliography is divided into two parts: the publications corresponding to my own works, cited with the initials of the authors followed by the year of publication (style *alpha* in latex), and the other publications, cited as a number (style *plain* in latex).

Chapter 1

Agents do not have power: optimization with centralized algorithms

In this chapter, we will consider combinatorial optimization problems in presence of a set $\mathcal{A}$ of agents who wish to minimize their costs. Contrarily to what we will see in the next chapters, we consider that the agents do not have the power to influence the algorithm. Given data, which are known and non manipulable by the agents, the aim is to construct a solution which is good for a considered social choice function (a global objective function). Let c_i denotes the cost of agent i. The two main classic aggregation operators on the costs of the agents are the following ones:

Utilitarian social choice function: $\sum_{a\in\mathcal{A}} c_i$**.** The aim is to minimize the sum of the costs of the agents, which is similar to minimizing the average cost of an agent.

Egalitarian social choice function: $\max_{a\in\mathcal{A}} c_i$**.** The aim is to minimize the maximum cost of an agent.

In the sequel, these two main social choice functions will measure the quality of the solution obtained[1]. Note that these aggregation operators are commonly used also for problems where there is only one agent which has several tasks to do. This is particularly true for scheduling problems, where the aim is usually to minimize the maximum cost of a task, or to minimize the sum of the costs of the tasks. For example, when the cost of a task is its completion time, problem $P||C_{max}$ aims to return a schedule which minimizes the maximum completion time, while problem $P||\Sigma C_i$ aims to return a schedule which minimizes the sum of the completion times[2]

In this chapter, we will illustrate different techniques to deal with such problems on an assignment problem that Krzysztof Rzadca and I we have introduced [PR15]. In Section 1.2, we start by motivating this new problem, which consists in assigning tasks to machines in data centers in order to optimize the average cost or the maximum cost of a task – of an agent, if each agent has a task. We will also briefly compare this measure to other performance measures used in this context. We start by a brief overview of techniques used to tackle an NP-hard combinatorial optimization problem.

[1]The L_p norm: $\sqrt[p]{\sum_{a\in\mathcal{A}} (c_i)^p}$, with a parameter $p \geq 1$, will also be used in the last chapter. The L_p norms form a spectrum of aggregations between the sum (L_1) and the max (L_∞).

[2]We use the three fields notation (α, β, γ) introduced by Graham et al. [89]: the first field, α, describes the machine environment, the second field, β, the task characteristics and constraints, and the third field, γ, the objective function. The input of problems $P||C_{\max}$ and $P||\Sigma C_i$ is the same: a number of identical parallel machines, a set of tasks T, and for each task $i \in T$ a length (processing time) p_i. A *schedule* (without preemption) assigns to each task a start time and a machine, in a way that no two tasks are executed at the same time on the same machine. The *completion time*, C_i of task $i \in T$ is the date at which it is completed: it is its start time plus its length. $C_{\max} = \Sigma_{i\in T} C_i$ denotes the maximum completion time of a task, and is called the *makespan*.

1.1 Tackling a NP-hard optimization problem.

When tackling with a new combinatorial optimization problem, the usual way to proceed is to start by studying the complexity of the problem. If it is NP-hard, then we can extract and solve polynomial time sub-cases of the problem, and solve with exact algorithms these problems. This is often done by designing dynamic programming algorithms (see Section 1.3.3).

Exact algorithms.
Several approaches can be used in order to solve an NP-hard problem with an exact algorithm. A first approach consists in designing algorithms which are polynomial when one parameter of the problem is fixed. This is the domain of parameterized algorithms [60] (see also Section 1.3.3). We can also design exact algorithms which are not polynomial but which have a power in the exponential term which is as low as possible. This is the aim of the super polynomial time algorithms field [76]. Branch and bound algorithms and mathematical programming techniques [49] can be used to return an exact solution in a delay much shorter that brute force algorithms. We can also resort to randomized algorithms which output an exact solution with an expected running time which is polynomial (Las Vegas algorithms) [127].

Approximate algorithms.
In order to deal with large instances, we need often to resort to algorithms which do not always output optimal solutions. One way is to consider randomized algorithm which output an exact solution with a high probability, but which can also output a non exact solution (as Monte Carlo algorithms), or fail to return an optimal solution [127].

Another way to proceed is design approximation algorithms [168], i.e. approximate algorithms with a guarantee on the quality of the solution returned. Given $\rho \in \mathbb{R}^+$, an algorithm is said to be ρ-approximate it always returns a solution whose cost is at most ρ times the optimal cost for a minimization problem (and at least ρ times the optimal profit for a maximization problem). The aim is to design a ρ-approximate algorithm, with an approximation ratio ρ as close as possible to 1. In Section 1.3.4, we will focus on the case where ρ can be as close as 1 as we wish and design polynomial time approximation schemes.

Since it is not always possible to get approximate algorithm with good approximation ratio (and it is sometimes even not possible to get an approximation algorithm with a constant approximation ratio – see Section 1.3.1), or since approximate algorithms are not always the best ones in practice, it is sometimes useful to produce heuristics, which returns solutions which are often close to the optimal ones. These heuristics can be designed using properties of the problem (see Section 1.3.5), and/or by using metaheuristics [66].

We start by presenting and motivating the problem which will be used through this chapter.

1.2 An illustrative problem: assignment of tasks with types

1.2.1 Context

Data centers, composed of tens to hundreds of thousands of machines, packaged as virtual machines or services, and denoted as “cloud”, are changing the way the industry (and, to some extent, academia and research) computes. While the large number of machines make think about High Performance Computing (HPC) supercomputers, resource management in data centers significantly differs from scheduling jobs on typical HPC supercomputers.

In their great majority, HPC workloads are composed of computationally intensive jobs only (although some recent HPC workloads may also be memory intensive, which requires changes to HPC resource managers [112]). The goal of an HPC scheduler is to order jobs so that they are completed as fast as possible, taking into account site's policies, fairness and efficiency. As jobs are computationally intensive, they all compete for the same resource – the CPU. So, a single node executes at most as many jobs as CPU cores.

In contrast, a data center workload is more varied, and require heterogeneous resources (CPU, memory, hard disk bandwidth, network bandwidth) [146, 42]. Furthermore, there are relatively few HPC-like computationally-intensive jobs. For example, in the Google trace [146], just 1.5% of applications contribute to 98.5% of CPU usage [62]. Thus, in a data center, a task usually does not saturate the resources of a single node [106]. Services are varied, from user-facing web applications to databases, or message-passing infrastructures. In the sequel, we will use the term *task* to denote a single instance of a service or a single job.

Another important difference between jobs in HPC and tasks in data centers is that certain classes of data center tasks, such as web servers or databases, almost persistently serve user traffic and never "complete". Thus, in contrast to scheduling in HPC, the aim is not to complete such tasks as soon as possible, but rather to allocate sufficient resources so that its perceived performance is satisfactory.

These new features of data center workloads make HPC models unsuitable for managing resources of a data center. As tasks require heterogeneous resources, sharing a single machine among many services is reasonable, and even desirable. Ideally, resource requirements of collocated tasks should complement each other, e.g., a memory-intensive database instance should be allocated with an IO-intensive web application.

In the following section, we will look at existing models for data center resource allocation, and introduce an alternative model that captures the complex performance relations that tasks have on each other when they are allocated to the same machine. This model will lead to two combinatorial optimization problems where we wish to maximize the average performance of a task, or to maximize the worst performance of a task. These problems will allow us to illustrate a panel of techniques used to solve a combinatorial optimization problem. Let us first define our model, and the associated problems.

1.2.2 Model and problems

We consider a system that assigns n tasks $J = \{1, \dots, n\}$ to m identical machines $\mathcal{M} = \{M_1, \dots, M_m\}$. Each task i has a known size $p_i \in \mathbb{N}$, and a type $t_i \in \mathcal{T}$, where $\mathcal{T}$ is a set of types. The size of a task corresponds to the amount of resources it needs (thus, if two tasks i and j of the same type are such that $p_j = 2p_i$, then task j will need twice more resources of each kind than task i).

A type can correspond to a specific application (as in [110]), but it can also be more general, gathering, for example, all web servers under a single type, and all databases under another one. Types are more or less compatibles: for example two types competing for same resources (CPU, bandwidth, memory, etc), will probably be less compatible than two types that need different resources. A coefficient $\alpha_{t,t'} \in \mathbb{R}^+$ is defined for each pair of types $(t, t') \in \mathcal{T}^2$. It measures the impact of the tasks of type t on the cost of the tasks of type t' (allocated on the same machine). If $\alpha_{t,t'} = 0$ then a task of type t has no impact on the cost of a task of type t'. The higher $\alpha_{t,t'}$ is, the larger the impact is.

A *partition* is an assignment of each of the n tasks to one of the m machines. It separates the tasks into at most m subsets: each subset corresponds to the tasks allocated to the same machine. Given a partition P, we denote by $M_{P,i} \in \mathcal{M}$ the machine on which task i is allocated. We express performance of a task i by a cost function c_i. To simplify the presentation of our results, we prefer to express our problems as minimization of costs, rather than maximization of performance. The cost c_i of task i depends on the *total load* of tasks j allocated to the same machine $M_{P,i}$, but different types have different impacts:

$$c_i = \sum_{j \text{ on machine } M_{P,i}} p_j \cdot \alpha_{t_j,t_i} \tag{1.1}$$

We denote by SumCT (SumCosts with Types) the problem of finding a partition minimizing the sum of the costs: $\Sigma_{i \in \{1,\dots,n\}} c_i$. The optimal partition minimizes the average cost a task experiences in the system, and thus corresponds to the utilitarian social choice function.

We denote by MaxCT (MaxCost with Types) the problem of finding a partition minimizing the maximum cost: $\max_{i \in \{1,\dots,n\}} c_i$. The optimal partition minimizes the worst cost a task experiences in the system, and thus corresponds to the egalitarian social choice function.

The novelty of these problems, compared to classical scheduling or load balancing problems, is to introduce types. The impact of a task on another task on the same machine does not depend only of its size - or length - but also of its type.

Relation with classical scheduling problems.

When all the tasks are of the same type, the cost of a task is the load of the machine on which it is allocated to. In this case, problem MaxCT is equivalent to the classical multiprocessor scheduling, $P||C_{\max}$ [90], where the aim is to minimize the makespan.

Likewise, in selfish load balancing games [115, 169], it is assumed that the cost of each task is the total load of the machine (their model does not consider types). This model has been extensively studied in the context where each agent has a task for which he chooses the machine on which it will be scheduled – this problem will be seen in Chapter 2. In these works, the aim is to bound the price of anarchy, which is the loss of performance in the sum of the costs of the agents due to the fact that agents have the power to choose the machine of their tasks. Interestingly, the problem which aims at minimizing the sum of the costs of the agents in a centralized setting had not been studied. As we will see in Section 1.3.3, this problem, which is in fact problem SumCT with a single type, can be solved polynomially.

Other approaches.

Before considering other approaches to assign tasks to machines in data centers, note that the coefficients $\alpha_{t,t'}$ in our model can be estimated by monitoring the performance of the tasks as a function of their collocation and their sizes (a data center runs many instances of similar services). Numerous works [110, 140, 173, 174] show how to form a performance model as a function of collocation.

There is no standard model of data center resource management (standard in the sense in which the parallel task model is standard for HPC). Existing models can be roughly categorized into variants of multi-dimensional bin-packing [165, 63, 162, 166, 103, 165, 83, 119] (to model heterogeneous resource requirements), stochastic optimization [87, 40, 172] (to model uncertainty), and statistical approaches [25]. Other issues include pricing and revenue management [143] (in our model, the provider does not select tasks to execute); or distributed scheduling [157] (our model targets a single data center, rather than a distributed cloud).

Papers relying on bin-packing have the same aim as us: taking into account the heterogeneity of the tasks. In this case, bin packing is extended to vector packing: an item's size is defined as a vector with dimensions corresponding to requirements on individual resources (CPU, memory, disk or network bandwidth). They implicitly assume a crisp performance model: as long as the total demand of collocated tasks remains below the resources available on a machine, the tasks' performance is considered satisfactory. Furthermore, bin packing approaches do not permit even small overpacking, while data center resource managers commonly oversubscribe at least for CPU [146], leading to probabilistic service level agreements. With bin-packing approaches, complex inter-tasks performance degradation have to be modeled by placement constraints [83, 119].

1.3 Approaches

In this section, we briefly introduce some classical notions presented in Section 1.1, and illustrate them mainly on problems presented in the previous section.

1.3.1 Complexity and inapproximability

As said before, the first step when tackling a new problem is to determine its complexity. Problems SumCT and MaxCT are strongly NP-hard [PR15, PR17]. This is true even when the tasks have all the same unit length, showing that types add complexity (indeed, the same problems without types – i.e. problems $(P|p_i = 1|C_{\max})$ and $(P|p_i = 1|C_{\max})$ – are polynomially solvable). We can even show a stronger result for MaxCT, by using gap creation arguments:

Theorem 1.1 ([PR17]) *There is no polynomial time r-approximate algorithm for* MaxCT*, for any number $r > 1$, unless $P = NP$. This is true even if all the tasks have unit size.*

As we cannot hope for a polynomial time approximate algorithm for MaxCT, we will restrict to particular cases. The case where the number of types is low is interesting, since, in practice, while the number of tasks and machines can be huge, the number of types is limited. When the number of types is constant, we get a polynomial time approximation scheme (see Section 1.3.4).

Complexity and inapproximability results usually hold under the hypothesis that $P \neq NP$. Note that other hypothesis, less strong, can be used, when it is not easy to determine whether a problem is strongly NP-complete or not.

1.3.2 Dominance rules

For a given combinatorial optimization problem, dominance rules (or dominance properties) are properties that are fulfilled by at least one optimal solution of the problem, for each possible instance. Expressing dominance rules for a problem reduces the number of solution to explore, by restricting the search of an optimal solution among the feasible solutions which fulfill the dominance property. While the number of feasible solutions is usually exponential, the number of solutions which fulfill the property can be polynomial with a good dominance rule, and in this case even a brute force algorithm looking at these solution takes a polynomial time.

In a work with Krzysztof Rzadca [PR15, PR18], we propose a dominance rule called *the Ordered Sizes (OS) property* for problem SumCT: take three tasks s (small), x (medium), l (large) of the same type and of sizes $p_s < p_x < p_l$. An allocation breaks the OS property if s and l are assigned to the same machine and x to another machine. An allocation fulfills the OS property if no triple breaks it.

Lemma 1.1 *([PR15]) For each instance of* SumCT *there exists an optimal allocation which fulfills the OS property.*

This dominance property allows us to get a "brute force algorithm" which looks at all the possible solutions which fulfill the OS property. This algorithm is polynomial in the number of tasks but exponential in the number of types and the number of machines [PR15]. As we will see in the next section, we can also use this dominance rule in dynamic programming algorithms, which are a class of algorithms in which dominance properties are very useful. Dominance rules are also very useful in branch and bound algorithms: instead of exploring a tree with all the feasible solutions, we can explore a tree in which the leafs are the solutions which fulfill the dominance rule.

1.3.3 Dynamic programming algorithms

Dynamic Programming is a method, formalized by Richard Bellman [20], which solves a problem by breaking it down into a collection of simpler subproblems, solving each of these subproblems just once, and storing their solutions using a memory-based data structure (such as an array). This technique of storing solutions to subproblems instead of recomputing them is called memoization.

Dynamic programming algorithms are often used to solve combinatorial optimization problems when no clear structure appears. The only thing we need is to express the optimal solution of a problem in a function of optimal solutions of subproblems – "smaller" instances of the same problem – (this is called the optimality principle of Bellman). Moreover, it is usually very easy to compute the number of subproblems considered, and thus it is easy to determine the complexity of such an algorithm.

Dynamic programming can lead to polynomial time optimal algorithms for problems which are in P. For example, with Safia Kedad-Sidhoum and Siao-Leu Phouratsamay, we have determined the best solution of the classical two level lot sizing problem[3] when the capacity of storage is limited at the retailer's level. For this, we have expressed the cost of the supply chain in function of the cost of the supply chain in smaller time intervals at each level [PKP18]. Dynamic programming is often used in scheduling, planning, and text algorithms[4], since it is often natural to express the

[3]The two level lot sizing problem (2ULS) is a generalization of the the dynamic Uncapacited Lot-Sizing problem (ULS), introduced by Wagner and Within [171]. In a ULS problem, we know a forecast of product demand d_t over a time horizon $t \in \{1, \ldots, n\}$. There is a setup cost s_t incurred for each production done at period t, a processing cost p_t incurred for each unit of production done at period t, and an inventory (holding) cost h_t for each unit of product stored between periods t and $t + 1$. The aim is to decide at which period we should produce – and how many units – in order to minimize the total cost (this is the cost of the supply chain). In the two level lot sizing problem, there are two levels: one supplier and one retailer, with production/storage at each level (the units produced by the supplier being either stored at the supplier level, or sent at the retailer level). The aim is also to minimize the total cost of the supply chain (the sum of the costs at each level).

[4]LaTeX uses a dynamic programming algorithm to do the text justification of this document.

optimal solution of a problem in these areas in a function of the optimal solutions of subproblems (planning in a shortest time interval, sub-words, etc.).

It is possible to exploit dominance rules to get the optimality principle of Bellman. For example, in [PR18], we used the "Ordered Sizes" dominance property presented in the previous section to obtain a polynomial time algorithm to solve SumCT when there is only one type. As said before this is the optimal, centralized, solution of load balancing algorithmic games, extensively studied in algorithmic game theory [170] – the papers usually study the efficiency of decentralized solutions via the price of anarchy (see Chapter 3). This dominance property also allows to polynomially solve problem SumCT with a fixed number of types and a fixed number of different sizes [PR18].

1.3.4 Approximation schemes

For a minimization (resp. a maximization) problem P, an approximation scheme is a family of $(1+\varepsilon)$-approximate algorithms (resp. $(1-\varepsilon)$)-approximate algorithms) A_ε over all $0 < \varepsilon < 1$. A polynomial time approximation scheme (PTAS) [84] is an approximation scheme whose time complexity is polynomial in the input size for any fixed ε, and a fully polynomial time approximation scheme (FPTAS) is an approximation scheme whose time complexity is polynomial in the input size and also polynomial in $1/\varepsilon$.

Approximation schemes are designed either by structuring the input or the output of the problem. When structuring the input, an instance is simplified, solved (in polynomial time) on the simplified instance, and the solution obtained on the simplified instance is translated back to the original problem. The instance is often simplified by rounding the numbers of the input; by merging small pieces into larger pieces of primitive shape; by cutting irregular shaped pieces form the instance; or by aligning the shapes of several similar items [156].

The terms PTAS and FPTAS have been introduced in a seminal paper by Garey and Johnson [84], and many approximation schemes have been designed for classical computer science problems since the 1970s. One of the most famous approximation schemes is the one of Hochbaum and Shmoys [97, 98] for problem $(P||C_{\max})$. This PTAS uses dichotomic search to find a minimal target makespan C. During the search, for a certain C, either the algorithm detects that the optimal makespan is larger than C, or it returns a schedule with a makespan at most $(1+\varepsilon)C$ (this is called *dual approximation*). In order to build a schedule close to the optimum, the algorithm partitions the tasks into two sets: the long tasks and the small tasks. The long tasks are rounded down to the nearest multiple of some given number X (which depends on ε). These tasks are scheduled optimally using dynamic programming. The small tasks are then added using a list scheduling algorithm at the end of the schedule obtained for the large tasks. In order to solve MaxCT, we have adapted this PTAS to take into account the types of the tasks, and our new cost function:

Theorem 1.2 ([PR19]) *There exists a PTAS for problem* MaxCT.

Our PTAS has a similar structure as the PTAS of Hochbaum and Shmoys described above: it uses dichotomic search to find a minimal target cost C. The algorithm also partitions the tasks into two sets: the long tasks and the small tasks, and the long tasks are rounded down to the nearest multiple of some given number X (which depends of ε and $\max_{t,t' \in \mathcal{T}} \alpha_{t,t'}$). As in the PTAS of Hochbaum and Shmoys, this corresponds to the technique of "rounding the numbers of the input" mentioned above. We add a step corresponding to the technique "merging small pieces into larger pieces of primitive shape" by gathering small tasks of a same type together in some new long tasks of size X, called containers. This gives us a modified instance made of the rounded long tasks and the newly introduced container tasks. These tasks are scheduled optimally using a dynamic programming algorithm.

Concerning problem SumCT we have designed a PTAS for this problem when the number of types is constant in [PR18]. This PTAS is obtained using rounding and a dynamic programming algorithm based on the OS property described in Section 1.3.2.

Scheduling tasks on machines with unavailability periods.
In a work with Florian Diedrich, Klaus Jansen and Denis Trystram, we have also designed a PTAS for scheduling tasks on identical parallel machines with unavailability periods (reservations) [DJPT07, DJPT10]. We are given a set of tasks and m machines, and the machines may be

"reserved" by other tasks/events during some time intervals: on each machine there can be some time intervals during which the machine is unavailable (no task can be executed on the machine during this time interval). The aim is to schedule the tasks (without preemption) on these machines in order to minimize the makespan. It is easy to see that this problem in inapproximable within a constant approximation ratio even for 2 machines (by reduction to partition: on each machine there is a large reservation starting at each machine at time $B = \sum p_i/2$).

Since this problem is inapproximable, we have introduced a variant of it: we consider that at least one machine is always available (i.e. without any reservation). The problem remains strongly $\mathcal{NP}$-hard but we can get an approximation ratio as close to 1 as we wish.

For this new problem (a least one machine has no reservation), we introduce a PTAS, which is based on dual approximation, and which uses algorithms for multiple subset sum problems. This is the best result we can have, from an approximation point of view, since the problem is strongly NP-hard, and this excludes the possibility of a FPTAS unless P=NP. We also give a FPTAS for a special case (when there are only two machines) [DJPT10].

1.3.5 Heuristics

Ideally, we would like to get, for NP-hard problems, fast polynomial time approximation schemes. In practice, this is not always possible, because of intrinsic constraints (inapproximability bounds), or because a PTAS, in practice, takes a too long time when the instances become large. Therefore, we often design fast heuristics which, if possible, are approximate algorithms with a performance guarantee (even if this is not the best possible performance guarantee), and which, in practice, return solutions close to the optimal. We will illustrate this on problems SumCT and MaxCT. Let us first present how data have been generated for these problems.

We conducted simulations on a dataset derived from a data center trace published by Google [146]. This trace describes all tasks running during a month on one of the Google clusters. For each task, the trace reports in its task record table, among other data, the task's CPU, memory and disk IO usage averaged over a 5-minute long period. We generated a random sample of 10 000 tasks among the tasks in this trace, and we decided how many types we wish to have (for example 4). To generate loads and types, we use data on the mean CPU and memory utilization.

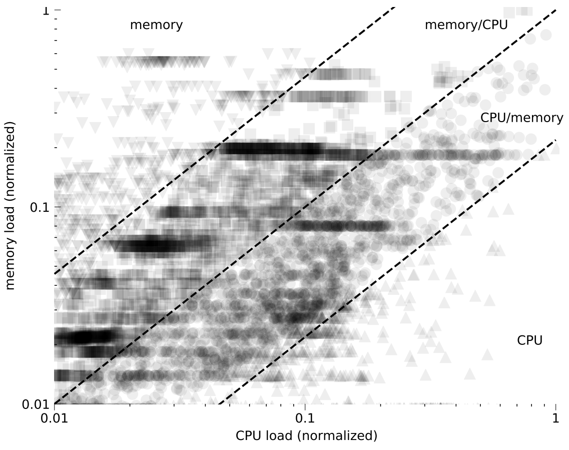
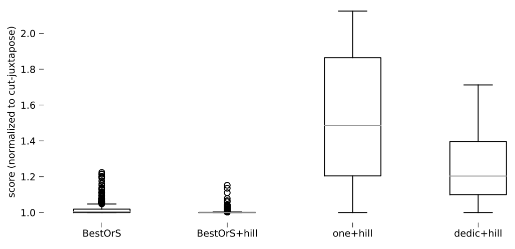
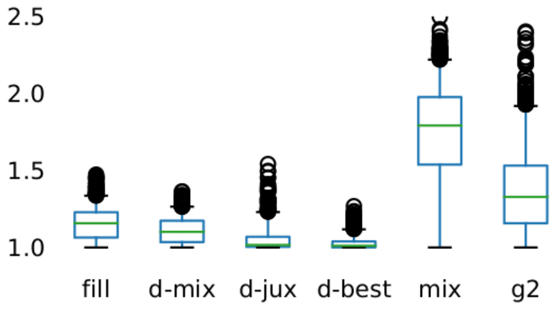


Figure 1.1: Task loads and types derived from the Google trace. Axes denote the CPU and memory usage from the dataset (normalized to the maximum usage). Four symbols (triangle, circle, square and inverted triangle) denote to which type a task belongs in instances with four types. Dashed lines denote boundaries for types.

Figure 1.1 shows the result obtained when we fix 4 types. Then, we set the coefficients $\alpha_{t,t'}$ based on how compatible the resource requirements are. After the conversion process, t_1 is a type for which memory dominates CPU, while the for the last type, t_T, CPU dominates memory. The further the type numbers are, the more compatible these types are, and we set coefficients accordingly (the coefficient $\alpha_{i,j}$ decreases when $|j - i|$ increases).



Performance of heuristics for $SumCT$.



Performance of heuristics for $MaxCT$.

Figure 1.2: Cost of the solutions returned by various heuristics normalized to the cost of an optimal solution (or a lower bound of such a cost). For SumCT, the cost of an optimal solution is computed using an optimal algorithm for this problem, whereas for MaxCT, we use a lower bound obtained using a quadratic program with binary variables which computes the optimal solution of a relaxed version of our problem (we allocate fractions of loads rather than individual, discrete tasks). We use boxplots, which are a method for graphically depicting groups of numerical data through their quartiles. In boxplots the middle line represents the median, the box spans between the 25th and the 75th percentile, the whiskers span between the 5th and the 95th percentile, and the circles show the remaining points (outliers).

For problem SumCT, the best heuristic we have starts by computing an optimal solution of the instance by considering the sizes of the tasks but not their type (this would be an optimal solution if all the tasks were of the same type). This can be obtained in polynomial time with the dynamic programming algorithm based on the OS property presented before. The resulting assignment is of course converted back to a multi-type solution. Then, starting from this solution, we perform a local search algorithm, where, for each type, the algorithm tries all possible movements of tasks between machines that maintain the OS order. If no improvement is possible, the algorithm stops. Otherwise, the algorithm accepts the assignment which has the lowest overall cost and, in the next iteration, tries to improve it. This algorithm, which converges quickly, outputs solutions close to the optimal one, and finds the optimal solution in 68% of the cases (it is the second heuristic in Figure 1.2 left [PR18]).

For problem MaxCT, we have also introduced fast heuristics. One of these heuristics is based on a partition of types into clusters of compatible types. It schedules the tasks of a same cluster greedily, by filling the machines until a threshold obtained by a dichotomic search on the makespan. This algorithm is the first on in Figure 1.2 right [PR19]). Another heuristic, which is a $\sqrt{2}$-approximate algorithm for a particular case of MaxCT, behaves well in practice (it is the fourth heuristic in Figure 1.2 right [PR19]).

1.4 A few words to conclude.

Optimizing problems with centralized algorithms is standard in computer science: I will not give further work on this subject, but rather on the model and problems SumCT and MaxCT seen through this chapter. The aim of the model presented in Section 1.2.2 is to capture tasks' heterogeneity while optimizing the performance of the tasks, and not just the usage of the resources (contrarily to the multi-dimensional bin packing models [165]). This model is a minimal extension of a standard scheduling model. It is a first step towards a model of assignment of tasks in data centers, and several things remains to be done.

First, more evolved heuristics could be designed. However, to analyze the quality of heuristics, it would be very useful to get better data: data on the tasks (usage of different resources, and size of the tasks), in order to determine the type of each task; and once the types have been determined, values of the coefficients between the types. This can be an experimental work, where the coefficients $\alpha_{t,t'}$ are estimated by monitoring the performance of the tasks as a function of their collocation and their sizes.

These experiments could not only give us data on the values of the coefficients, but also allow us to refine the model. As it is standard in scheduling, we have assumed that the function of performance is linear: if all the tasks have the same type, the cost of each task is proportional to the load of its machine; if the size of each task on a machine is doubled, the cost of each task is doubled. However, more complex cost functions are interesting from the systems perspective (e.g., webserver's response time as a function of load is convex [161, 31, 109]). We started to study a generalization of our cost model: we studied the problem which minimizes the sum of the costs of the tasks when all the tasks have the same type and the cost of a task is any convex function of the total load of the machine. This problem is strongly NP-hard (note that when the cost function is linear, the problem was easy), and we give an approximate algorithm [PR18]. This work with convex cost functions remain to be done when considering types. Note that a concave cost function is not realistic, since it would mean that the average cost of a task decreases when the number of tasks on the same machine increases.

More realistic variants of data center resource management problem, for example on-line tasks, can also be taken into account, similarly as they are considered in classic scheduling.

Lastly, note that the notion of type of a task can also be applied to generalize other problems, as for example load balancing games (with agents choosing on which machines their tasks will be scheduled, as in [169]). Similarly, types may be also used to reduce the complexity of coalition structure generation problems.

In the applications seen in this chapter, we have focused on the utilitarian social choice function (which optimizes efficiency), and the egalitarian social choice functions (which is a basic fairness criterion). More evolved social choice functions, using the L_p norm, an OWA aggregator, or other fairness criteria could be use. In the next chapter, we will look at situations in which the agents can have a possibility of action which impacts the solution obtained, and thus their own cost.

Chapter 2

Agents have power: strategic games

2.1 Introduction: algorithmic game theory

In this chapter, we consider situations where a solution of a combinatorial optimization problem is not built by a single entity, but is obtained through the combination of individual decisions made by the members of a group of agents. Every agent is influenced by the actions of the others, creating a strategic situation.

A natural formalization of the situation is done with the help of *game theory* [135]. A game is defined as follows. Let $\mathcal{N} = \{1, \ldots, n\}$ denote a group of n agents. Each agent $i \in \mathcal{N}$ has a non empty set of strategies (actions) A_i, and each agent i chooses a (pure) *strategy* $s_i \in S_i$[1].

A *state* of the game (or a *strategy profile*) is a vector $S = (s_1, s_2, \ldots, s_n)$ from $S_1 \times S_2 \times \cdots \times S_n$. Each agent has an objective function, called *utility function* and denoted by u_i when she wants to maximize this function, or called *cost* and denoted by c_i when she wants to minimize it. The utility or the cost of an agent depends on the state of the game S. Each agent is selfish and selects a strategy which optimizes her utility or cost.

Central to game theory is the definition of a plausible outcome of the game. Numerous *solution concepts* exist in the literature. The most popular one is probably the *Nash equilibrium* [130]. This is a situation where no agent can improve her utility or cost by changing her strategy (the strategies of the other agents remaining the same). It is therefore a situation in which the system is considered as stable. Formally, a Nash equilibrium is is a state $S = (s_1, \ldots, s_n)$ such that[2] :

$$\forall i \in \mathcal{N}, \forall s'_i \in S_i, c_i(S) \leq c_i(s_1, \ldots, s_{i-1}, s'_i, s_{i+1}, \ldots, s_n).$$

Since pure Nash equilibria do not always exist, a major challenge is to identify which classes of games are guaranteed to always admit a pure Nash equilibrium. A good example is the class of *congestion games* which was shown to contain a pure Nash equilibrium in any case [147]. When a system designer has the power to set the "rules of the game", it is desirable to design systems (games) such that the agents behave (i.e. choose strategies) in a way which always leads to a Nash equilibrium (in Section 2.3 our aim will be to design systems which lead not only to Nash equilibria, but to Nash equilibria which are good for a global objective function expressing the social welfare). The following paragraph presents coordination mechanisms, which concern systems where independent and selfish agents use shared machines.

Coordination mechanisms.

Coordination mechanisms have been introduced by Christodoulou, Koutsoupias and Nanavati [47, 48]. They are used when a set of independent and selfish agents have each one a task to

[1]In this chapter, we do not consider mixed strategies, which are for each agent a distribution of probabilities on A_i. We will therefore omit the term pure when we will speak of the (pure) strategy of an agent.

[2]Since we focus on pure strategies, we do not consider in this report mixed Nash equilibria, which are Nash equilibrium where agents can have mixed strategies (it is known that a mixed Nash equilibrium always exists in a finite game [130]). We will mainly consider in this manuscript situations where agents have costs. If each agent has a utility, the definition is symmetric: a (pure) Nash equilibrium is a situation such that $\forall i \in \mathcal{N}, \forall s'_i \in S_i, u_i(S) \geq u_i(s_1, \ldots, s_{i-1}, s'_i, s_{i+1}, \ldots, s_n)$.

schedule on a set of $m \geq 2$ shared machines, and when each agent chooses on which machine she will assign her task. Each machine i has a public *policy*, which is an algorithm which returns a schedule of the tasks assigned to i. This policy may introduce idle times between the tasks. However, the system is decentralized and the policy of machine i depends only on the tasks assigned to it: it cannot be a function of the tasks assigned to the other machines. A set of policies, one for each machine, is called a *coordination mechanism.* The main scheduling rules studied are SPT (resp. LPT) which schedules the tasks assigned to a machine in increasing (resp. decreasing) order of their lengths. The cost that each agent wishes to minimize is usually the completion time of her task in the schedule obtained [48, 102, 113, 18, 54, 33, 55, 59, 99][ABP06b, CGP07]. Usually, the authors first show that the designed coordination mechanisms induce pure Nash equilibria, and then they study its performance in term of price of anarchy (see Section 2.3). We will see in the next section how we can characterize the existence of pure Nash equilibria.

2.2 Dynamics and characterization of the existence of stable solutions

Let us first look at the dynamics of a game, which is also important. The *dynamics* of a game is the way the agents successively modify their strategy. In game theory, a best response of agent $i \in N$ is a strategy which minimizes the cost (or maximizes the utility) of agent i, taking other agents' strategies as given: strategy $x \in S_i$ is a *best response* of agent $i \in N$ with respect to the state $s \in \mathcal{S}$ if $\forall s'_i \in S_i, c_i(s_1, \dots, s_{i-1}, x, s_{i+1}, \dots, s_n) \leq c_i(s_1, \dots, s_{i-1}, s'_i, s_{i+1}, \dots, s_n)$[3]. Therefore, a Nash equilibrium is a state in which each agent in a game has selected a best response to the other agents' strategies.

A *Best Response Dynamics* (*BRD* in short) is a sequence of states $s^0, s^1, \dots,$ in $\mathcal{S}$ such that each state s^k (except s^0) results from a profitable deviation (best response) of some agent from state s^{k-1}. Note that if a BRD reaches a Nash equilibrium after a finite number of states, then no further change of strategies is expected. Do the players converge to an equilibrium? Is it possible from any state? Do we have convergence after a small number of deviations? These questions are of major importance in algorithmic game theory [136, 73, 72].

A way to show that a game always admits a Nash equilibrium (and that a BRD leads to a Nash equilibrium) is to show that a game is a potential game [126]. This can be shown by exhibiting a potential function, following a concept introduced by Monderer and Shapley [126]. A *potential function* Φ takes as input a state S of a game and a state S' where one agent i changes her strategy to a best response (the strategies of the other agents remaining the same)[4]. The difference $\Phi(S) - \Phi(S')$ is linked to the decrease of cost of agent i (for example, for exact potential games, $\Phi(S) - \Phi(S') = c_i(S) - c_i(S')$). Since the number of possible states is finite, potential games always admit at least one Nash equilibrium (which can be obtained with a BRD from any state). For example, any congestion game is a potential game [148].

Another way to show that a Nash equilibrium exists consists in giving an algorithm which outputs Nash equilibria (usually these Nash equilibria can also be obtained by a BRD, but in a longer time). In Section 2.3, we will give such an algorithm. When a game does not always admit a Nash equilibrium, it is sufficient to show an instance of the game where no Nash equilibrium exists, as we will do it in the section 2.2.2. It is also sometimes possible to characterize the existence of a Nash equilibrium, as shown in the following section.

2.2.1 Example: investment on sites by two mobiles operators

As part of the PhD thesis of Alexandre Blogowski, co-funded by Orange Labs and the French State (through a "thèse CIFRE"), and co-supervised by Philippe Chrétienne, Adam Ouorou, and myself, we have been interested in the sharing of a radio access network infrastructure by two mobile operators. The setting is the following one: two mobile operators A and B share a set of sites (which are possible places for base stations used by the mobile operators). Each operator $i \in \{A, B\}$ has, for each site j, the choice not to invest (in this case it has no profit), or to invest.

[3]Likewise, if each agent has a utility, a strategy $x \in S_i$ is a best response of agent $i \in N$ with respect to the state $s \in \mathcal{S}$ if $\forall s'_i \in S_i, u_i(s_1, \dots, s_{i-1}, x, s_{i+1}, \dots, s_n) \geq u_i(s_1, \dots, s_{i-1}, s'_i, s_{i+1}, \dots, s_n)$.

[4]This also works if S' is a state in one agent changes her strategy to a better response (which is not necessarily her best response).

In the latter case, it will have a profit s_i^j if the other operator does not invest on site i, or a profit t_i^j if the other operator invests on site i (note that profits can be positive or negative – this last case occurs when the investment on a site is higher than the expected gain). The aim of each operator (agent) is to maximize its total profit, which is the sum of the profits it gains on all the sites. Alexandre Blogowski, Philippe Chrétienne and I gave a necessary and sufficient condition on the values s_i^j and t_i^j so that a Nash equilibrium exists [BCP15]. In the following chapter, we well go back to this problem and see how we could return an assignment of operators to sites which should be accepted by both operators.

In the next section, we will introduce a new game and two models, and show that in the first model there does not not always exist a stable solution, while in the second model, a pure Nash equilibrium is always guaranteed.

2.2.2 Example: scheduling tasks from selfish multi-tasks agents

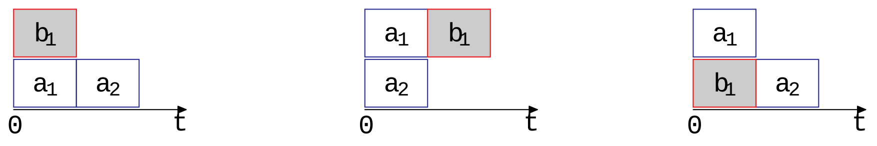
In this section, we will focus on a problem that Johanne Cohen and I have introduced [CP15]. We consider a set of n selfish agents $\{1, \ldots, n\}$, each agent i owning a set of n_i tasks that she wants to schedule on a set of shared machines. Each task has a unique identification number (ID) and an arbitrary processing time (length). Knowing the policies of the machines, each agent chooses, for each of her tasks, on which machine it will be scheduled. The *strategy* of each agent is thus an assignment to a machine of each of her tasks. The aim of each agent is to minimize the average completion time of her tasks. This is equivalent to minimize the sum of completion times of her tasks: in the sequel the cost of each agent is thus the sum of the completion times of her tasks.

We consider two models. In the first one, the machines cannot distinguish the tasks of one agent from the tasks of another agent: a machine is only aware of the length and identification number of the tasks it has to schedule. In the second model, the machines know the owner of each task.

The aim is to design a coordination mechanism when an agent can have more than one task to schedule. Therefore, this generalizes the notion of coordination mechanism, introduced when each agent has a single task to schedule[5].

We first give a condition on the policies of the machines which is necessary to guarantee that a Nash equilibrium always exists. This condition holds when the machines are not able to determine the owners of the tasks (and their policies are thus only based on the lengths and the IDs of the tasks).

Theorem 2.1 *([CP15]) If all the machines have the same deterministic policy, and if this policy does not introduce idle times between the tasks, then the coordination mechanism does not always induce a pure Nash equilibrium.*



(a) $C_A = 3$ and $C_B = 1$ (b) $C_A = 2$ and $C_B = 2$ (c) $C_A = 3$ and $C_B = 1$

Figure 2.1: Instance with no pure Nash equilibrium when all the machines have the same deterministic policy without idle times. C_A (resp. C_B) is the cost of agent A (resp. B).

To prove this, we simply provide a instance without pure Nash equilibrium. This instance, depicted in Figure 2.1, consists in two machines and two agents A and B: Agent A has two tasks a_1 and a_2, each of length 1, while Agent B has one task b_1 of length 1. Given two tasks

[5]Abed et al. [1] also consider coordination mechanism (on unrelated machines) when agents own several tasks, with weights. However, they do not consider that the coordination mechanism should induce Nash equilibria, but a superclass of Nash equilibria: they consider that a schedule is stable (they call such a schedule a *weak Nash equilibrium*) if no agent can decrease her cost by moving *exactly one* of her task to a different machine. They show that when the policies of the machines order the tasks according to their length to weight ratio, then there exists a weak Nash equilibrium.

i and j of length 1, we note $i \prec j$ if a machine schedules i before j. We consider three tasks a_1, a_2, and b_1 such that $a_1 \prec b_1$ and $b_1 \prec a_2$. Note that given three tasks i, j, k, and any deterministic policy, there always exists a permutation of the tasks such that $i \prec j$ and $j \prec k$. The configuration which consists of three tasks on the same machine is not a Nash equilibrium since b_1 would have incentive to move on the idle machine. The other configurations are represented in Figure 2.1 and are also not Nash equilibria: in Figure 2.1(a) Agent A can decrease her cost by assigning task a_1 to the first machine; in Figure 2.1(b) Agent B has incentive to move her task; in Figure 2.1(c) Agent A has incentive to exchange the assignment of her two tasks a_1 and a_2.

Note that the classical policies LPT and SPT have this property, and thus they do not always induce pure Nash equilibria (contrary to the case where each agent has only one task [102]). Moreover, the move of only two tasks is needed to show this result (we know that when multi-tasks agents are able to move only one task to improve their cost, then the SPT policy is stable [1]: for each instance there exists a schedule where the agents cannot improve their costs by moving at most one of their tasks). Knowing whether there is a coordination mechanism which induces Nash equilibria without introducing idle times and/or having different policies for the machines, is an open problem.

If the machines are able to know the owner of the tasks (for example owners have identification numbers, and their ID is reported on each of their tasks), then there always exists a Nash equilibrium [CP15], as for example when all the machines schedule the tasks by increasing order of the ID of their agent. This algorithm is unfair and may lead to Nash equilibria with very bad social cost, but we can improve it: assuming that the number of agents, K, is smaller than or equal to the number of machines, m, then, for each agent i, there are $\lfloor m/K \rfloor$ or $\lfloor m/K \rfloor + 1$ machines on which the tasks of Agent i are scheduled first (from the smallest one to the largest one). On these machines, once the tasks of i have been scheduled, the tasks of the following agent $(1 + i \mod K)$ are scheduled, from the smallest one to the largest one, and then the tasks of the following agent $(1 + (i + 1) \mod K)$, and so forth. This converges quickly to a Nash equilibrium, which furthermore has a bounded deterioration of the sum of the costs of the agents, compared to an optimal solution, in which the agents would not have the choice of the machines [CP15]. Obtaining equilibria good for the whole set of agents is indeed a desirable quality of a system.

We will now focus on games which always admit Nash equilibria, and, assuming that the agents naturally converges towards a Nash equilibrium, we will analyze the quality of the obtained solutions, for a social choice function that we wish to optimize.

2.3 Measure of the system's efficiency

The quality of an equilibrium is usually given by the sum of the costs of the agents (or the sum of their profits), or by the maximum cost (or minimum profit) of an agent, that we wish to minimize (resp. maximize). These are the utilitarian and egalitarian social choice functions which are also used in centralized settings, and that we have already seen in the previous chapter. The value of the social choice function in a solution is usually called the *social cost* of the solution. In this section, we will focus on games that always admit Nash equilibria, and measure the quality (in terms of social cost) of the Nash equilibria of these games. This is usually done using the price of anarchy, introduced by Koutsoupias and Papadimitriou [115].

Price of anarchy.
When the agents minimize a cost function, the *price of anarchy* is the maximum ratio (over all the instances) between the social cost in the *worst* possible Nash equilibrium and the optimal social cost[6]. The price of anarchy measures how far a Nash equilibrium can be from an optimal solution for the social cost: a large price of anarchy means that the selfish behavior of the agents and the lack of coordination between the agents, have a strong negative impact on the social cost, while a price of anarchy close to 1 means that the social cost of Nash equilibria is good though the agents do not optimize it. The probably most famous example of this situation has been given for routing games, which model an important practical problem: how to route traffic in a large

[6]Similarly, when the agents maximize a utility function, the price of anarchy is the minimum ratio (over all the instances) between the social cost in the *worst* possible Nash equilibrium and the optimal social cost.

network without central authority (this is the case of roads network, but also of the Internet)[7]. Roughgarden and Tardos [151] have shown that the price of anarchy of a routing game where the travel time on an arc is an affine function of the traffic of this arc, is at most 4/3: despite the absence of central authority, the average travel time of a unit of traffic is at most 4/3 the average travel time of a unit of traffic in an optimal solution. Many games, however, admit much larger price of anarchy.

In systems used by selfish agents, the designer of the system – the person deciding the policies of the machines in coordination mechanisms, for example – wishes to design a system which induces Nash equilibria, and Nash equilibria with a price of anarchy as good as possible. For example, if the social cost of a coordination mechanism is the makespan, that we wish to minimize, and if each agent has one task to schedule, it is better to set the policies of the (identical) machines as LPT rather than SPT. Indeed, the price of anarchy will be $\frac{4}{3} - \frac{1}{3m}$ in the first case, whereas it will be $2 - \frac{1}{m}$ in the second case [47].

Recall that for NP-hard problems, we aim to design approximation algorithms with ratios as close to 1 as possible, and, when there is no PTAS, we wish to give inapproximability bounds such that, for a given value $\alpha > 1$, we know that there does not exist α-approximate polynomial time algorithm if $P \neq NP$. For games, we have the same approach: we wish to design systems which induce games with a good price of anarchy, and, at the same time, we wish to exhibit bounds α such the price of anarchy cannot be below a certain α (or over a value α when the agents maximizes a utility function). For example, for the multi tasks agents coordination mechanism problem introduced in the previous section, we have [CP15]:

Theorem 2.2 *([CP15]) Consider a coordination mechanism in which all the machines have the same deterministic policy which depends only on the lengths of the tasks. If this coordination mechanism always induces a pure Nash equilibrium, then its price of anarchy is larger than or equal to* 2.

The proof of this properties lies on case analysis, taking the instance depicted in Figure 2.1 and analyzing the idle times that have to be before each task in order to get a Nash equilibrium.

With Georges Christodoulou and Laurent Gourves [CGP07], we also studied the price of anarchy in the context of coordination mechanisms when agents have private data (we will see this more precisely these kind of situations in Chapter 4). We gave lower bounds on the best price of anarchy that a coordination mechanism which induces Nash equilibria can have when each agent chooses on which machine will be executed her (single) task, and when each agent is the only one to know the true length of her task.

In the same way as the approximation ratio of an algorithm measures the loss of performance due to the fact that we wish to use a polynomial time algorithm, the price of anarchy bounds the deterioration of the social cost due to the fact that the agents act selfishly and without coordination. Note however that a (deterministic) approximation algorithm always output the same solution. In contrast, in games, there can be several Nash equilibria, and the price of anarchy is a "pessimistic" vision since we focus on worst quality of a Nash equilibrium.

Price of stability.
An optimistic vision of the loss of performance due to the fact that the agents act selfishly is given by the *price of stability* [153, 10], which is the maximum ratio (over all the instances) between the social cost in the *best* possible Nash equilibrium and the optimal social cost[8].

We can imagine that, given the data of the agents, a trusted third-party suggests to each agent a strategy. This solution suggested has to be a Nash equilibrium, otherwise some agent would have incentive to change strategy; however the entity can of course output the best Nash equilibria.

[7]A (nonatomic) routing game [150] is a special class of congestion games. We are given a graph $G = (V, E)$ with a defined set of sources $s_i \in V$ and sinks $t_i \in V$. Each source-sink pair corresponds to an amount of traffic (also called a commodity) r_i that must travel along the arcs of G from s_i to t_i. Each arc e of G is associated to a positive, continuous and increasing cost function c_e. The cost of each commodity (agent) is the sum of the costs incurred by each unit of traffic of this commodity – and each unit of traffic incurs a cost equal to $c_e(f_e)$ on each arc e that it takes, and which is taken by f_e units of traffic (i.e. $c_e(f_e)$ represents the travel time of arc e when f_e units of traffic takes it). The social cost is the sum of the costs of the commodities.

[8]This holds when the agents minimize a cost function. Similarly, when the agents maximize a utility function, the price of stability is the minimum ratio (over all the instances) between the social cost associated with the *best* possible Nash equilibrium and the optimal social cost.

Hence, the price of stability, as its name suggests, measures the loss of performance due to the fact that we are looking for a stable solution.

In a game with a bad price of anarchy but a price of stability close to 1, it is worth trying to make the agents coordinate, for example via a third entity, in order to obtain a good Nash equilibrium. In the next chapter, we will see how a third entity can propose good solutions (Nash equilibria) for some scheduling and assignment games [DPRT11, GMP12].

On the contrary, a game with a bad price of stability show that the impact of the selfishness of the agents is high. Fortunately, there is often possibilities to get better solutions, by trying to "modify" the game.

Ways to improve the system's efficiency.
In routing games for example, adding or removing arcs to the graph (adding or removing roads), adding delays or costs to the arcs (slowing down the traffic, or adding tolls to roads), can reduce the social cost of a game[9] [67]. Likewise, to circumvent the negative result of Theorem 2.2, we can think about a coordination mechanism in which all the machines do not have the same policy, a coordination mechanism with randomized policies, or a coordination mechanism which has more information, such as for example the agent owner of each task. More generally, the following tools can be used:

- *Modifying the costs of the game.* This can be done by modifying the game: slowing down the travel time on a certain road, or increasing the travel time on another road, in a routing game, for example [67]. In coordination mechanisms, it is even easier to modify the game since the policies define the costs of the tasks. This kind of situations is sometimes call "reverse game theory": the aim is to fix the rules of the game (the policies of the machines) in order to incite tasks to converge towards desirable Nash equilibria. Paying agents (when their cost/utility can be expressed as a monetary cost) is also a popular way to do. This is what is done in routing games by putting tolls on arcs. Note that paying the agents is a tool that is often used in the context of truthful mechanisms (see Chapter 4).

- *Forcing some agents to adopt a given strategy (Stackelberg games)* [149]. In some settings, the system designer can force some agents to have a desired strategy. The other agents choose their strategies by themselves. The aim is to force a number of agents as low as possible to obtain an equilibrium, called a Stackelberg equilibrium, which is as good as possible. Some work consider the minimal number of agents to force in order to obtain an optimal solution (such a number is called the *price of optimum*) [107].

- *Modifying the set of strategies of the agents.* The extreme case is the one of Stackelberg games, in which some agents are forbidden to take another strategy that the one imposed. Usually, we can modify the strategies without forcing any agent, but by modifying the game. This can for example be done by adding or removing new arcs (roads) in a routing game, or by putting different policies on the machines in a coordination mechanism (some machines executing only a subset of tasks for example).

- *Adding information to the game.* This is for example the case when in the multi-task coordination mechanism problem, the machines are able to determine the owners of the tasks. We can also envision games where information is exchanged between the resources so that they can adapt their cost functions, in order to induce better Nash equilibria.

2.4 Other approaches

Note that the social cost is usually the utilitarian or the egalitarian social choice function. It would be interesting to study some other criteria. For example, an OWA aggregator or other fairness criteria could be used (the minimization of the maximum cost being a first step to output a "fair" Nash equilibrium). Social cost not associated to the users but associated to the system designer could also be considered. For example, the aim could be to minimize the energy of the system

[9]More precisely, they change the game, and may transform a game with a high social game into a game with a better social cost.

(machines for example, in a scheduling problem), when the system is used by selfish agents (whose aim is not necessary to reduce their energy impact).

In this chapter, as it is the case in most of the works, we considered that a stable situation is a Nash equilibrium (or an α-approximate Nash equilibrium, with α close to 1). Stronger notions of equilibrium exist, and have been introduced in order to take into account the possibility for a group of agents to collectively choose a set of strategies which is better of for each of them. Robert Aumann [17] defined a *strong equilibrium* as a state where no group of agents (coalition) can deviate in a way that benefits each of the deviators. Hence strong equilibria are a subset of Nash equilibria. For a game, it can be worth looking whether a strong equilibrium always exists, and if it is the case, to study the quality of theses equilibria. This can be done via the strong price of anarchy [5] (or the strong price of stability), defined similarly as the price of anarchy (stability) but on the set of strong equilibria.

We have seen that, when the system is used by selfish agents which have a power (choice between different strategies), we can guarantee a loss of performance at most equal to the price of stability if a central entity proposes a solution (strategy profile) to the agents. A central entity could also propose a solution in which each agent decreases her cost by a factor at most $\alpha = (1+\varepsilon)$ by changing strategy – such a solution is called an α-approximate Nash equilibrium[10]. Indeed, if each agent knows that the central entity will return an $(1+\varepsilon)$-approximate Nash equilibrium, then, if ε is small, we can assume that she will not look whether there exists a strategy better that the one proposed by the entity (it is not worth trying to find a better strategy for such a small gain). It is then interesting to look at the tradeoff between ε and the quality of the solutions obtained: we get a curve giving for each value of ε the best approximation ratio of an algorithm returning $(1+\varepsilon)$-approximate algorithm. Therefore, for a fixed acceptable ε we could know the performance we could expect from the system. Following the definition of the price of stability, the price of approximate stability [44][ABP06a] is the maximum ratio, over all the instances, of the social cost in the best α-approximate Nash equilibrium, over the optimal social cost. Note that the use of α-approximate Nash equilibrium could also lead to another kind of "price of optimality", looking at the smallest ε needed to output an optimal solution. The price of approximate stability has been studied for coordination mechanisms [ABP06a, Pas06], for traffic routing and congestion games [160, 35, 93, 36, 45] and other problems [65][11]. Note that the same approach could also be used for strong equilibria (this would lead to the price of approximate stability for strong equilibria).

In the next chapter, we will follow this idea to make the agents cooperate in order to obtain solutions which are, as far as possible, good for the social cost.

[10]This corresponds to what is done for traffic advices given at the radio by the national road information center "Bison Futé" – the paths proposed are in some cases not the shortest paths, so that a (good) approximate Nash equilibrium is obtained.

[11]Of course, the same approach can be used when the agents have utilities instead of costs. In this case, an α-approximate Nash equilibrium is a state in which no agent can increase her utility by a factor larger than α.

Chapter 3

Agents cooperate thanks to a trusted third-party

As in the previous chapters, we consider a set of agents, each agent having her own interest. As in the previous chapter, we consider that the agents have a possibility of action (a set of strategies), and that each agent will act (choose a strategy) in a way that maximizes her utility. Contrarily to what we have done in the previous chapter, we will consider that there is a central entity that can be considered as a trusted third-party. The aim of this entity will be propose a solution (i.e. a strategy for each agent) which optimizes a social choice function, while ensuring that the agents will accept the solution proposed.

Let us first note that, as noted in the previous chapter, a way to make agents to cooperate through this central entity is to make the entity propose the best (or one of the best) Nash equilibrium (with respect to the considered social choice function). Such a solution optimizes the social choice function under the constraint that no agent has incentive to deviate from the strategy which is suggested by the central entity. The best approximation ratio that such an algorithm can have corresponds thus to the price of stability of the game.

In practice, numerous work study the price of stability of several games, but few work aim at returning the best – or a good – Nash equilibrium. One of the most famous examples which shows the interest of the price of stability (vs the price of anarchy) is the network formation game[1]. The price of anarchy of this game for the utilitarian social choice function is the number of agents, k, whereas the price of stability is H_k, the k-th harmonic number [10]. This game is a famous example showing the interest of the price of stability in part because its price of stability is much lower than its price of anarchy. However, despite the proof given to show the price of anarchy is constructive, it does not lead to a polynomial time algorithm which computes a H_k-approximate solution. Indeed, the constructive proof starts by computing an optimal solution of the problem which consists in minimizing the total cost of the network, and this problem is NP-hard (it is the generalized Steiner tree problem). Then, starting from this optimal solution, it applies a best response dynamics, which converges towards a H_k-approximate Nash equilibrium.

Like for this game, it is possible to bound the price of stability of potential games using this method, called the potential function method [167]: we can have a guarantee on the quality of a Nash equilibrium obtained with a best response dynamics whose initial state is an optimal solution for the considered social choice function (for this class of games the best response dynamics is guaranteed to converge). With this kind of proofs, there is no guarantee that the (hopefully) good Nash equilibrium can be obtained in polynomial time for two reasons: first, because the initial solution cannot always be obtained in polynomial time, and second, because it is usually

[1]The *network formation game* is the following game: we are given a directed graph $G = (V, E)$ with nonnegative edge costs c_e for all edges e. There are k agents, and each agent has a specified source node s_i and sink node t_i. Agent i's goal is to build a network in which t_i is reachable from s_i, while paying as little as possible to do so. A strategy for agent i is a path P_i from s_i to t_i in G. Given a strategy for each agent, we define the constructed network to be $\cup P_i$. The mechanism splits the cost of an edge evenly among all agents whose path contains it: if k_e denotes the number of agents whose path contains edge e, then each agent using e has to pay c_e/k_e for this edge. The total cost incurred by agent i is given by $cost(i) = \Sigma_{e \in P_i} c_e/k_e$. The social choice function is the utilitarian social choice function, which consists in minimizing the sum of the cost of the agents, and thus the sum of the edges selected.

not guaranted that the best response dynamics converges in a polynomial time.

In this chapter, our aim will be to obtain *polynomial time* algorithms which will lead to good and stable solutions. We will follow two approaches:

- In the first approach, our aim is to optimize a social choice function under the constraint that the solution returned is a Nash equilibrium. In particular, we will focus on problems where agents can either cooperate (for example by accepting the solution returned by a trusted third party), or they can refuse to cooperate, and compute a solution on their own. We will in thus design an algorithm, used by a trusted third party, that computes a good solution (for the social choice function) that all the agents have incentive to accept: they will not have a better utility by computing a solution on their own. Note that this corresponds to the *rationality constraint* in contract theory, which, in economics, studies how economic actors can construct contractual arrangements. The rationality constraint insures the incentive that an individual economic agent has to participate in a given game: an agent will not participate to a game if she has a cost (resp. a utility) larger (resp. smaller) than if she does not participate[2]. In other words, our aim is to make the agents cooperate in order to optimize a social choice function, while each agent does not increase her cost (or does not decrease her utility) compared to the situation where she does not cooperate. We will see two situations: in the first one (section 3.1), agents own resources that they can share (for a scheduling and an assignment problem), and in section 3.2, we will be interested in situations in which non symmetric agents interact (and we will illustrate this on a lot sizing problem).

- We will introduce a second approach, in section 3.3, for a problem where agents build new resources, alone or together. The aim will be there to design an algorithm which, for each instance, returns a solution in which each agent has a utility at least equal to the best utility she could have in a Nash equilibrium (if at least a Nash equilibrium exists for the instance).

Let us first look on problems in which agents own resources that they can share, and see how making the agents cooperate, via a trusted third party, can improve the quality of the solution obtained with respect to the (utilitarian or egalitarian) social choice function.

3.1 Agents own resources that they can share

In the last chapter, we have seen situations where agents compete for the same shared resources, trying to choose strategies (usually choosing which resources to use) in order to optimize their objective function. This is, for example, the case for congestion games (and thus routing games [150]), for coordination mechanisms [48], and for load balancing games [170]. In this section, we will assume that each agent has a set of resources (her resources), and a set of tasks that need resources. Each agent can assign her tasks on her own resources, or cooperate by sharing tasks and resources. The aim is to optimize a social choice function under the constraint that each agent has a cost at most equal to the cost she would have by executing her tasks on her resources (this constraint is the *rationality constraint*: an agent which would decrease her cost would not accept the proposed solution and would prefer to keep here tasks on her resources).

3.1.1 The multi-organization scheduling problem.

We will focus here on the multi-organization scheduling problem (MOSP), introduced by Krzysztof Rzadca, Denis Trystram and I [PRT07, PRT09], and followed by several works [DPRT11][133, 51, 53, 52, 58, 134, 37, 50]. This problem takes as input a set of n organizations $O_1, \ldots, O_n$ (think that an organization is an administrative entity grouping users and computational resources such as private companies or academic research groups). Each organization O_i has a cluster M_i of m_i machines, and a set of tasks (called its local tasks). These tasks are parallel rigid tasks: each task j has a length (processing time) p_j and a number of machines q_j on which it has to be executed (i.e. task j has to be executed in parallel on q_j machines during p_j time units).

If each organization O_i schedules its tasks on its cluster M_i it obtains a (local) makespan $C^{loc}_{\max}(O_i)$. We aim at scheduling all the tasks on all the machines, and not only the tasks of

[2]Note that this type of constraint is also called *voluntary participation* in algorithmic game theory

an organization on her machines. Each organization wants to minimize its makespan, where the makespan of organization O_i, denoted by $C_{\max}(O_i)$, is the maximum completion time of a task of O_i. The global makespan is $C_{max} = \max_{i \in \{1,\dots,n\}} C_{\max}(O_i)$: it is the maximum cost of an agent. By sharing machines, organizations may decrease this global makespan.

The multi-organization scheduling problem (MOSP) consists in computing a schedule of all the tasks on all the machines such that the global makespan, C_{max}, is minimized, under the constraint that no organization increases its makespan[3]*. This problem is therefore an extension of the classical parallel tasks scheduling problem by the constraints stating that no organization's makespan can be worsened.*

Note that the rationality constraints do not allow to always obtain the makespan we could have obtained without these constraints, as shown in Figure 3.1. With the instance depicted in this figure, we see that the makespan of an optimal solution of MOSP can be $\frac{3}{2}$ higher than the optimal makespan without the rationality constraints (with a more complicated instance, we can see that this bound is in fact at least 2).

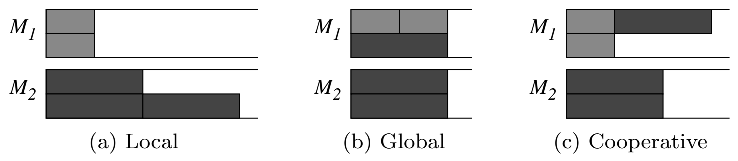


Figure 3.1: There are two organizations, each having a cluster of 2 machines. O_1 has light grey tasks, while O_2 has dark grey tasks. Globally-optimal solution (b) is inadmissible, as it extends the makespan of organization O_1 in comparison with the local solution (a). The best solution which does not increase any local makespan (c) has a makespan equal to $\frac{3}{2}$ the optimal makespan.

The previous instance shows the impact of the rationality constraint on the global makespan. Note on the contrary that cooperation can decrease by an arbitrary factor the cost of an organization, if this organization is very loaded compared to the other organizations. On a positive side, the following result also shows that the rationality constraints do not induce an arbitrary increase of the makespan compared to an optimal solution without these constraints.

Theorem 3.1 *[DPRT11] There exists an algorithm which computes a solution to MOSP in which the makespan is at most 3 times the optimal makespan of the same instance without the rationality constraints*[4]*.*

Note that the optimal makespan with the whole set of machines and tasks but without the rationality constraints is a lower bound of the optimal makespan of our problem: our algorithm is thus 3 approximate. Our algorithm starts by scheduling all the tasks on the cluster of their organization in a greedy way, in decreasing order of their height (q_i). Then it computes a lower bound of the makespan, LB, it "cut" the schedule at time $3LB$, and it unschedules the tasks that are completed after $3LB$. These tasks are then rescheduled before $3LB$. Figure 3.2 right shows the schedule returned by this algorithm when the local schedules are the one of Figure 3.2 left. The 3-approximation proof consists in showing, by surface arguments, that all the unscheduled tasks fit before $3LB$, and that the resulting schedule is thus valid.

It is an open question whether there exists a polynomial time algorithm returning a schedule whose makespan is between 2 and 3 the optimal makespan without the rationality constraints.

Note that we considered here parallel tasks, but this problem is also of interest for sequential tasks (i.e. for each task i, $q_i = 1$: each task needs only one machine to be executed), and some works focus on this setting [53, 133, 52, 50]. Other extensions have been studied, such as MOSP on unrelated machines [133], the case where we add as a constraint that no organization should

[3]In other word, in the solution returned, we must have: $\forall k \in \{1,\dots,n\}$ $C_{\max}(O_k) \leq C^{loc}_{\max}(O_k)$. These constraints are the *rationality constraints*.

[4]We consider that the local schedules are obtained by scheduling the local tasks in decreasing order of their height in each cluster – this constraint can be relaxed to any local schedule [DPRT11].

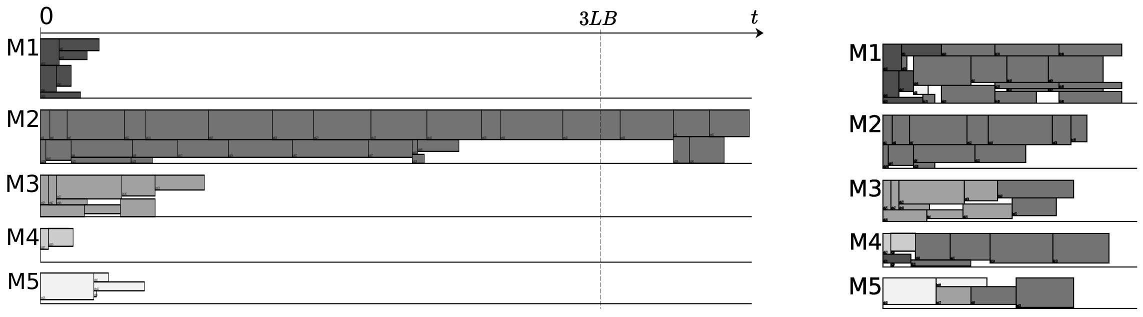


Figure 3.2: Left: local schedules. Right: schedule computed by the algorithm. It improves both the global makespan and local makespans of O_1, O_2, O_3 and O_4.

be able to execute earlier one of its task on its own machines [53], or problem α-MOSP [133, 37], where the aim is to return a schedule in which each organization increases its makespan of a factor at most α. Another work consider as a global objective to return a schedule which minimizes the total energy consumed while not increasing the energy consumed by an organization [50]. Many further works are possible.

Future work.
First of all, it would be interesting to study more varied objective functions. One organization could aim at minimizing the average completion time of its task, while another one wishes to minimizes its makespan, and a third one the sum of the tardiness of its tasks. Sharing machines is even more interesting when all the organizations do not have the same goal, and is particularly interesting when the tasks have different deadlines.

To be closer to the reality, taking into account tasks which arrive over time (online) is an open issue. Many works also remain to be done on problem α-MOSP (in particular, when the machines are identical, the bounds obtained in the unrelated case [133] could be improved).

Another interesting direction concerns fairness of the solution returned. A solution to MOSP guarantees only that no organization increases its makespan (or its local objective function if we consider that an organization can have any objective function). However, in an optimal solution for MOSP, it is possible that an organization decreases a lot its cost, while another one does not decrease at all its cost. Returning a solution which maximizes the smallest gain (decrease of cost) of an organization could be an interesting direction. Returning a schedule in which no organization O_i is "jealous" from another one O_j (for example because they are similar – same number of tasks and machines – but O_j decreases its makespan more than O_i does), would also be an interesting direction. More generally, taking fairness notions from social choice, and trying to return schedules as fair as possible in function of these notions, is a promising direction. This could lead to a list of fairness axioms (Pareto optimality, envy-freeness, proportionality, etc)[5], some axioms being incompatible. For example, there are instances where it is not possible to have a schedule which is at the same time Pareto optimal[6] and envy-free (where envy-freeness means that no organization has more tasks scheduled on non local machines than another organization which has not a higher cost – other definitions of envy-freeness are possible). Fairness axioms are often used in situations where agents compete for resources (but do not own resources). However, problems where agents own tasks and resources, and share resources, could also lead to interesting problems in the fair division area.

3.1.2 Other problems.

We presented in the previous section the multi-organization scheduling problem, but the idea of sharing resources can be applied to any problem where independent agents have resources, as for example the multi-organization assignment problem, that Laurent Gourvès, Jérôme Monnot and I have studied [GMP08, GMP12].

[5]Some of these axioms will be seen in Chapter 5.

[6]a schedule S is Pareto optimal if there does not exists another schedule S' where the cost of each organization in S is smaller than or equal to her cost in S' – and strictly smaller for at least one organization.

The multi-organization assignment problem (MOAP).
Consider a set of organizations, each organization having its own pool of clients who either supply or demand one indivisible product. Knowing the profit induced by each buyer/seller pair, an organization's task is to conduct such transactions within its database of clients in order to maximize the amount of the transactions. Inter-organizations transactions are allowed: in this situation, two clients from distinct organizations can trade and their organizations share the induced profit (a fraction p is given to the buyer's organization, while the other fraction, $1-p$, is for the seller's organization). Since maximizing the overall profit leads to unacceptable situations where an organization can be penalized, we study the problem of maximizing the overall profit such that no organization gets less than it can obtain on its own. We show that MOAP is strongly NP-hard but polynomially solvable if the graph is not weighted. In the general case, we give a p-approximate algorithm and a matching inapproximation bound [GMP08, GMP12].

Future work.
As the multi-organization scheduling problem and the multi-organization assignment problem extend classical scheduling and assignment problems, we can envision multi-organization version of numerous classical combinatorial optimization problems. We could for example think at a "multi-organization routing problem", where an organization owns roads (arcs) between nodes, and clients (which want to go from one source to a destination), and when, by sharing their arcs the organization decrease the average travel time of their clients. Actually, any problem with resources and tasks or clients using these resources seems to be a candidate to a multi-organization version of the problem (and an approximate multi-organization version of the problem, if we assume that each organization accepts to increase its cost of a factor α, in order to get a better overall solution).

Let us now mention, in the following section, a cooperation situation where agents are not symmetric.

3.2 Agents interact with asymmetric relations: example of the 2ULS problem.

In this section, we will focus on the two level lot sizing problem, 2ULS, mentioned in Chapter 1, and that we recall now. In this problem, the supply chain is composed of two levels: one supplier and one retailer. The supplier produces and sends a product to a retailer, which transforms this product in order to satisfy demands over a finite planning horizon (there is a time horizon $\{1,\ldots,T\}$, and a demand d_t of units of product at the retailer level at each period t). There is production and storage at each level (the units produced by the supplier being either stored at the supplier level, or sent at the retailer level). At each period t, there is, at each level, a setup (fixed ordering) cost if there is a production at this level at t, a cost for each unit of product which is produced at t, and a holding cost for each unit of product stored between $t-1$ and t. Each agent (the retailer and the supplier) has a cost which is the sum of the production and storage costs at his level[7], and each agent wishes naturally to reduce his cost. From a global point of view, it is desirable to minimize the cost of the supply chain, which is the sum of the costs of the retailer and the supplier.

Numerous work consider ways to make the actors of a supply chain cooperate, and this for different supply chain models. Indeed, the supply chain members are primarily concerned with optimizing their own objectives, and this self-serving focus often results in poor performance of the supply chain. The past 20 years has witnessed a high volume of activity in supply chain coordination research, with the overwhelming majority of the results applying to systems with either time-invariant costs and demand parameters, or situations where the time dimension is ignored [30, 85]. Recently, Geunes et al.[85] considered a two-level supply chain composed of a supplier and a retailer, where the supplier has a complete knowledge of all the costs. They model this problem as a Stackelberg game where the supplier sets some of the cost parameters that

[7]More precisely, the cost of the supplier is $\Sigma_t\left(f_t^S y_t^S + p_t^S x_t^S + h_t^S s_t^S\right)$, where f_t^S is the setup cost at period t at the supplier's level, p_t^S his unit production cost, h_t^S his unit storage cost, and $y_t^S \in \{0,1\}$ is equal to 1 if there is a production at the supplier's level at period t, x_t^S is the number of units produced at the supplier's level at period t, and s_t^S is the number of units stored at the supplier's level between periods $t-1$ and t. Likewise, the cost of the retailer is $\Sigma_t\left(f_t^R y_t^R + p_t^R x_t^R + h_t^R s_t^R\right)$ – where R stands for "at the retailer's level".

impact the ordering decisions of the retailer, in order to make the retailer choose a replenishment plan that is good for the supplier.

Coordination when one of the agent has the market power.
As part of the PhD thesis of Siao-Leu Phouratsamay[8], that Safia Kedad-Sidhoum and I have supervised, we have been interested in showing the interest to collaborate in a two level supply chain. We considered two settings. The first one is when the retailer has the market power: he can impose his replenishment plan – the periods at which the supplier sends the product to the retailer, and the number of units of product sent at each period. The second setting is when the supplier has the market power: he decides of a replenishment plan that fulfill the demand of the retailer.

When the retailer has the market power, he chooses to order when he wishes, and chooses thus a replenishment plan that minimizes his cost. Given the costs of the two agents, a trusted third party produces a contract that is made of a replenishment plan and a payment from the supplier to the retailer. The aim is that the cost of the retailer - the cost of his replenishment plan minus the payment he receives - is at most the cost of his favorite replenishment plan. We do this under three assumptions: when no side payment is allowed; when the payment can be done only on the holding costs; or when the payment can be on all the costs. For the second assumptions (side payment on the holding costs), the problem is NP-hard, but we provide dynamic programming algorithms for the cases in which the problem is solvable in polynomial time. Above all, we conduct experiments which show that the decrease of the cost of the supplier in the two last cases is significant, showing the interest for the supplier to collaborate[9]. Note that the benefit due to this cooperation is an upper bound of the benefits induced by any cooperation with other approaches, since it minimizes the cost of the supply chain under the rationality constraint: other approaches can have other advantages – all the costs do not have to be given to the trusted third party by the agents for example –, but they will lead to benefits smaller than or equal to the benefit obtained with this approach. Note also that the supplier and the retailer can share the benefit of the collaboration.

When the supplier has the market power (i.e. is the leader), we also provide several other cooperation scenarios (product stored at the supplier level, with a side payment in exchange; or subcontracting scenarios: the price to produce a unit when it was not decided in the favorite replenishment plan of the supplier is more expensive). We also provide complexity results, algorithms for polynomial cases, and experimental results[10].

The approach which consists of searching for a minimal supply chain cost under the constraint that the cost of the leader is at least the optimal cost he would have without cooperation, is close to the approach done in the previous section: we are looking for the optimal solution which fulfill the rationality constraint. This approach can be used for all the problems with two actors when one has the power to impose his favorite solution to the other agent. When we deal with settings in which no agent has the power to impose a solution to the other, we get a game. In the next section, we will study such a situation, where two agents build/invest on resources, either alone, or together.

3.3 Agents build new resources

In this section, our aim is to show how agents building resources could cooperate. We will focus on a work that we have done with Alexandre Blogowski and Philippe Chrétienne [BCP15], and that we have mentioned in the previous chapter. Let us recall the setting: two mobile operators, A and B, share a set of sites (possible locations for base stations used by the mobile operators). Each operator $i \in \{A, B\}$ has, for each site j, the choice not to invest (in this case it has no profit), or to invest. If the operator invests, it will have a profit s_i^j if the other operator does not invest on site i, or a profit t_i^j if the other operator also invests on site i. The aim of each operator (agent) is to maximize its total profit, which is the sum of the profits it gains on all the sites. We will in particular be interested by the (realistic) case where the costs have a special shape: it is not possible that both operators have incentive to invest alone but not together (i.e. we cannot have:

[8]This thesis is part of the FUI project "Risk, Credit Chain & Supply Chain Management", and has been founded by the "Ile de France" region.

[9]This work is under submission. It can be read on `http://www.optimization-online.org/DB_HTML/2018/07/6702.html`.

[10]This work is under submission. It can be read on `http://www-poleia.lip6.fr/p̃ascualf/doc/2ULSsupplier-leader.pdf`.

$s_i^A > 0, s_i^B > 0, t_i^A \leq 0$ and $t_i^B \leq 0$).

Having a profit at least as good as in our favorite Nash equilibrium.
Problem NSC (for Network Sharing Cooperation) is the following one. The aim is to maximize the total profit of the operators (the sum of their profits) under the constraint that each operator has a profit at least equal to the best profit it would have in a Nash equilibrium. We show that there always exists such a solution:

Theorem 3.2 *[BCP15] There is always a solution in which each operator has a profit at least equal to the best profit it would have in a Nash equilibrium.*

Actually, problem NSC is expressed in a more general way, where the aim is to maximize the overall profit, while the profit of operator A (resp. B) is at least a certain bound P_A (resp. P_B), input of the problem. For our problem, it is easy to calculate the best profit P_i^{Nash} that a operator $i \in \{A, B\}$ could have in a Nash equilibrium, but note that it is not the case for all the problems (finding a Nash equilibria is PPAD complete [41] in the general case). By fixing $P_i = P_i^{Nash}$, we calculate an optimal solution. Note that the returned solution is not necessarily a Nash equilibrium (i.e. an operator could have incentive either to stop investing on a site, or, on the contrary, to invest on a site on which it did not invest). However, the trusted third-party which computes this solution can guarantee the operators that they have a least the profit they would have in any Nash equilibrium, and this may make the operators accept the solution (think that this is a contract: before asking the trusted third-party, the operators agree to accept the solution returned).

As in section 3.1, it would be interesting to take into account fairness criteria to decide which solution to return (a priori among the possible solutions where each agent i has a profit at least P_i^{Nash}). From a practical point of view, we did experiments for problem α-NSC, where we want each operator to have a profit at least equal to α times the best profit it has in a Nash equilibrium – if $\alpha > 1$, it is not always possible that such a solution exists, but in practice, on randomly generated data, when $\alpha \leq 1.1$, this is most often the case [BCP15].

For games for which it is possible, doing a “contract” which guarantee to each agent that she will have a profit at least equal to what she can have in her favorite Nash equilibrium, permits to obtain a good solution for each agent. Note that this approach is well adapted to games where agents build new resources, but can be applied to other games. Doing such a contract would be useful for the agents of the famous “prisoner dilemma” problem in game theory! Unfortunately, there are many games (such as zero sum games, for example) in which such a solution does not always exist. Let us now review a few other issues/approaches to make agents cooperate in these cases.

Other approaches.

- We have seen than the price of stability measures the best quality that a trusted third-party can guarantee when proposing a solution to the agents which is a Nash equilibrium. When the strategy of each agent is not necessarily deterministic but can be mixed (in this case, each agent wants to maximizes her expected profit), correlated equilibrium can be an interesting option. A *correlated equilibrium*, introduced by Robert Aumann in 1974 [16], is a concept that generalizes the (mixed) Nash equilibrium. A trusted third-party recommends to each agent a strategy (a probability distribution over the possible strategies). Knowing these recommendations, each agent has incentive to follow the strategy recommended by the entity: in a correlated equilibrium, each agent would not increase her (expected) profit by choosing a different strategy. Therefore, a (mixed) Nash equilibrium is a particular correlated equilibrium where the choice of the strategies of the agents is not correlated. Since correlated equilibrium generalize Nash equilibrium, the best correlated equilibrium (according to a social cost) is necessarily at least equal to the best Nash equilibrium, and can be better in some cases. Therefore, the “price of stability”, applied to correlated equilibria (i.e. the expectation of the social cost of the best correlated equilibrium divided by the optimal social cost, in the worst instance), can be better than the standard price of stability restricted to Nash equilibria. Moreover, another advantage of correlated equilibria is that they are computationally less expensive than Nash equilibria: computing a correlated equilibrium only requires solving a linear program whereas solving a Nash equilibrium requires finding its fixed point [137]. Some works follow this approach (see e.g. [46]).

- When the problem implies, as in the example above, two agents, then the problem can sometimes be considered as a *two person bargaining problem* [129]. A two-person bargaining problem studies how two agents share a surplus that they can jointly generate. Each agent should have a profit at least equal to the profit she could obtain without the other agent (this is called the *breaking point*, and corresponds to the rationality constraint seen before). There exist several axioms that the outcome of a bargaining problem should satisfy, and several solutions (algorithms) fulfilling each some of these axioms. For example, the *Nash bargaining solution*, proposed by John Nash in 1950, is the unique solution to a two-person bargaining problem that satisfies the axioms of scale invariance, symmetry, efficiency, and independence of irrelevant alternative [129]. Intuitively, the product of the benefits of the two agents is maximized. Other solutions, such as the *Kalai-Smorodinsky bargaining solution* [104], or the *egalitarian bargaining solution* [105], have been introduced 25 years later and satisfy other sets of axioms. Intuitively[11], the Kalai-Smorodinsky bargaining solution maintains the ratios of maximal gains (the ratio of benefits of the agents should be the same than the ratio of the maximum gains they could obtain). On the contrary, the egalitarian bargaining solution attempts to grant equal gain to both agents: it is the point which maximizes the minimum payoff among agents. Note that these solutions, contrarily to what we have done for our operator resources sharing problem seen in the previous section, imply that the utilities are transferable, i.e. one agent can "give" part of her profit to another agent – which is easy to do when profits can be expressed as money, and when agents can exchange money, but which is not always possible otherwise.

- Let us conclude this section by mentioning *cooperative game theory*, which can be useful when there are more than two agents and when agents can make coalitions (agreements between groups of agents). Note that most of the works in this setting consider that the utilities of the agents are transferable. The most famous way of dividing resources in this context is the Shapley value [159], which fulfill desirable axioms. Cooperative game theory is a large research domain, which led to numerous work in Economics Sciences and in computer science [38].

The three above mentioned concepts/areas come from the economic sciences area[12], but are now also widely used in computer science. We will see in the next two chapters other notions coming from the economic sciences (where axioms have been defined), and useful in computer science (where researchers are either interested by using them for their problems, or by computational aspects of algorithms returning solutions which fulfill desirable axioms).

3.4 A few words to conclude.

In this section, we have seen several problems where agents collaborate through a trusted third-party. When agents can solve a problem without the other agents, our aim was to compute a globally optimal solution (for the egalitarian or the utilitarian social choice function), under the constraint that the cost of each agent is at most the cost she would have alone. This is the approach we have taken for the multi-organization scheduling problem and the multi-organization assignment problem. As we have seen in Section 3.2, we can have the same approach for non symmetric agents when one agent is the leader and can impose her favorite solution to the others (this is the case for the two-level lot sizing problem when one of the agents has the market power).

These situations can be seen as particular games where the strategy of each agent is to share her resource (an agent will not share her resources if her utility would be better if she would keep her resources only for herself). In this way, the solutions we have looked for are Nash equilibria which are good for the considered social choice function. More generally, it would be interesting to design, for any game in which a central entity could be used, polynomial time algorithms which return good Nash equilibria. For such games, it would be interesting to compute a a kind of "price

[11] `https://en.wikipedia.org/wiki/Bargaining_problem`.
[12] Note that Aumann, Nash, Shapley, have all three won the Nobel Memorial Prize in Economic Sciences.

of stability obtained in polynomial time", i.e. the worst ratio, over all the instances, of the ratio between the social cost in the best Nash equilibrium which can be computed in polynomial time, and the optimal social cost.

As we have just seen in section 3.3, other approaches can also be taken to obtain good and stable solutions thanks to a central entity which suggesst a solution to the agents. In any case, these solutions can be implemented through a trustworthy third-party: agents give their data (tasks processing time and size for problem MOSP, costs for the 2ULS problem, etc.) to this trusted third-party, and this trusted party uses an algorithm which computes the desired solution. The agents have incentive to participate since they know that the algorithm either guarantees them that they will have a cost smaller than or equal to the cost they would have without cooperating, or guarantees a cost smaller than or equal to the cost they would have in their favorite Nash equilibrium. However, note that, to return such a solution, the algorithm used by the trusted third-party should be robust to manipulation from the agents: an agent should not be able to decrease her cost by giving false information to the trusted third-party. This is the purpose of *mechanism design*, as we will see in the next chapter.

Chapter 4

Agents have private information: truthful algorithms

4.1 Introduction: algorithmic mechanism design

Let us start this chapter by a detour to the field of economic sciences. An important goal of economic theory is to understand what allocation mechanisms are best suited to minimize the economic losses generated by private information[1]. *Mechanism design theory*, initiated by Leonid Hurwicz and refined and applied by Eric Maskin and Roger Myerson, provides tools for analyzing these situations. The 2007 Nobel Memorial Prize in Economic Sciences was awarded to these three researchers "for having laid the foundations of mechanism design theory"[2].

A mechanism is an algorithm that maps a set of individual actions (strategies) to an outcome. Agents may have some information (or private data) that is not known by the other agents, and, in mechanism design, the strategy of each agent is to bid a value (or some values) representing her private value(s). An agent may report false value(s) if this optimizes her own objective function. Thus, a mechanism induces a game, and a stable solution of this game will be a Nash equilibrium. A mechanism is said to *implement* a social choice function f if, for every possible instance (preferences of the agents), it induces a Nash equilibrium in which the outcome is the solution returned by f when it takes in input the preferences of the agents[3].

In mechanism design, the aim is to design a mechanism which induces a good Nash equilibrium for the considered social choice function. Mechanism design has broad applications, from economics and politics (markets, auctions, voting procedures) to networked-systems (internet inter-domain routing, sponsored search auctions, for example). A fundamental principle in mechanism design is the revelation principle.

The revelation principle.
The *revelation principle* states that if a social choice function can be implemented by an arbitrary mechanism, then the same function can be implemented by a mechanism in which the agents have incentive to reveal their true information (and not to lie on their information in order to optimize their own objective function). Such a mechanism is called an *incentive-compatible mechanism*, a *strategy-proof*, or a *truthful mechanism* [131].

In mechanism design, the revelation principle is of utmost importance in finding solutions. Indeed, if the mechanism designer wants to implement some property, or optimize some social choice function, he or she can restrict his/her search to mechanisms in which agents are willing to reveal their private information to the mechanism designer. If no such mechanism exists, no mechanism can implement this property/optimize this social choice function. By narrowing the area needed to be searched, the problem of finding a mechanism becomes much easier[4].

[1]`https : //www.nobelprize.org/uploads/2018/06/popular − economicsciences2007.pdf`
[2]`http : //nobelprize.org/nobel_prizes/economics/laureates/2007/press.html`
[3]Note that the preferences of the agents (over the set of possible outcomes) are usually deduced from the objective functions of the agents and their private data.
[4]`https : //en.wikipedia.org/wiki/Revelation_principle`.

Approach.
The approach usually taken in mechanism design, and that we will consider in this chapter, is the following one. A set of agents have private data (or information), that they declare (bid) to a central authority (a trusted third-party). Based on the bids of the agents, the authority computes a solution (or outcome) that impacts all the agents. Each agent has her own individual objective function, and may bid a false information to the authority if this improves the solution with respect to her objective function. The aim is thus to design a mechanism[5] that is truthful and which optimizes a given social choice function. Let us illustrates this on the facility location problem, which has been extensively studied from a mechanism design perspective.

Optimal truthful mechanism: example of the facility location problem.
Consider the installation of some public service facilities involving n decision makers (agents) representing n districts of a single area, characterized by a metric space (Ω, d), where $d : \Omega \times \Omega \to \mathbb{R}$ is the metric function. The authority announces that k locations will be chosen within the area and runs a survey over the n representatives of the districts; each agent a_i should declare the position (spot) where she would like a facility to be located. The true desired position for a_i is denoted by $x_i \in \Omega$ and her bid is denoted by $x'_i \in \Omega$. Every agent a_i wishes to minimize the distance $d(x_i, y_i)$ between the desired position x_i and the closest facility y_i, possibly by reporting a position $x'_i \neq x_i$ to the authority. The aim is to design a truthful algorithm that maps the reported positions $x' = (x'_1, \ldots, x'_n)$ to a set of k locations where the facilities will be opened at. The aim of the system's designer is usually to minimize the sum of the costs of the agents, i.e. the sum of the distances between the (true) desired position x_i and their closest facility.

A well-known positive result has been given when there is a single facility and when all the positions are on a line. In this case, Moulin [128] gives an optimal truthful mechanism. Assume that the agents are indexed from 1 to n such that $x'_1 \leq \ldots \leq x'_n$ (x'_1 is the position of the leftmost agent and x'_n is the position of the rightmost agent). The algorithm locates the facility on the position of the agent indexed by $\lceil \frac{n}{2} \rceil$, i.e. at the median of the positions bidden by the agents.

Unfortunately, in many cases, it is not possible to get an optimal truthful mechanism. Many works, in computer science, are interested in the best approximation ratio that a truthful mechanism can have for a considered social choice function [131]. This worst case approach, as well as the focus on the research of polynomial time algorithm, which were not considered in the economics area, lead to the creation of the field of *algorithmic mechanism design*, term introduced by Nisan and Ronen in the seminal paper of this field [132]. In this research area, some works also consider the tradeoff between optimality and other desirable properties such as simplicity, robustness, and practicality: this tradeoff is also measured thanks to the best approximation ratio of a simple, robust, and practical approximation mechanism in comparison to a complicated mechanism [94].

The search for the best approximation ratio that a truthful mechanism can have lead however quite often to disappointing results. We illustrate this on the assignment problem, that I have studied with Bruno Escoffier, Jérôme Monnot, and Olivier Spanjaard [EMPS13]. Note that we have also used this illustrative example in a book chapter done with Diodato Ferraioli, Laurent Gourvès, Stefano Moretti and Olivier Spanjaard [FGM+14].

No approximate truthful mechanism: example of the assignment problem.
Let us first define the assignment problem. Consider a set of n tasks $\{t_1, \ldots, t_n\}$ that have to be assigned to a set of n agents $\{a_1, \ldots, a_n\}$. Let $x_{ij} = 1$ if task t_j is assigned to agent a_i, and $x_{ij} = 0$ otherwise. A weight (score) $w_{ij} \geq 0$ can for example represent the interest shown by agent a_i to perform task t_j. The authority has to assign the tasks to the agents. Each agent a_i reports a weight for each task. She wishes to maximize her individual satisfaction $\sum_j w_{ij} x_{ij}$, possibly by reporting false weights to the authority. The aim is to design a truthful algorithm that returns a perfect matching: an assignment of one task to each agent. Two social choice functions can be considered: the utilitarian social choice function aims at maximizing the sum of the weights of the

[5]A mechanism can be considered as an *algorithm*, which, given the values bidden by the agents, outputs a solution. In Section 4.2.3 and 4.2.4, we will see that we can consider that some mechanisms have more "power" than classical algorithms: they can for example give some payments to the agents, or verify that some agents bid their true values. In the remaining of this chapter, expect in these two sections, the terms "algorithm" and "mechanism" can be used interchangeably.

matching returned, while the egalitarian social choice function aims at minimizing the maximum weight of an edge in the matching returned[6]. For both objectives, we get negative results, as shown below for the utilitarian objective.

Theorem 4.1 *[EMPS13] For any $c \geq 1$, there is no $\frac{1}{c}$-approximate truthful deterministic algorithm for the assignment problem with the utilitarian social choice function.*

In order to give a flavor of the construction of such results, let us give a sketch of the proof, which is quite simple. Let us consider an algorithm $\mathcal{A}$ which is $\frac{1}{c}$-approximate for the utilitarian objective. We consider the following instance, I_1, depicted in Figure 4.1 left: two agents a_1, a_2 and two tasks t_1, t_2. The weights are $w_{11} = \gamma > c$, $w_{12} = 0$, $w_{21} = 1$ and $w_{22} = 0$. There are only two possible perfect matchings in the considered instance: $M_1 = \{\{a_1, t_1\}, \{a_2, t_2\}\}$, and $M_2 = \{\{a_1, t_2\}, \{a_2, t_1\}\}$. Since $\gamma > c$, algorithm $\mathcal{A}$, which is $\frac{1}{c}$-approximate, returns M_1 because the optimal weight is the one of M_1, and $\frac{w(M_2)}{w(M_1)} = \frac{1}{\gamma} < \frac{1}{c}$.

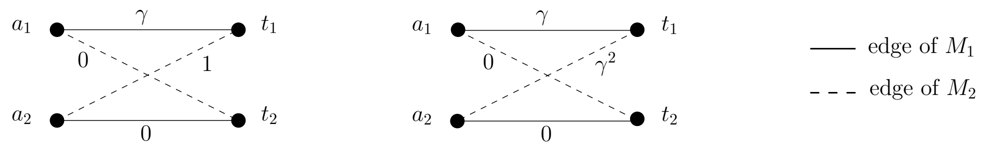


Figure 4.1: Left: instance I_1 (M_1 is the optimal matching, while M_2 is not $\frac{1}{c}$-approximate). Right: instance I_2 (M_2 is the optimal matching, while M_1 is not $\frac{1}{c}$-approximate).

Now, take the situation where $w_{21} = \gamma^2$, all other things being equal (this corresponds to instance I_2 depicted in Figure 4.1 right). The two possible allocations are again $M_1 = \{\{a_1, t_1\}, \{a_2, t_2\}\}$, and $M_2 = \{\{a_1, t_2\}, \{a_2, t_1\}\}$, but now the weight of M_1 and M_2 are γ and γ^2, respectively. Since $\gamma > c$, algorithm $\mathcal{A}$, which is $\frac{1}{c}$-approximate, returns M_2 since $\frac{w(M_1)}{w(M_2)} = \frac{\gamma}{\gamma^2} < \frac{1}{c}$. Consequently, in the first situation, agent a_2 has incentive to declare a false weight $w_{21} = \gamma^2$ in order to get task t_1: this lie will allow her to get task t_1, leading to a utility of 1, whereas she would obtain task t_2, with a utility of 0 by giving her true weights. Hence algorithm $\mathcal{A}$ is not truthful.

When, like in this case, there is no deterministic truthful algorithm having a fixed approximation ratio, we have to look at some techniques which allow truthful algorithms with better performance. In the following section, we describe different ways to obtain a truthful algorithm with an acceptable approximation ratio. The first of these techniques, widely used, is to consider randomized algorithms, instead of deterministic algorithms.

4.2 Techniques to improve the system efficiency

4.2.1 Randomization

A randomized algorithm is an algorithm whose execution depends on the entry of the problem but also on random variables. Therefore, for a given instance, a randomized algorithm does not always return the same solution. If the algorithm used by a central entity to compute the solution is randomized, then the utility (or cost) of an agent may also change in function of the solution returned. The agents bid their private values to the central entity before the execution of the algorithm, and the aim of each agent is therefore to maximize her expected utility (or to minimize her expected cost)[7]. A randomized algorithm is truthful if no agent has incentive to bid values different from her true values.

[6]Note that in [EMPS13] we have considered a slightly more general version of this problem, where p tasks are assigned per agent (this is thus a b-matching problem, instead of a matching problem). There is no big difference with the assignment problem in the techniques used and in the results obtained, and for simplicity, in the sequel we will consider the assignment problem.

[7]Note that we could consider that some agents are risk-averse and thus, for example, wish to maximize the minimum utility they will get. Note however that the vast majority of papers consider that the aim of each agent is to maximize her expected utility.

Note that some works distinguish *universally truthful algorithms*, which are a probability distribution over deterministic truthful algorithms (in this case the algorithm is truthful even when the solution returned by the mechanism is known), from *algorithms truthful in expectation* in which a bidder maximizes her expected profit by bidding truthfully [64]. In their seminal paper introducing algorithmic mechanism design, Nisan and Ronen [132] showed that universally truthful mechanisms can be more powerful than their deterministic counterparts. For some applications, the performance of the best truthful in expectation algorithms is the same than the performance of the best universally truthful algorithms [124]. For some other applications, it is proved that truthful in expectation algorithms are more powerful than universally truthful algorithms [64]. Most designed truthful randomized algorithms are truthful in expectation, and in the sequel, for simplicity, we will the term "truthful" will refer to "truthful in expectation".

Approximation ratio of a randomized algorithm.
The approximation ratio of a randomized algorithm is the maximal ratio, over all the instances, of the expected social cost[8] in the solution returned by the algorithm over the value of the social cost in an optimal solution of the same instance.

We can distinguish two cases. In the first case, which is the most frequent one, for a given instance, the social cost in the solution returned changes in function of the execution of the randomized algorithm. It may therefore happen that the ratio between the social cost obtained after a given execution and the optimal social cost for this instance is larger that the approximation ratio of the algorithm. In the second case, for a given instance, the social cost of the solution returned is the same for any execution of the algorithm on the same instance. In this case, the approximation ratio is fixed, the randomization only helps to prevent the agents from lying. This is for example the case of an algorithm we presented we Eric Angel, Evripidis Bampis and Alex-Ariel Tchetgnia for a scheduling problem [ABPT09]. We randomly sort the tasks allocated to each machine (adding fake tasks for the less loaded machines). This allows us to obtain an optimal randomized algorithm, whereas we have proved with Georges Christodoulou and Laurent Gourvès that no truthful deterministic algorithm has an approximation ratio better than $\frac{7}{6}$ for this problem [CGP07].

Benefit of randomized algorithms.
For most problems, randomization decreases the incentive for the agents to lie. It is common that an algorithm is designed in a way that the expected utility (resp. cost) of each agent is a constant plus a value which does not increase (resp. decrease) when the agent lies. This is for example what we have done for a scheduling [ABPT09] and a facility location problem [EGNK$^+$11].

Another, trivial, way to obtain a (randomized) truthful algorithm is to design an algorithm which does not take into account the values bidden by the agents. The agents obviously do not have incentive to lie since theirs bids have no impact on the solution which is returned. However, of course, the social cost of the algorithm will probably be bad, even if it is often better that the worst case performance that a truthful deterministic algorithm can have. For example, for our assignment problem described in the previous section, the algorithm which returns a random perfect matching, among all the possible matchings, is $\frac{1}{n}$ approximate, whereas we have seen that there is no $f(n)$-approximate deterministic truthful algorithm [EMPS13]. For this problem, we can show that there is no randomized truthful algorithm with a better approximation ratio [EMPS13][9]. In this case, as well in some others, randomization is not sufficient to obtain a good approximation ratio, and we will have to resort to other techniques, presented below, to obtain better approximate algorithms.

4.2.2 Considering specific classes of instances

When no good (deterministic or randomized) truthful algorithm is known (or even possible) for a problem, a natural thing to do is to isolate classes of instances for which we can get an approximate truthful algorithm. This is the case for the facility location problem expressed above, for which it is still unknown whether there exists a deterministic truthful algorithm with bounded approximation ratio in a general metric space. Many works consider this problem, which has numerous

[8]We recall that the social cost is the value of the social choice function.

[9]The technique used is very similar to the one used in the proof of Theorem 4.1 seen above - the cost of an agent is just replaced by her expected cost.

applications. As we have seen before, Moulin [128] has given a truthful algorithm for placing one facility on a line which is optimal for the utilitarian objective. Other classes of instances have been considered, as for example, when the facilities have to be be placed on a circle [142], on a tree [154], or on a graph [154, 3]; when two facilities have been open [142, 118, 79], or when more than 2 facilities have to be open [78, 80]. For each of these subproblems, the authors give lower bounds and upper bounds on the best approximation ratio that a truthful deterministic or randomized algorithm can have.

With Bruno Escoffier, Laurent Gourvès, Thang Nguyen Kim, and Olivier Spanjaard, we have studied the extreme case where there are $n-1$ facilities for n agents [EGNK$^+$11], and we provided lower and upper bounds on the best approximation ratio a truthful algorithm can have for the tree metric space or the general metric space.

Besides its theoretical interest, considering special classes of instances permits to determine, when we consider a given instance, which algorithm is the best one, in term of approximation ratio, among truthful algorithms which can be applied to this instance. However, usually, we do not have realistic special cases for which there are truthful algorithms with ratios much better than the ones of truthful algorithms for the general case. As we will see in the next section, giving payments to agents is a very popular and efficient way to obtain good truthful algorithms.

4.2.3 Mechanisms with monetary transfers

A *mechanism with money* consists of an *algorithm* which computes a solution (an outcome) based on the bids of the agents, and a *payment function* which gives to each agent i a payment $p_i(o)$ depending on the outcome o computed. In this setting, we consider that the utility $u_i(o)$ of agent i can be expressed monetarily. The objective of an agent is then to maximize $u_i(o) + p_i(o)$[10]. The seminal work of Nisan and Ronen [132] considered monetary transfers, and most of the works in algorithmic mechanism design have been carried out with this model [131], in particular at the beginning of the algorithmic mechanism design field. This is perhaps due to the fact that monetary transfers are extensively used in mechanism design, in economics sciences.

A famous result in economic literature is the Vickrey-Clarke-Groves auction. It is a type of sealed-bid auction of multiple items: agents (bidders) value several items and report to the mechanism their valuations for the items. This mechanism, which is truthful and which assigns the items to the agents who value them the most (which is considered as a socially optimal manner), is a generalization of Vickrey auction, where there is only one item and where the highest bidder wins the item but pays the price of the second-highest bid. Let us now look at two general mechanisms which are extensively used in algorithmic mechanism design. The first one is based on the Vickrey-Clarke-Groves auction.

Some famous mechanisms.

- A *Vickrey-Clarke-Groves (VCG) mechanism* is a generic truthful mechanism that implements the utilitarian social choice function for problems where we wish to select any outcome out of a set of possible outcomes. A typical mechanism in the VCG family works in the following way: each agent bids her value for each possible outcome[11]. Based on these values, the mechanism outputs an optimal solution s^* for the utilitarian social choice function. It also gives to each agent an amount of money which depends on the total values of the *other* agents in s^*[12].

  Note that VCG mechanisms need to calculate the optimal solution, based on the agents' bids. Therefore, if the combinatorial optimization problem considered is NP-hard, then these mechanisms won't always output the result in polynomial time (unless $P \neq NP$). For problems which can be solved in polynomial time, VCG mechanisms are however well adapted. These mechanisms have therefore been used for many computer science problems, as for example the problem which consists in buying a path in a network [131].

[10]This is similar if each agent wants to minimizes her cost $c_i(o)$: in this case the cost of each agent has to be expressed monetarily and the objective of each agent is then to maximizes $p_i(o) - c_i(o)$.

[11]Usually each agent bids a single private value (or a few private values), and we deduce the utility/cost of the agent for each of the possible outcomes given her objective function and her private values.

[12]The solution returned always maximizes the sum of the utilities of the agents given the private values obtained. The payment function, which is given by the VCG mechanism, ensures that the agents have incentive to bid their true values.

- A second famous family of mechanisms has been given by Aaron Archer and Éva Tardos [12] for situations where each agent's private data is expressed by a single positive real number. The goal of the considered mechanisms is to allocate loads placed on the agents, and the agent's private data is the cost she incurs per unit load. Archer and Tardos give an exact characterization of the algorithms that can be used to design truthful mechanisms for such load balancing problems using appropriate side payments. This setting can for example be used for scheduling problems, maximum flow or special case of facility location [12].

Ad hoc mechanisms.
For problems for which we cannot use the above mechanisms, (ad hoc) mechanisms using money also usually allow to reduce the approximation ratio compared to mechanisms without money. Such mechanisms have for example allowed us to get a $(1+\varepsilon)$-approximate algorithm for a scheduling problem on related machines when the agents are the tasks and their private data are their lengths [ABPT09]. We also obtained an optimal algorithm for a scheduling problem on a machine when the agents are the tasks and their private data are their lengths and weights [ABPT16].

Screenings games.
Truthful mechanisms with money are also used in the context of screening games [164]. Screening games are particular two-player gamed used in economics science. Such games model situations where there are two agents, which have each one their own objective, and where one agent (called the principal-agent), who does not know all the data of the other agent, makes offers to the second agent. The second agent accepts the offer that fits her best (or rejects all the offers if none suits her). Examples of truthful screening games in supply chains have been given by Cachon [30]. We studied a screening game for the 2ULS problem that we have seen in the previous chapter, and when the retailer has the market power[13]: we assume that the supplier knows the different "types" of retailers which may contact him (theirs ordering and holding costs), and the probability that a retailer is of a given type. However, the supplier does not know of which type a given retailer is. The supplier proposes him several contracts (each contract is made of a replenishment plan plus a side payment), and the retailer chooses one of these contracts (we insure that among these contracts there is a contract for which the cost is at most the cost of the optimal replenishment plan of the retailer). Like in the previous case with complete information, our aim is to design contracts in order to minimize the cost of the supplier. Given the revelation principle, we focus our study on the set of contracts which ensures that the retailer will choose the contract that corresponds to his true type. We showed that restricting to truthful contracts, in the worst case, can arbitrarily decrease the gain of the supplier, but that in practice the gain of the supplier is close to the gain he would obtain if he had complete information.

Monetary transfers are usually very useful, but they are not always sufficient to obtain truthful algorithm with good approximation ratio. Moreover, in certain applications, money is unavailable, morally unacceptable or might simply be at odds with the objective of the mechanism [77]. In some application, for infrastructural reasons, payments are also not always suitable. For example, most Internet mechanisms, including its congestion control mechanisms, do not currently permit monetary transfers [94]. As we have seen for the facility location problems in Section 4.2.2, many works consider thus mechanisms without payments [155]. In the following section, we will see another way to obtain good algorithms despite the fact that agents have private data.

4.2.4 Verification

Mechanisms with verification are mechanisms which have the power to verify that the agents give their true values. Verifying the declarations of most agents and imposing large penalties on liars should suffice to obtain good truthful mechanisms [34, 81]. However, verifying that the agents give their true private values has a cost. A mechanisms with *selective verification* selects few critical agents and detects, using a verification oracle, whether they have reported their true values. If this is the case, the mechanism returns the solution it has computed with the bidden values. Otherwise, it ignores any misreports and proceeds recursively with the remaining agents [81].

Let us take as an example the problem where we want to place one facility one a line, when the social choice function is the minimization of the maximum distance of an agent to the facility.

[13]This work is under submission. It can be read on `http://www.optimization-online.org/DB_HTML/2018/07/6702.html`.

Contrarily to the case of the utilitarian social choice function, for which, as we have seen in the beginning of this chapter, there exists a truthful optimal algorithm [128], there is no $(\frac{3}{2} - \varepsilon)$-approximate truthful algorithm (even if randomization is allowed) for this problem. Note that if we inspect the bidden locations of the two extreme agents and verify that they coincide with their preferred locations, then we obtain a truthful optimal algorithm: if the agents did not lie, the algorithm places the facility at equal distance between them; otherwise, it ignores their bids and recurse on the other agents. With this mechanism, non extreme agents do not affect the facility allocation, whereas the two extreme agents do not have incentive to lie because of the verification step. This example motivated Fotakis, Tzamos and Zampetakis [81] to introduce mechanisms with selective verification for the setting of utilitarian voting with m outcomes and n agents, where each agent has a non negative utility for each outcome (there is no monetary transfer).

Mechanisms with verification either consider penalties for liars (this can be useful for mechanisms with money), or they simply ignore the preferences of liars when they compute the solution to return. A large amount of work analyzes the impact of verification on the (worst case) performance of the mechanism. Some other works show that partial verification (where verification only verifies that there the bidden value is not too far from the true value) is not useful [11, 34, 82]. Caragiannis et al. introduced *probabilistic verification* [34]: they assume that any deviation from the truth is detectable with a probability depending on the distance of the misreport to the true value. They use this to implement mechanisms with money and with large penalties when agents lie. Some other works are interested in cases where the mechanism can detect certain lies of the bidders [77, 78, 141]. Let us now look at a scheduling problem that Eric Angel, Evripidis Bampis, Nicolas Thibault, and I have introduced [ABPT16], and which could be placed in this category.

A scheduling problem: how the use of preemption can improve the performance of a truthful algorithm.

A set of n agents schedule each one a task on a set of m machines. Each agent wants to minimize the completion time of her task. Each agent has a private length (execution time) and possibly a private weight. Given the values bidden by the agents, an algorithm computes a schedule of the tasks. The social choice function consists in minimizing the weighted sum of completion times.

The aim is to study the best approximation ratio that a truthful algorithm can have. This problem is in the same vein as previous works [ABP06b, CGP07, ABPT09][9] where agents had each one a task with a private length and where the aim of each agent was to minimize the completion time of her task. The difference in this problem is that each agent may also have a private weight, and that the social choice function is the minimization of the weighted completion times of the tasks (whereas in the previous work the social choice function was the egalitarian social choice function: the minimization of the makespan)[14].

We defined *preventive preemption* for this problem: if a task bids a length l_i, then the schedule is allowed to preempt it l_i time units after its starting date, and to schedule it again later (therefore, preemption of a task happens only if the task didn't bid its true length). As we will see, this will allow to decrease the approximation ratio that a truthful algorithm can have. We studied, for several cases (one or several machines, private weights or not), the impact of several techniques: the use of randomization, of monetary transfers, and of preventive preemption on the performance of a truthful algorithm. For example, when there is one machine and when the tasks have private lengths (but no weight), without preemptive preemption, there is not $(2 - \varepsilon)$-deterministic truthful algorithm, and no $(\frac{3}{2} - \varepsilon)$-randomized truthful algorithm (both without monetary transfer). With preemptive preemption, we obtain an optimal deterministic algorithm without monetary transfer (this algorithm only consists in scheduling the tasks in non decreasing order of their bidden lengths, and to schedule the tasks which have not been completed in a round robin policy at the end of the schedule). The use of preemptive prevention is stronger than the use of money in this case since there is no optimal truthful algorithm with payment. Note however that preventive preemption is useful only when the tasks lie on their lengths: when tasks have private weights, then monetary transfers and preemptive prevention are necessary simultaneously to obtain an optimal truthful algorithm [ABPT16].

[14]Note that Heydenreich et al. [96] also consider this social choice function, in a different setting: they designed truthful coordination mechanisms using menetary transfers.

The use of preemption gives the possibility to postpone a task which bids a length smaller than its true length: knowing this, the agents do not have incentive to bid lengths smaller than their true lengths. In the next section, we will look at problems for which we know that agents cannot bid any value.

4.2.5 Taking into account the way the agents can lie

For some problems, one can assume that the agents cannot bid any possible value. In particular, in several applications, we can consider that the agents can not overbid (i.e., bid values values that are larger than the true values), or underbid (i.e., bid values values that are smaller than the true values). Indeed, in some applications, the application can detect that an agent lies when it bids in one direction (as we will see in some examples below), or because the agent has not incentive to bid in a certain direction (e.g. when an agent bids her need of a given resource and the amount of resource given by the algorithm to each agent is exactly the value she has bidden – an agent will not be satisfied if she has not an amount of resource which fulfills her need).

Auletta et al. [15] studied combinatorial optimization problems involving one-parameter agents considered by Archer and Tardos [12] (see section 4.2.3), and with monetary transfers. They show that if agents can lie in one direction (that is, they either overbid or underbid) then any (polynomial-time) c-approximation algorithm, for the optimization problem without selfish agents, can be turned into a (polynomial-time) $c(1+\varepsilon)$-approximation truthful mechanism, for any $\varepsilon > 0$. We can apply this to a scheduling problem where agents are the machines that bid their speed, and when monetary transfers are given to the agents only after the machines have completed the tasks assigned. Indeed, in this case, the mechanism can detect that an agent declared a speed larger than her real speed: knowing this, the agents do not want to be penalized (the mechanism could withdraw their payment) and they do not overbid their speed.

We studied the assignment problem, that we have presented in section 4.1, in the case where the agents cannot underbid or overbid [EMPS13] (and without using menetary transfers). This problem arises for example when a town council has to share a public room between n sport or cultural associations: it has to allocate one time slot (task) to each association (agent). To this end, for each time slot t_j, each association a_i indicates the expected number w_{ij} of members that would attend the activity (w_{ij} is the participation level). It is natural in this situation to assume that the agents do not overbid. Indeed, as soon as the actual participation level at the various activities is controlled afterwards, an association cannot overbid, at the risk of losing credibility.

Whereas there there is no ε-approximate truthful deterministic algorithm for this problem with the utilitarian objective, for any $\varepsilon > 0$, as we have seen in Theorem 4.1, the assumption that the agents cannot overbid enables to obtain a simple $\frac{1}{2}$-approximate deterministic truthful algorithm for this setting:

> Sort the edges by non-increasing weights. Let $M = \emptyset$ and $(e_1, \ldots, e_m)$ denote the sorted list of edges. For i from 1 to m, if $M \cup \{e_i\}$ is a matching then $M = M \cup \{e_i\}$. Return M.

Theorem 4.2 *[EMPS13] When the agents cannot overbid, the above algorithm is a truthful $\frac{1}{2}$-approximate algorithm for the assignment problem with the utilitarian social choice function.*

With a proof close to the one used for Theorem 4.1, we can show that there is no truthful deterministic algorithm which has a better approximation ratio when the agents cannot overbid. The assumption that the agents cannot underbid does not change anything for the utilitarian social choice function, but it allows to obtain an optimal deterministic truthful algorithm for the egalitarian social choice function (whereas the best - randomized - truthful algorithm simply consists in giving a random matching, and is $\frac{1}{n}$-approximate...) [EMPS13].

As we have seen with the work of Auletta et al. and with the assignment problem presented above, assuming that the agents can lie in only one direction is a very powerful tool to decrease the approximation ratio of truthful algorithm. Other applications include, for example, scheduling algorithms, where the tasks bid their lengths [ABP06b], or mechanisms for hiring a matroid base (where it is assumed that the cost of an agent is equal to the maximum of her declared and actual cost) [141].

4.3 A few words to conclude.

Applications where there exist an optimal truthful algorithm, as is is the case for the placement of a facility on a line in order to minimize the utilitarian social choice function, are quite rare. Contrarily to what we have seen in the first chapter, this is not a problem of complexity: these results are valid even if we consider non polynomial time algorithms. To circumvent this problem, algorithmic mechanism design attempts to associate computation and incentives by using several tools: by using randomized algorithms, we can decrease the difference of utilities the agents can have if they change their bids; by using monetary transfers between the mechanism and the agents, we can change the utilities of an agent; by using selective verification mechanisms or taking into account the way the agents can lie, we can threaten the agents to decrease their utilities if they are seen lying. The most efficient is often to use several of these tools simultaneously. For example, it is very useful to use randomized algorithms when we know that the agents cannot underbid (or overbid): we can express the utility of an agent as a constant plus something which can only decrease when we know that the agent cannot underbid (or overbid). As we have seen, when it is possible, monetary transfers also help by allowing the mechanism to set the utility of each agent as a constant plus something which depends on the bids of the other agents.

A nice thing in this domain is that tools to improve the performance of an algorithm are quite open, and, depending on the application considered, we can imagine new ways to improve the performance of algorithms. For example, for a scheduling problem where agents bid the lengths of their tasks to the mechanism, we have introduced two models [CGP07]: one where each agent gets the result of the execution of her task as soon as the task is completed, even if her execution is faster than the length bidden by the agents, and another model where each agent has to wait, after the beginning of the execution of her task, the duration that she has bidden. This simple "retention of the results" technique permits to improve significantly the worst case performance of truthful algorithms. Another example concerns the multiple facility location problem, where Fotakis and Tzamos [78], due to the absence of any positive result on the approximability of this game, considered a variant of the game where an authority can impose on some agents the facilities where they will be served. Therefore, contrarily to classical optimization problem, where there exist lower bounds of the approximation ratio of an algorithm, and where imagination is useful to design approximation algorithms matching these lower bounds, or to exhibit such lower bounds, in algorithmic mechanism design, the way to solve a problem, introducing new tools to improve the approximation ratio of an algorithm, can also be imaginative and not restrictive.

Note that stronger notions of truthfulness exist. In the same way as strong Nash equilibria are more resistant than Nash equilibria to coalitions of agents, some works [111, 80, 125] consider mechanisms that are resistant to coalitions of agents: while a mechanism is truthful (or strategyproof) if no agent may misreport her private data and be better off, a mechanism is *group-strategyproof* if no coalition of agents benefits by jointly misreporting their private data. All the problems studied in term of truthfulness can thus also be studied in term of group-strategyproofness, if agents have the possibility to make coalitions.

Let us conclude this chapter by mentioning works that consider *almost truthful mechanisms* [70, 116, 114]. For problems where it is proved that there is no good truthful algorithm, some authors consider ε-truthful mechanisms, which may have slightly different definitions, but which all mean that an agent should not gain more than a small amount by bidding a false value. This is close in spirit to approximate Nash equilibria: as it is assumed that an agent will not deviate from one assigned strategy in an approximate Nash equilibrium, it is assumed here that it is not worth lying for an agent when the amount of utility gained by doing this is too small. This can be seen as another tool to circumvent the impossibility of getting good truthful algorithms.

Chapter 5

Computing a collective solution for agents with different preferences

In this chapter, we consider problems which deal with resources (or objects) belonging to all the agents. A solution common to all the agents has to be returned, knowing that each agent has a preference on each of the possible outputs. Among these problems, we can cite the multi-objective maximum Traveling Salesman Problem (TSP), where the aim is to find a tour (an Hamiltonian cycle) of maximum profit, given that agents have different profit on each tour (see Section 5.1 for this problem). We can also cite the problem where agents have preferences on the order in which common tasks will be done (collective schedule problem - see Section 5.2). Two broad research domains deal with this kind of problems: *multi-objective combinatorial optimization* [91], and *combinatorial social choice* [27]. We will briefly present these domains and illustrate some of their concepts on the bi-objective maximum TSP problem, on the bi-objective matching problem, and on the collective schedule problem.

5.1 Multi-objective combinatorial optimization

In a multi-objective combinatorial optimization problem, a problem has a (finite) set of feasible solutions S, and $k > 1$ objective functions $f_i : S \to \mathbb{R}$, $i \in \{1, \ldots, k\}$. These functions can be viewed (and are often presented as) conflicting objectives of a single agent, who is looking for solutions which present good compromises of the different objectives. These functions f_i can also be seen as objectives of different agents: f_i is thus the objective of Agent i, and the aim is to return a solution which is a good compromise of the different objectives of the agents. In the sequel, as it is common in multi-objective combinatorial optimization problems, each value $i \in \{1, \ldots, k\}$ will be called an *objective*. In a multi-objective combinatorial problem, it is specified for each function f_i whether it should be minimized or maximized. For simplicity, we will assume here that each function f_i has to be maximized. The value of the feasible solution x on objective $i \in \{1, \ldots, k\}$ is denoted by $f_i(x)$. Thus, the vector $(f_1(x), f_2(x), \ldots, f_k(x))$ characterizes the performance of solution x. Comparing solutions consists thus in cfomparing vectors. Several approaches can be used to tackle this kind of problems.

5.1.1 Approaches

The first kind of approaches consists in returning *all* the solutions which may be "interesting". A solution x is considered as interesting if there is no other feasible solution x' such that x' is at least as good as x on all the objectives, and strictly better on at least one objective. In this case, solution x is a *Pareto Optimal* solution.

Figure 5.1 shows the set of the feasible solutions for an instance of a two-objective problem. In this figure, black disks are Pareto-optimal solutions, while white disks are the other feasible solutions (these solutions are said to be *dominated*). The *Pareto set* (a.k.a. Pareto frontier, or

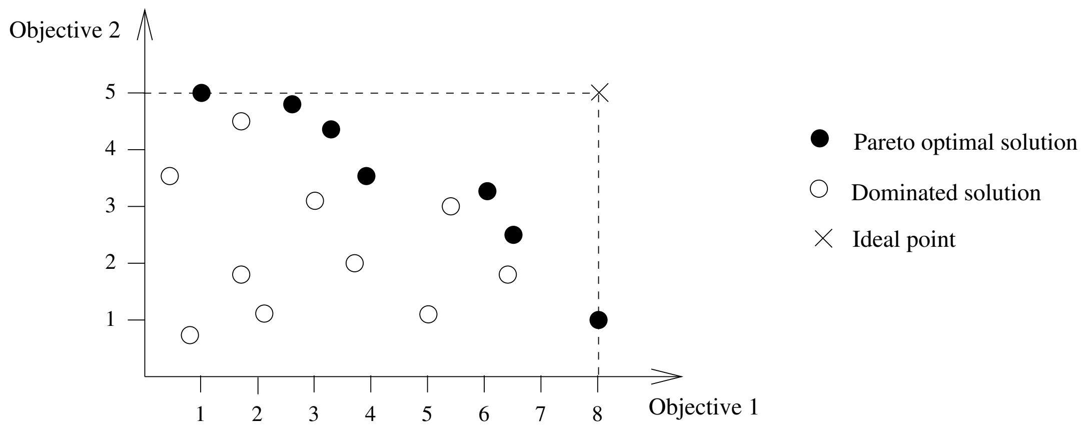


Figure 5.1: Small disks represent the set of feasible solutions S of an instance of a bi-objectives problem. The Pareto set is the set of black disks.

Pareto curve [138]) is the set of Pareto optimal solutions (the set of black disks in Figure 5.1).

Determining the Pareto set (or a subset of the Pareto set).
Many works in multi-objective combinatorial optimization aims at returning the whole Pareto set [68]. Since the number of Pareto optimal solutions can be exponential[1], some works aim at returning an ε-approximate Pareto set, which is a set S' a feasible solutions such that, for each feasible solution $s \in S$, there exists a solution $s' \in S'$ such that for all $i \in \{1, \dots, k\}$, $f_i(s') \geq (1-\varepsilon) f_i(s)$. For each multi-objective problem, there always exists a ε-approximate Pareto set whose number of solutions is polynomial in the size of the instance and in $\frac{1}{\varepsilon}$ [138].

The other approaches aim at returning a single "good" solution, where the notion of "good" can be seen in several ways. The most classical approaches are the following ones.

Linear scalarization.
A way to proceed is to associate a weight w_i to each objective function i, and then to return a weighted sum of the objective functions: $\max_{x \in S} \sum_{i=1}^{k} w_i f_i(x)$. This linear scalarization returns a Pareto optimal solution.

The budget approach (also called *ε-constraint method*).
In this method, the decision maker has to assign thresholds that he/she wishes to achieve for all but one objectives. These values are incorporated into the problem as additional constraints. The aim is to optimize one objective, while the other objectives are constrained[2]:

$$\begin{array}{ll} \max & f_i(x) \\ \text{such that} & x \in S \\ & f_j(x) \geq B_j \text{ for } j \in \{1, \dots, k\} \setminus \{i\}, \end{array}$$

This method is often used for scheduling problems, when agents have each one their one set of tasks that they schedule on a shared machine, or when several objectives have to be optimized simultaneously on a same set of tasks [2, 152, 7]. This approach is also used for some multi-objective network-design problems [145], shortest path [95], and the matching problem, and some generalizations [22, 91].

Goal Programming.
Goal Programming [39] is a method close to these two previous methods. In this method, the decision maker has to assign a target B_i that he/she wishes to achieve for each objective i. The objective function consists in minimizing the absolute deviations from the targets to the objective:

[1]it is for example the case of the bi-objective shortest path problems [92].

[2]We present it for a maximization problem, but of course the same approache can be used for minimization problems: the aim is to minimize $f_i(x)$ under the constraint that $f_j(x) \leq B_j$ for $j \in \{1, \dots, k\} \setminus \{i\}$.

$\min_{x \in X} \sum_{i=1}^{k} |f_i(x) - B_i|$.

In these previous methods, a decision maker has to decide the weights of the objectives, or the thresholds (or goals) fixed on each objective. In the following method, that we will use in the following section, the aim is to optimize all the objectives in a similar way.

The ideal point approach.
For each $i \in \{1, \dots, k\}$, let opt_i be the value of the optimal solution for objective i: $opt_i = \max_{s \in S} f_i(s)$. For a given instance of a multi-objective problem, the *ideal point* $(opt_1, opt_2, \dots, opt_k)$ is the image of a (not necessarily feasible) solution reaching optimality on all objectives at the same time. For example, on Figure 5.1, the ideal point, which here does not correspond to a feasible solution, is point (8,5) since the best value of a feasible solution for objective 1 (resp. objective 2) is 8 (resp. 5).

In practice, it is unlikely that the ideal point is a feasible solution. We thus need to resort to approximation. A feasible solution is said to be a ρ-approximation of the ideal point (or, in short, a ρ-approximate solution), for $\rho \in (0, 1]$, if $f_i(x) \geq \rho \; opt_i$, for all $i \in \{1, \dots, k\}$. It is challenging to identify, for a given problem, the best approximation ratio under which the existence of a feasible ρ-approximate solution is always guaranteed. It is also relevant to have a constructive approach: what is the largest ratio $\rho \in (0, 1]$ such that a feasible ρ-approximation of the ideal point can be computed in polynomial time?

Approximation of the ideal point has been done for the multi-objective versions of many well-studied optimization problems. It has been used for scheduling problems with two classical objective functions (e.g. makespan, sum of completion times, etc) [163, 144, 6]. For these problems, the authors show positive results, with the existence of ρ-approximate schedules, where ρ is a constant. This approach has also been used for the bi-objective version of the traveling salesman problem [120, 121], for the bi-objective maximum spanning tree problem [69], for the maximum cut problem [8], and generalization of these problems [88, 19].

Since we wish to obtain a single solution, and since we do not want to fix weights or bounds on the different objectives, our work [BGMP11, BGMP13, GMPV17], presented in the next section, focus on the ideal point approach. Note that numerous other methods exist, since multi-objective optimization is a large research domain. An overview of the field of multi-objective optimization can be found in [56, 68].

5.1.2 The bi-objective TSP and the bi-objective matching problems

In this section, we focus on bi-objective problems: the bi-objective max traveling salesman problem (TSP), and the bi-objective matching problem. We will consider a graph, where each edge e has two labels (valuations): a non-negative weight $w(e)$ and a non-negative length $\ell(e)$. The weight of a set of edges is the sum of its edges' weights. Likewise, the length of a set of edges is the sum of its edges' lengths. We will refer as *cost* of e the couple $(w(e), \ell(e))$ of labels of each edge e. Given a set of edges M, $w(M)$ (resp. $\ell(M)$) represents the sum of the weights (resp. lengths) of the edges of M.

In the bi-objective TSP, the first objective consists in finding a Hamiltonian cycle of maximum weight, while the second objective consists in finding a Hamiltonian cycle of maximum length. In the bi-objective matching problem, the first objective consists in finding a matching of maximum weight, while the second objective consists in finding a matching of maximum length.

Since there does not necessarily exist solutions which are a good approximation of the ideal point in general graphs, besides general graphs, we will consider graphs where both labels (the weight and the length) fulfill the triangle inequality, and graphs where one label (wlog. the length), fulfills the triangle inequality[3].

The bi-objective max TSP problem.

The ideal point approach has been used on the bi-objective version of the traveling salesman

[3]A graph fulfills the triangle inequality for label l if for each triple of vertices a, b, c, then $l(\{a, b\}) \leq l(\{a, c\}) + l(\{c, a\})$.

problem in [120, 121]. For general graphs, Manthey [121] shows that no single tour can be $(1/3+\epsilon)$-approximate ($\epsilon > 0$). To show this, he exhibits the graph G of Figure 5.2, for which there is no $(1/3+\epsilon)$ approximation of the ideal point. In this graph, the solid edges plus two dotted edges form a Hamiltonian cycle of cost $(3,0)$. The dashed edges plus two other dotted edges form a Hamiltonian cycle of weight $(0,3)$. The ideal point has thus a cost of $(3,3)$. In order to obtain a $(1/3+\epsilon)$ $(1/3+\epsilon)$, we would need to return an Hamiltonian cycle with at least two solid edges and two dashed edges, which is impossible. This result on a 5 vertices example can be extended to graphs of any size [BGMP13], so that this result is true even for large graphs.

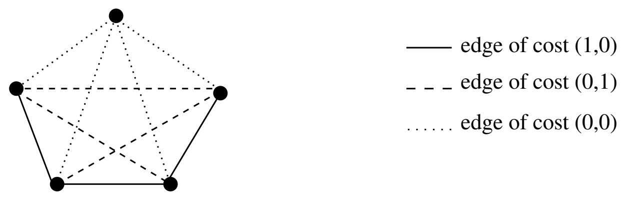


Figure 5.2: Graph for which there is no $(1/3+\epsilon)$ approximation of the ideal point. Solid edges are of cost $(1,0)$, dashed edges are of cost $(0,1)$, dotted edges have cost $(0,0)$.

If the single objective max TSP problem is ρ-approximable, Manthey [120] shows that there is a $\frac{\rho}{3}$-approximation of the ideal point for the biobjective max TSP. Taking the best polynomial time approximation algorithms known so far for the symmetric max TSP, he derives a $\frac{7}{27}$-approximate (resp. $\frac{7}{24}$-approximate) tour without (resp. with) the triangle inequality.

With Cristina Bazgan, Laurent Gourvès and Jérôme Monnot, we propose a generic algorithm [BGMP11, BGMP13] which returns an Hamiltonian cycle which is a ρ-approximation of the ideal point, the ratio ρ depending on the fact that labels fulfill the triangle inequality or not. The algorithm consists in computing a maximum weight matching M_w of G, and a maximum length matching M_ℓ of G. For each component $C_i \subset (M_w \cup M_\ell)$ which is a cycle, we remove the edge in $C_i \cap M_w$ which has minimum weight. We thus obtain a set of paths, and add edges in order to connect these paths and obtain an Hamiltonian cycle. By carefully adding the right edges in the last step, we can assure that when both objective functions fulfill the triangle inequality, the approximation ratio is $\frac{5}{12} - \varepsilon \approx 0.41$, and when one objective function fulfills the triangle inequality, the approximation ratio is $\frac{3}{8} - \varepsilon$. When the triangle inequality is not assumed on any objective function, the algorithm is $\frac{1+2\sqrt{2}}{14} - \varepsilon \approx 0.27$-approximate.

The bi-objective matching problem.

Let us now have a look at the bi-objective matching problem. The first objective consists in returning a matching of maximum length, while the second one consists in returning a matching of maximum weight. We will assume that both objectives satisfy the triangle inequality, since otherwise no approximation of the ideal point is possible [GMPV17]. This problem has been studied in [19], where the authors provide an approximation method for a class of biobjective problems. An application of this method to the bi-objective maximum matching yields a $\frac{1}{6}$-approximation of the ideal point.

With Laurent Gourvès, Jérôme Monnot, and Daniel Vanderpooten, we have shown that the problem which consists in returning the best approximation of the ideal point is NP-complete, even when both objectives fulfill the triangle inequality [GMPV17]. We proposed a general method which, combined with results stating the existence of a good tradeoff in graphs of small size, implies the existence and computation of a good tradeoff solution in a graph of any size.

Approximation. The theorem below provides a general method, which, combined with some results for small graphs provides a solution which is a $\frac{1}{3}$-approximation of the ideal point.

Theorem 5.1 *[GMPV17] Let $k \geq 2$ be an integer, and let ρ be a real in $(0,1]$. Suppose that for every $q \leq 2k$ and every complete graph K_q of q nodes, there always exists a matching which is a ρ approximation of the ideal point. Then, for any instance of the biobjective Maximum Matching problem, there exists a matching which constitutes a $\rho\frac{k-1}{k}$-approximation of the ideal point.*

The proof of this theorem is constructive. The idea is to start with Opt_w and Opt_ℓ, which are a maximum weight matching and a maximum length matching respectively. The union of these matchings is a set of disjoints cycles and of paths. Starting from this subgraph, and considering several cases, we are able to find a $\rho\frac{k-1}{k}$-approximation of the ideal point.

This theorem requires that, for small complete graphs, there always exist a matching which is a good tradeoff between both objectives, i.e. a ρ-approximation of the ideal point. We have shown, by case analysis, that for complete graphs with at most 6 vertices, with label on edges w and ℓ satisfying the triangle inequality, there exists a matching which is a $\frac{1}{2}$-approximation of the ideal point. Using this result with $\rho = \frac{1}{2}$ and $k = 3$ in Theorem 5.1, we obtain that we can build in polynomial time a $\frac{1}{3}$-approximation of the ideal point.

A computer aided approach. The proof of the existence of $\frac{1}{2}$-approximate matchings for complete graphs with at most $n = 6$ nodes is analytical. Our proofs become more and more tedious as n grows, and it is difficult to follow the same approach for larger n. In order to test the existence of a single $\frac{1}{2}$-approximate matching for larger values of n, we propose a mixed 0-1 linear programming formulation. This formulation provides, for any value of n, the best possible approximation ratio ρ^*. In our formulation, we consider, for a given n:

- the set $\mathcal{M}_n$ of all maximal matchings which can be defined on a complete graph of size n.
- the set $\mathcal{I}_n$ of all possible instances corresponding to a complete valued graph $G = (V, E)$ with $|V| = n$ where each edge $e \in E$ has two values $w(e)$ and $\ell(e)$ which satisfy the triangle inequality. We assume wlog. that $0 \leq w(e) \leq 1$ and $0 \leq \ell(e) \leq 1$ for each edge $e \in E$. Moreover, we impose that Opt_w and Opt_ℓ, which are optimal matchings for w and ℓ respectively, are such that $w(Opt_w) = \ell(Opt_\ell) = 1$. If these conditions are not met, then compute $w(Opt_w)$ and $\ell(opt_\ell)$, and for each edge e, replace $w(e)$ by $w(e)/w(Opt_w)$ and $\ell(e)$ by $\ell(e)/\ell(Opt_\ell)$. This transformation has no incidence on the approximation guarantee because it is a ratio. We also assume the worst case situation where the subgraph induced by $Opt_w \cup Opt_\ell$ is connected since otherwise we deal with connected components separately. Therefore, $Opt_w \cup Opt_\ell$ is either a cycle or a path which alternates edges of Opt_w and Opt_ℓ and covers all vertices.

Our formulation aims at identifying an instance $I \in \mathcal{I}_n$ minimizing a variable ρ such that for all matchings $M \in \mathcal{M}_n$ we have $w(M) \leq \rho$ or $\ell(M) \leq \rho$. Let ρ_n^* be the optimal value for variable ρ. Therefore, for any $\rho < \rho_n^*$, for any instance $I \in \mathcal{I}_n$, there exists $M \in \mathcal{M}_n$ such that $w(M) > \rho$ and $\ell(M) > \rho$. Moreover, ρ_n^*, corresponding to a feasible solution, is such that $w(M) \leq \rho_n^*$ or $\ell(M) \leq \rho_n^*$ for all matchings $M \in \mathcal{M}_n$. It follows that ρ_n^* is the largest value such that, for any instance $I \in \mathcal{I}_n$, there exists $M \in \mathcal{M}_n$ such that $w(M) \geq \rho_n^*$ and $\ell(M) \geq \rho_n^*$ (with at least one equality).

Owing to the quickly growing cardinality of $\mathcal{M}_n$, we were able to solve this mathematical program for n up to 10, using the solver CPLEX. For all $n \leq 10$, we found that $\rho^* \geq \frac{1}{2}$ (and $\rho^* = \frac{1}{2}$ for $n \in \{3, 4, 5, 8\}$). These computational results cannot be considered as totally valid owing to the limited precision of the computations (the values manipulated by CPLEX have a limited precision). They suggest, however, the existence of $\frac{1}{2}$-approximate matchings for complete graphs satisfying the triangle inequality, up to 10 nodes. Admitting this and using again Theorem 5.1, this improves the result about the approximation ratio of our algorithm from $\frac{1}{3}$ to $\frac{2}{5}$. Since we have shown that there does not exist $(\frac{1}{2} + \varepsilon)$-approximate algorithm for our problem [GMPV17], these results tend to show that the best approximation could be $\frac{1}{2}$. To confirm (or infirm) this conjecture is an open question.

Concluding words.

The ideal point approach is very interesting for problems for which there always exist solutions which are good simultaneously on all the objectives. It allows to return a solution which is good for both criteria without having to distinguish the agents by putting weight on them, and without consulting a decision maker, which is not always possible and suitable. Unfortunately, there are many problems for which there does not always exist solutions which are good approximation of the ideal point. This is often the case when there is no structure to the problem (for example for the bi-objective matching problems, there is no ρ-approximation, for any $\rho > 0$, if both labels do not fulfill the triangle inequality). It is also the case when the number of objectives grows (for example for the multi-objective matching problems, there is no ρ-approximation, for any $\rho > 0$, if there are 3 objectives which all fulfill the triangle inequality [GMPV17]).

For these problems for which the ideal point approach is not adapted, is useful to look at tools from social choice theory to see how to output a "good" solution common to all the agents.

5.2 Using tools from social choice theory

Social choice theory is the study of collective decision processes. It gathers models and results concerning the aggregation of individual inputs (e.g., votes or preferences) into collective outputs (e.g., collective decisions, or preferences). Central questions are: how can a group of individuals choose a winning outcome (e.g., a winning alternative, or a collective ranking of alternatives) from a given set of alternatives? What are the properties of different voting systems? How can we rank different social alternatives in an order of social welfare?

Social choice theory has been developed in the 18th century by Adam Smith, Nicolas de Condorcet and Jean-Charles de Borda and in the 19th century by Charles Dodgson (also known as Lewis Carroll). It had a renewed interest, and a great development, in the 20th century with the works of Kenneth Arrow [14], Duncan Black [24], Kenneth May [123], and Amartya Sen [158]. Two of the most famous results in social choice are the following ones. In these settings, the input is a set of alternatives (choices), and, for each individual (also called agent, or voter), his/her preferences as a ranking (total order) on the set of alternatives. An *aggregation function* outputs a ranking of alternatives, and a *voting rule* outputs one (winning) alternative.

- Kenneth Arrow's impossibility theorem [14] is generally acknowledged as the basis of the modern social choice theory. This theorem states that when voters have at least three distinct alternatives, no aggregation function can convert the ranked preferences of individuals into a collective ranking, while meeting simultaneously the three following criteria: Pareto efficiency (also called unanimity)[4], independence of irrelevant alternatives[5], and non-dictatorship[6].

- Another famous theorem is the one of Gibbard-Satterthwaite [86]: it states that when voters have at least three distinct alternatives, there is no voting rule which can convert the preferences of individuals into a selected alternative, while meeting simultaneously the three following criteria: non-dictatorship, surjectivity[7], and strategyproofness[8].

Social choice influence extends across economics, political science, philosophy, mathematics, and recently computer science, with the development of the *computational social choice* field[9].

5.2.1 Computational social choice

Computational social choice is an interdisciplinary field of study at the interface of social choice theory and computer science [43]. A few works at the boundary of the two fields [71, 100, 101] were precursors of this new field, formed in the early 2000s. It has two main directions.

Tools from computer science for social choice issues.
Computational social choice imports techniques developed in computer science to provide an analysis of social choice mechanisms, and to construct new mechanisms. Indeed, most of the work

[4]if every voter prefers alternative X over alternative Y, then in the ranking returned, X is preferred over Y.

[5]The relative order of two alternatives in the collective ranking only depends on the relative orders of these two alternatives in the individual preferences.

[6]The non-dictatorship property requires that the result of voting is not always the preferences of a distinguished voter, without consideration of the other voter's preferences.

[7]For each alternative, there exist a preference profile such that this alternative is the alternative selected by the voting rule

[8]This is the strategyproofness notion that we have seen on the previous chapter. This criterion is more commonly called *non-manipulability* in social choice. It means that no agent has incentive to declare false preferences.

[9]The interested reader can find an introduction to social choice in `https://plato.stanford.edu/entries/social-choice/`. For an overview of tools and methods for aggregating agents' preferences see the book of Arrow et al. [13].

in social choice theory consists in establishing abstract results regarding the existence (or non-existence) of procedures meeting certain requirements (Arrow's theorem is one example of such results). Computational issues, which are central in computer science, are usually not studied. An active field in computational social choice consists in studying complexity questions of known (and new) voting protocols (see e.g., [101, 75, 32]). It is good to have polynomial time voting protocols, as if it is NP-hard to find the selected alternative or ranking of an election, then this voting protocol may be difficult to use in practice (furthermore, the result of a voting protocol should be easily verifiable by voters if we want them to trust the voting procedure). Complexity may on the contrary be helpful in some other cases. While Gibbard-Satterthwaite's theorem shows that all "interesting" voting protocols are manipulable, computer scientists showed that for some voting protocols it is NP-hard for a voter to know which vote he or she has incentive to do. By using such voting protocols, the risk of manipulation from the voters should be reduced (see e.g, [100, 23]).

Other algorithmic issues, such as the design of resource allocation mechanisms, represent also active research fields where computer science adds its contribution to the social choice theory. Resource allocation mechanisms aim at assigning a finite set of items to a set of agents, given the preferences of agents over all possible sets of items. The criteria to measure the quality of a (centralized or distributed) protocol are efficiency criteria – among which the most fundamental criterion is Pareto efficiency –, and fairness criteria – among which the most studied criterion is *envy-freeness*[10]. An overview on resource allocation problems may be found in [26].

Tools from social choice for computer science issues.
Computational social choice also imports concepts from social choice theory into computational environments. This is for example the case for introducing negotiation and voting procedures for making joint decisions in multi-agent systems, which are populated by heterogeneous, possibly selfish, softwares [27]. Results from social choice are also used in group recommendation systems [27], and to develop page ranking systems for Internet search engine [4]. It is also the case in the collective schedules example that we will see in the next section, where we will introduce social choice tools for a classical scheduling problem (note, however, that the two directions discussed here :"tools from social choice for computer science issues", and the opposite, are not completely hermetic, and this problem could also be considered as extending the notion of ranking used in social choice).

An overview of the field of computational social choice can be found in [27].

5.2.2 An illustrative example: collective schedules

We propose in this section the notion of *collective schedule* that Krzysztof Rzadca, Piotr Skowron and I have introduced [PRS18]. In this problem, different agents have different preferences on common tasks, and the aim is to return a collective (compromise) schedule of the tasks. The motivation of this issue comes from the fact that public infrastructure projects, such as extending the city subway system, are often phased: all the planned phases (e.g. creation of new metro lines) are not built at the same time. Indeed, as workforce, machines and yearly budgets are limited, phases have to be developed one by one. Some phases are inherently longer-lasting than others. Moreover, individual citizens have different preferred orders of phases. Should the construction start with a long phase with a strong support, or with a less popular phase, that, however, will be finished faster? If the long phase starts first, the citizens supporting the short phase would have to wait significantly longer.

We formalize the collective schedule issue as follow. We are given a set of k alternatives (that we will call tasks) $\mathcal{T} = \{T_1, \ldots, T_k\}$, a duration (or length) p_i for each alternative (task) T_i, and a set of n agents (voters). Each agent wants all tasks to be completed as soon as possible, yet agents differ in their views on the relative importance of the tasks. The preference of an agent can be seen as a preferred schedule, where there is one machine and the tasks are scheduled in the order in which the agent would like them to be scheduled (thus there is no idle time between tasks in this schedule). We assume that each agent a has a certain preferred schedule σ_a, and

[10]An allocation is *envy-free* if no agent prefers the set of items given to an other agent to the one that she receives.

when building σ_a, an agent is aware of the lengths of the tasks. A collective schedule is a schedule of the tasks of $\mathcal{T}$ on one machine and without idle times between the tasks.

A schedule can be encoded as a (transitive, asymmetric) binary relation: $T_i \; \sigma_a \; T_j$ means that agent a prefers task T_i to be scheduled before task T_j. Thus, $T_1 \; \sigma_a \; T_2 \; \sigma_a \; \ldots \; \sigma_a \; T_k$ means that agent a wants T_1 to be processed first, T_2 second, and so on. Such a schedule can be denoted as $(T_1, T_2, \ldots, T_k)$. A *scheduling rule* is a function which takes a preference profile (vector of the preferred schedules of the agents) as an input and returns a collective schedule.

The difference with the other scheduling problems seen before in this manuscript, is that the tasks are common to all the agents. The aim is to find a good solution (a schedule of the tasks) given that the agents have different preferences (on the order in which the tasks should be scheduled). Tools from social choice ar e particularly adapted in this case.

If all the tasks have the same length, our problem consists in aggregating agents' preferences, one of the central problems studied in social choice theory [13]. However, the differences in the lengths of the tasks have to be explicitly considered. We focus here on three extensively studied tools from social choice theory: positional scoring rules, cost functions, and the Condorcet principle. We present them, and see how to extend them to the case where tasks have different lengths.

Positional scoring rules.

In social choice, positional scoring rules are perhaps the most straightforward, and the most commonly used in practice, tool to aggregate agents' preferences. Under a positional scoring rule, the score that an alternative receives from an agent is based only on the position of this alternative in the agent's preference ranking. For each alternative, the scores that it receives from all the agents are summed up, and the alternatives are ranked in the descending order of their total scores.

The most famous positional scoring rule is probably the one of Borda [61], where the first alternative ranked by an agent adds $k-1$ to the score of the alternative, the second alternative adds $k-2$, and so forth till the last alternative which adds 0.

There is a natural way to adapt this concept to the collective schedule issue. We define the score of a task T_i as the total duration of tasks scheduled after T_i in all preferred schedules (if the tasks are all of length 1, this scheduling rule is thus the Borda's rule). Despite this rule favors short tasks, it does not seem sufficient, as illustrated by the following example.

Example. [PRS18] Consider three tasks, $T_{\ell,1}, T_{\ell,2}, T_s$, with the lengths ℓ, ℓ, and 1, respectively. Assume that $\ell \gg 1$, and consider the following preferred schedules of agents:

$3n/8 + \epsilon$ of agents :	$T_{\ell,1}$	σ	$T_{\ell,2}$	σ	T_s
$3n/8 + \epsilon$ of agents :	$T_{\ell,2}$	σ	$T_{\ell,1}$	σ	T_s
$n/8 - \epsilon$ of agents :	T_s	σ	$T_{\ell,1}$	σ	$T_{\ell,2}$
$n/8 - \epsilon$ of agents :	T_s	σ	$T_{\ell,2}$	σ	$T_{\ell,1}$

In the returned schedule, $T_{\ell,1}$ and $T_{\ell,2}$ are scheduled before T_s. However, starting with T_s would delay $T_{\ell,1}$ and $T_{\ell,2}$ by only one, while starting with $T_{\ell,1}$ and $T_{\ell,2}$ delays T_s by 2ℓ, an arbitrarily large value. Moreover, T_s is put first by roughly $1/4$ of agents, a significant fraction.

Cost functions.

A cost function quantifies how a given ranking differs from an agent's preferred ranking. In our case, it quantifies how a given schedule differs from an agent's preferred schedule. The most famous cost function is perhaps the Kendall-Tau [108] (or swap) distance, $\mathbf{K}$, which measures the number of swaps of adjacent alternatives (tasks) to turn one ranking (schedule) into another one[11]. For a cost function f (for example $f = \mathbf{K}$) and an aggregation $\alpha \in \{\Sigma, \max, L_p\}$, by α-f we denote a (ranking or scheduling) rule returning a solution (ranking or schedule) that minimizes the α-aggregation of the f-costs of the agents. For example, Σ-$\mathbf{K}$ is simply the well-known Kemeny rule.

[11]An equivalent definition counts all pairs of alternatives executed in a non-preferred order.

Swap costs, as the Kendall-Tau distance, are not adapted when tasks have lengths. We define delay costs, based on the *completion times* $C_i(\sigma)$ of tasks in the preferred schedule σ (and thus, indirectly, on tasks' lengths). The completion times form tasks' due dates, $d_i = C_i(\sigma)$. A delay cost quantifies how far the completion times c_i in the proposed schedule are from their due dates d_i by one of the six classic criteria defined in Peter Brucker's reference book on scheduling problems [28]. The most used of these criteria is probably the Tardiness (**T**): $T(c_i, d_i) = \max(c_i - d_i, 0)$. We studied the complexity of finding the collective ranking with these criteria:

Theorem 5.2 ([PRS18]) *The problem of finding a collective schedule minimizing the total tardiness (Σ-T) is strongly NP-hard.*

This result can be shown with by a reduction from `3-Partition` where numbers are represented in unary encoding. Likewise, we have shown that max-T is NP-hard, even for unit size tasks. We did some experiments on data generated and from PrefLib[12] [122], and we saw that these methods are feasible if the size of the input is moderate, e.g. 20 tasks and 5 000 voters (these instances may represent realistic situations).

Scheduling based on cost functions avoids the problems exposed by the previous example (indeed for that instance, e.g., the Σ-T rule starts with the short task T_s). Additionally, $\sum$-f methods satisfy some naturally-appealing axiomatic properties, such as reinforcement[13]. This is however not the case of Pareto efficiency: a scheduling rule $\mathcal{R}$ satisfies *Pareto efficiency* if all agents prefer T_k to be scheduled before T_ℓ, then in the collective schedule T_k should be before T_ℓ. If all tasks are unit-size, the scheduling rule $\sum$-T is Pareto efficient. If tasks are not unit size however, there is no algorithm which is good for max-T or Σ-T and which satisfies Pareto efficiency. Indeed, we have shown [PRS18] that for any $\alpha > 1$, there is no scheduling rule that satisfies Pareto efficiency and is α-approximate for max-T or Σ-T.

Pareto efficiency is one of the most fundamental properties in social choice. However, sometimes (especially in our setting) there exist reasons for violating it. Indeed, even if all the agents agree that T_x should be scheduled before T_y, the preferences of the agents with respect to other tasks might differ. Breaking Pareto efficiency can help to achieve a compromise with respect to these other tasks. Nevertheless, this motivated us to formulate alternative scheduling rules based on axiomatic properties. We choose the Condorcet principle, a classic social choice property that is stronger than Pareto efficiency. This property can be adapted to consider the lengths of the tasks.

The Condorcet principle.

The *Condorcet principle* states that if there exists an alternative that is preferred to any other alternative by the majority of agents, then this alternative should be put on the top of the aggregated ranking. It can be generalized to the remaining ranking positions. Assume that the graph of the preferences of the majority of agents is acyclic, i.e., there exists no sequence of alternatives $o_1, \ldots, o_\ell$ such that o_1 is preferred by the majority of agents to o_2, o_2 to o_3, ..., $o_{\ell-1}$ to o_ℓ and o_ℓ to o_1. Whenever an alternative a is preferred by the majority of agents to another alternative b, then a should be put before b in the aggregated ranking.

Definition 5.1 (Lengths aware (LA) Condorcet principle) *A schedule σ is LA Condorcet consistent with a preference profile if for each two tasks T_k and T_ℓ, it holds that $T_k \ \sigma \ T_\ell$ whenever at least $\frac{p_k}{p_k+p_\ell} \cdot n$ agents put T_k before T_ℓ in their preferred schedule. A scheduling rule $\mathcal{R}$ satisfies the LA Condorcet principle if fo r each preference profile it returns a LA Condorcet consistent schedule, whenever such a schedule exists.*

The value of the ratio $\frac{p_k}{p_k+p_\ell}$ can be explained as follows. Consider a schedule σ and two tasks, T_k and T_ℓ, scheduled consecutively in σ. If we swapped T_k and T_ℓ in σ, the level of dissatisfaction of an agent who prefers T_k to T_l can be expressed as p_l (her preferred task is delayed by p_l) The ratio $\frac{p_k}{p_k+p_\ell}$ assures that the average level of dissatisfaction of an agent is minimized when T_k is scheduled before T_l.

[12]PrefLib (www.preflib.org) is a reference library of preference data and links. It houses over 3 000 datasets which can be used by the research communities that deal with preferences, including computational social choice, recommender systems, data mining, machine learning, and combinatorial optimization [122].

[13]A scheduling rule $\mathcal{R}$ satisfies *reinforcement* if for any two groups of agents N_1 and N_2, when a schedule σ is selected by $\mathcal{R}$ both for N_1 and for N_2, then it should be also selected for the joint instance $N_1 \cup N_2$.

There exist several polynomial time Condorcet consistent election rules [117, 57] that can be extended to tasks with varying lengths [PRS18].

5.2.3 Other questions and concluding words.

The collective schedule problem that we have described in this section is a first step towards unifying the scheduling field and social choice theory. It is natural to consider more complex scheduling models in the context of collective scheduling. Extensions could be for example: processing several tasks simultaneously – multiple machines with sequential or parallel tasks –, tasks with different release dates, or dependencies between tasks. Each of these extensions raises new questions on computability/approximability of collective schedules. Another interesting direction is to derive desired properties of collective schedules (distinct from LA-Condorcet), and then formulate scheduling algorithms satisfying them.

In participatory budgeting [29, 139, 74, 21] agents express preferences over projects which have different costs. The goal is to choose a socially-optimal set of items with a total cost which does not exceed a given budget. Thus, in a way, participatory budgeting extends the knapsack problem similarly to how we extend scheduling. Following this approach, we could tackle other classical problems studied in computer science. For example, we could think about computing a collective matching (several agents having preferences on the matchings). Another issue would be to compute a collective Hamiltonian path (agents have to go together from a given point to another, and each agent has her own preferences on the paths taken for this trip – for example each agent has a preferred path).

When the preferences of the agents can be expressed as preferences on each of the resources (e.g. each edge has a value for each agent in a graph, for graphs problems; or each object has a value for each agent, for the knapsack problem), classical computer science problems have usually been studied as multi-objective problems. However, since the aim in the works in this field is usually to compute the Pareto set of solutions (or an approximate set of Pareto optimal solutions), these works are adapted in the case where a decision maker selects himself or herself which solution to use in the returned set. This is suitable when the objectives are antagonist objectives of a single agent, which chooses himself or herself the solution he or she prefers among Pareto optimal solutions. This approach does not seem suitable however when the objectives concern different agents. Indeed, in this case a decision maker should have a rule to select one single solution among these Pareto optimal solutions, which are not as good for all the agents. The aim is rather to design an algorithm which outputs a single good collective solution. In this case, the ideal point approach can be used when there are good approximation of the ideal point, i.e. solutions of good compromises. As we have seen, this is far from being always the case, and other approaches should be used when the objectives are antagonist. Moreover, the ideal point approach optimizes the worst approximation ratio on an objective. This can lead, in certain situations, to a solution which is on the worst objective only a little bit better than another solution which is, in contrast much better on the other objectives. To circumvent this issue, using an Ordered Weighted Average (OWA) aggregator on the approximation ratio of the objectives could be interesting.

When the ideal point approach is not adapted, we should use social choice notions to design good collective solutions. Classical computer science problems, when agents have different preferences over the feasible solutions (these preferences being expressed by values on the resources, or not), can have strong impact on the society, since the preferences of the agents can be preferences of citizens, looking for a collective solution. For example, participatory budgeting has been implemented in various practical situations[14]. Therefore, classical problems, extensively studied in computer science for one agent, should benefit from the use of tools from social choice theory when considering several agents. As it has been done in multi-objective optimization, classical combinatorial optimization problems could be revisited in the view of social choice tools, to obtain good solution for problems involving several agents. More generally, computer science and social choice theory benefit from each other, and decompartmentalizing these research area broadens each of these domains.

[14]For example, the *Participatory Budgeting Project* (https://www.participatorybudgeting.org) is a nonprofit organization that supports participatory budgeting processes in the US and in Canada. Their projects have gathered more than 400,000 people to directly decide how to spend 300 million in public funds in 29 cities.

Personal publications

[ABP04] Eric Angel, Evripidis Bampis, and Fanny Pascual. Traffic grooming in a passive star WDM network. In *SIROCCO 2004: Structural Information and Communication Complexity, 11th International Colloquium*, pages 1–12, 2004.

[ABP05] Eric Angel, Evripidis Bampis, and Fanny Pascual. Truthful algorithms for scheduling selfish tasks on parallel machines. In *WINE 2005: Internet and Network Economics, First International Workshop*, pages 698–707, 2005.

[ABP06a] Eric Angel, Evripidis Bampis, and Fanny Pascual. The price of approximate stability for scheduling selfish tasks on two links. In *Euro-Par 2006: Parallel Processing, 12th International Euro-Par Conference*, pages 157–166, 2006.

[ABP06b] Eric Angel, Evripidis Bampis, and Fanny Pascual. Truthful algorithms for scheduling selfish tasks on parallel machines. *Theoretical Computer Science*, 369(1-3):157–168, 2006.

[ABP08a] Eric Angel, Evripidis Bampis, and Fanny Pascual. An exponential (matching based) neighborhood for the vehicle routing problem. *J. Comb. Optim.*, 15(2):179–190, 2008.

[ABP08b] Eric Angel, Evripidis Bampis, and Fanny Pascual. How good are SPT schedules for fair optimality criteria. *Annals OR*, 159(1):53–64, 2008.

[ABP08c] Eric Angel, Evripidis Bampis, and Fanny Pascual. The impact of local policies on the quality of packet routing in paths, trees, and rings. *J. Scheduling*, 11(5):311–322, 2008.

[ABPT07] Eric Angel, Evripidis Bampis, Fanny Pascual, and Alex-Ariel Tchetgnia. On the truthfulness and the approximation for scheduling selfish tasks. In *SPAA 2007: Proceedings of the 19th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 196–197, 2007.

[ABPT09] Eric Angel, Evripidis Bampis, Fanny Pascual, and Alex-Ariel Tchetgnia. On truthfulness and approximation for scheduling selfish tasks. *J. Scheduling*, 12(5):437–445, 2009.

[ABPT16] Eric Angel, Evripidis Bampis, Fanny Pascual, and Nicolas Thibault. Truthfulness for the sum of weighted completion times. In *COCOON 2016: Computing and Combinatorics - 22nd International Conference*, pages 15–26, 2016.

[BCP15] Alexandre Blogowski, Philippe Chrétienne, and Fanny Pascual. Network sharing by two mobile operators: beyond competition, cooperation. *RAIRO - Operations Research*, 49(3):635–650, 2015.

[BGMP11] Cristina Bazgan, Laurent Gourvès, Jérôme Monnot, and Fanny Pascual. Single approximation for biobjective max TSP. In *WAOA 2011: Approximation and Online Algorithms - 9th International Workshop, WAOA 2011*, pages 49–62, 2011.

[BGMP13] Cristina Bazgan, Laurent Gourvès, Jérôme Monnot, and Fanny Pascual. Single approximation for the biobjective max TSP. *Theoretical Computer Science*, 478:41–50, 2013.

[CGP07] George Christodoulou, Laurent Gourvès, and Fanny Pascual. Scheduling selfish tasks: About the performance of truthful algorithms. In *COCOON 2007: Computing and Combinatorics, 13th Annual International Conference*, pages 187–197, 2007.

[CP15] Johanne Cohen and Fanny Pascual. Scheduling tasks from selfish multi-tasks agents. In *Euro-Par 2015: Parallel Processing - 21st International Conference on Parallel and Distributed Computing*, pages 183–195, 2015.

[DJPT07] Florian Diedrich, Klaus Jansen, Fanny Pascual, and Denis Trystram. Approximation algorithms for scheduling with reservations. In *HiPC 2007: High Performance Computing - HiPC 2007, 14th International Conference*, pages 297–307, 2007.

[DJPT10] Florian Diedrich, Klaus Jansen, Fanny Pascual, and Denis Trystram. Approximation algorithms for scheduling with reservations. *Algorithmica*, 58(2):391–404, 2010.

[DPRT11] Pierre-François Dutot, Fanny Pascual, Krzysztof Rzadca, and Denis Trystram. Approximation algorithms for the multiorganization scheduling problem. *IEEE Transactions on Parallel and Distributed Systems*, 22(11):1888–1895, 2011.

[EGNK$^+$11] Bruno Escoffier, Laurent Gourvès, Thang Nguyen Kim, Fanny Pascual, and Olivier Spanjaard. Strategy-proof mechanisms for facility location games with many facilities. In *ADT 2011: Algorithmic Decision Theory - Second International Conference*, pages 67–81, 2011.

[EMPS13] Bruno Escoffier, Jérôme Monnot, Fanny Pascual, and Olivier Spanjaard. Truthful many-to-many assignment with private weights. In *CIAC 2013: Algorithms and Complexity, 8th International Conference*, pages 209–220, 2013.

[FGM$^+$14] Diodato Ferraioli, Laurent Gourvès, Stefano Moretti, Fanny Pascual, and Olivier Spanjaard. Combinatorial optimization with competing agents. In Vangelis Th. Paschos, editor, *Paradigms of Combinatorial Optimization, 2nd Edition.* John Wiley & Sons, 2014.

[GMP08] Laurent Gourvès, Jérôme Monnot, and Fanny Pascual. Cooperation in multiorganization matching. In *WAOA 2008: Approximation and Online Algorithms, 6th International Workshop*, pages 78–91, 2008.

[GMP12] Laurent Gourvès, Jérôme Monnot, and Fanny Pascual. Cooperation in multiorganization matching. *Algorithmic Operations Research*, 7(2), 2012.

[GMPV17] Laurent Gourvès, Jérôme Monnot, Fanny Pascual, and Daniel Vanderpooten. Biobjective matchings with the triangle inequality. *Theoretical Computer Science*, 670:1–10, 2017.

[LMPZ07] Maria Liazi, Ioannis Milis, Fanny Pascual, and Vassilis Zissimopoulos. The densest k-subgraph problem on clique graphs. *J. Comb. Optim.*, 14(4):465–474, 2007.

[Pas06] Fanny Pascual. *Optimisation dans les réseaux : de l'approximation polynomiale à la théorie des jeux. (Optimization in networks: from polynomial time approximation to game theory).* PhD thesis, University of Évry Val d'Essonne, France, 2006.

[PKP18] Siao-Leu Phouratsamay, Safia Kedad-Sidhoum, and Fanny Pascual. Two-level lot-sizing with inventory bounds. *Discrete Optimization*, 30:1–19, 2018.

[PR15] Fanny Pascual and Krzysztof Rzadca. Partition with side effects. In *HiPC 2015: 22nd IEEE International Conference on High Performance Computing*, pages 295–304, 2015.

[PR17] Fanny Pascual and Krzysztof Rzadca. Optimizing egalitarian performance in the side-effects model of colocation for data center resource management. In *Euro-Par 2017: Parallel Processing - 23rd International Conference on Parallel and Distributed Computing*, pages 206–219, 2017.

[PR18] Fanny Pascual and Krzysztof Rzadca. Colocating tasks in data centers using a side-effects performance model. *European Journal of Operational Research*, 268(2):450–462, 2018.

[PR19] Fanny Pascual and Krzysztof Rzadca. Optimizing egalitarian performance when colocating tasks with types for cloud data center resource management. *IEEE Transactions on Parallel and Distributed Systems*, 2019.

[PRS18] Fanny Pascual, Krzysztof Rzadca, and Piotr Skowron. Collective schedules: Scheduling meets computational social choice. In *AAMAS 2018: International Conference on Autonomous Agents and Multiagent Systems*, 2018.

[PRT07] Fanny Pascual, Krzysztof Rzadca, and Denis Trystram. Cooperation in multi-organization scheduling. In *Euro-Par 2007: Parallel Processing - 13th International Euro-Par Conference, Rennes, France, August 28-31, 2007, Proceedings*, pages 224–233, 2007.

[PRT09] Fanny Pascual, Krzysztof Rzadca, and Denis Trystram. Cooperation in multi-organization scheduling. *Concurrency and Computation: Practice and Experience*, 21(7):905–921, 2009.

Other publications

[1] Fidaa Abed, José R. Correa, and Chien-Chung Huang. Optimal coordination mechanisms for multi-job scheduling games. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 13–24, 2014.

[2] Alessandro Agnetis, Jean-Charles Billaut, Stanislaw Gawiejnowicz, Dario Pacciarelli, and Ameur Soukhal. *Multiagent Scheduling: Models and Algorithms.* Springer, 2014.

[3] Noga Alon, Michal Feldman, Ariel D. Procaccia, and Moshe Tennenholtz. Strategyproof approximation mechanisms for location on networks. arXiv.org:0907.2049, 2009.

[4] Alon Altman and Moshe Tennenholtz. Ranking systems: the pagerank axioms. In *EC 2005: Proceedings 6th ACM Conference on Electronic Commerce*, pages 1–8, 2005.

[5] Nir Andelman, Michal Feldman, and Yishay Mansour. Strong price of anarchy. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 189–198, 2007.

[6] Eric Angel, Evripidis Bampis, and Aleksei V. Fishkin. A note on scheduling to meet two min-sum objectives. *Oper. Res. Lett.*, 35(1):69–73, 2007.

[7] Eric Angel, Evripidis Bampis, and Laurent Gourvès. Approximation results for a bicriteria job scheduling problem on a single machine without preemption. *Inf. Process. Lett.*, 94(1):19–27, 2005.

[8] Eric Angel, Evripidis Bampis, and Laurent Gourvès. Approximation algorithms for the bi-criteria weighted max-cut problem. *Discrete Applied Mathematics*, 154(12):1685–1692, 2006.

[9] Eric Angel, Evripidis Bampis, and Nicolas Thibault. Randomized truthful algorithms for scheduling selfish tasks on parallel machines. *Theor. Comput. Sci.*, 414(1):1–8, 2012.

[10] Elliot Anshelevich, Anirban Dasgupta, Jon M. Kleinberg, Éva Tardos, Tom Wexler, and Tim Roughgarden. The price of stability for network design with fair cost allocation. *SIAM J. Comput.*, 38(4):1602–1623, 2008.

[11] Aaron Archer and Robert Kleinberg. Truthful germs are contagious: a local to global characterization of truthfulness. In *Proceedings 9th ACM Conference on Electronic Commerce (EC-2008), Chicago, IL, USA, June 8-12, 2008*, pages 21–30, 2008.

[12] Aaron Archer and Éva Tardos. Truthful mechanisms for one-parameter agents. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 482–491, 2001.

[13] K. J. Arrow, A. Sen, and K. Suzumura, editors. *Handbook of Social Choice & Welfare*, volume 2. Elsevier, 2010.

[14] Kenneth Arrow. *Social Choice and Individual Values.* Wiley, 1951.

[15] Vincenzo Auletta, Roberto De Prisco, Paolo Penna, and Giuseppe Persiano. The power of verification for one-parameter agents. In *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, pages 171–182, 2004.

[16] Robert Aumann. Subjectivity and correlation in randomized strategies. *Journal of Mathematical Economics*, 1:67–96, 1974.

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

[18] Yossi Azar, Kamal Jain, and Vahab Mirrokni. (almost) optimal coordination mechanisms for unrelated machine scheduling. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, SODA 2008, pages 323–332. SIAM, 2008.

[19] Cristina Bazgan, Laurent Gourvès, and Jérôme Monnot. Approximation with a fixed number of solutions of some multiobjective maximization problems. *J. Discrete Algorithms*, 22:19–29, 2013.

[20] Richard Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[21] G. Benade, S. Nath, A. Procaccia, and N. Shah. Preference elicitation for participatory budgeting. In *AAAI 2017*, pages 376–382, 2017.

[22] André Berger, Vincenzo Bonifaci, Fabrizio Grandoni, and Guido Schäfer. Budgeted matching and budgeted matroid intersection via the gasoline puzzle. *Math. Program.*, 128(1-2):355–372, 2011.

[23] Nadja Betzler, Susanne Hemmann, and Rolf Niedermeier. A multivariate complexity analysis of determining possible winners given incomplete votes. In *IJCAI 2009: Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 53–58, 2009.

[24] Duncan Black. On the rationale of group decision-making. *Journal of Political Economy*, 56:23–24, 1948.

[25] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. Dynamic placement of virtual machines for managing SLA violations. In *IFIP/IEEE International Symposium on Integrated Network Management (IM), Proc.*, pages 119–128. IEEE, 2007.

[26] S. Bouveret, Y. Chevaleyre, and N. Maudet. Fair allocation of indivisible goods. In F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. D. Procaccia, editors, *Handbook of Computational Social Choice*, chapter 2. Cambridge University Press, 2016.

[27] Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, Ariel D. Procaccia, and Hervé Moulin. *Handbook of Computational Social Choice.* Cambridge University Press, 2016.

[28] Peter Brucker. *Scheduling Algorithms.* Springer, 2006.

[29] Y. Cabannes. Participatory budgeting: a significant contribution to participatory democracy. *Environment and Urbanization*, 16(1):27–46, 2004.

[30] Gérard P. Cachon. Supply chain coordination with contracts. In *Supply Chain Management: Design, Coordination and Operation*, volume 11 of *Handbooks in Operations Research and Management Science*, pages 227 – 339. Elsevier, 2003.

[31] Jianhua Cao, Mikael Andersson, Christian Nyberg, and Maria Kihl. Web server performance modeling using an M/G/1/K* PS queue. In *International Conference on Telecommunications (ICT), Proc.*, volume 2, pages 1501–1506. IEEE, 2003.

[32] I. Caragiannis, E. Hemaspaandra, and L. A. Hemaspaandra. Dodgson's rule and Young's rule. In F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. D. Procaccia, editors, *Handbook of Computational Social Choice*, chapter 2. Cambridge University Press, 2016.

[33] Ioannis Caragiannis. Efficient coordination mechanisms for unrelated machine scheduling. *Algorithmica*, 66(3):512–540, 2013.

[34] Ioannis Caragiannis, Edith Elkind, Mario Szegedy, and Lan Yu. Mechanism design: from partial to probabilistic verification. In *Proceedings of the 13th ACM Conference on Electronic Commerce, EC 2012, Valencia, Spain, June 4-8, 2012*, pages 266–283, 2012.

[35] Ioannis Caragiannis, Angelo Fanelli, Nick Gravin, and Alexander Skopalik. Efficient computation of approximate pure nash equilibria in congestion games. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 532–541, 2011.

[36] Ioannis Caragiannis, Angelo Fanelli, Nick Gravin, and Alexander Skopalik. Approximate pure nash equilibria in weighted congestion games: Existence, efficient computation, and structure. *ACM Trans. Economics and Comput.*, 3(1):2:1–2:32, 2015.

[37] Anirudh Chakravorty, Neelima Gupta, Neha Lawaria, Pankaj Kumar, and Yogish Sabharwal. Algorithms for the relaxed multiple-organization multiple-machine scheduling problem. In *20th Annual International Conference on High Performance Computing, HiPC 2013, Bengaluru (Bangalore), Karnataka, India, December 18-21, 2013*, pages 30–38, 2013.

[38] Georgios Chalkiadakis, Edith Elkind, and Michael Wooldridge. *Computational Aspects of Cooperative Game Theory (Synthesis Lectures on Artificial Intelligence and Machine Learning)*. Morgan & Claypool Publishers, 1st edition, 2011.

[39] A Charnes, WW Cooper, and RO Ferguson. Optimal estimation of executive compensation by linear programming. *Management Science*, 1(2):138–151, 1955.

[40] Ming Chen, Hui Zhang, Ya-Yunn Su, Xiaorui Wang, Guofei Jiang, and Kenji Yoshihira. Effective vm sizing in virtualized data centers. In *IFIP/IEEE International Symposium on Integrated Network Management (IM), Proc.*, pages 594–601. IEEE, 2011.

[41] Xi Chen and Xiaotie Deng. Settling the complexity of two-player nash equilibrium. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 261–272, 2006.

[42] Yue Cheng, Ali Anwar, and Xuejing Duan. Analyzing alibaba's co-located datacenter workloads. In *IEEE International Conference on Big Data, Big Data 2018*, pages 292–297, 2018.

[43] Yann Chevaleyre, Ulle Endriss, Jérôme Lang, and Nicolas Maudet. A short introduction to computational social choice. In *SOFSEM 2007: Theory and Practice of Computer Science, 33rd Conference on Current Trends in Theory and Practice of Computer Science*, pages 51–69, 2007.

[44] Steve Chien and Alistair Sinclair. Convergence to approximate nash equilibria in congestion games. *Games and Economic Behavior*, 71(2):315 – 327, 2011.

[45] George Christodoulou, Martin Gairing, Yiannis Giannakopoulos, and Paul G. Spirakis. The price of stability of weighted congestion games. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 150:1–150:16, 2018.

[46] George Christodoulou and Elias Koutsoupias. On the price of anarchy and stability of correlated equilibria of linear congestion games. In *Algorithms - ESA 2005, 13th Annual European Symposium, Palma de Mallorca, Spain, October 3-6, 2005, Proceedings*, pages 59–70, 2005.

[47] George Christodoulou, Elias Koutsoupias, and Akash Nanavati. Coordination mechanisms. In *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, pages 345–357, 2004.

[48] George Christodoulou, Elias Koutsoupias, and Akash Nanavati. Coordination mechanisms. *Theoretical Computer Science*, 410(36):3327–3336, August 2009.

[49] Vasek Chvatal. *Linear Programming.* Bedford Books, 2016.

[50] Johanne Cohen, Daniel Cordeiro, and Pedro Luis F. Raphael. Energy-aware multi-organization scheduling problem. In *Euro-Par 2014 Parallel Processing - 20th International Conference, Porto, Portugal, August 25-29, 2014. Proceedings*, pages 186–197, 2014.

[51] Johanne Cohen, Daniel Cordeiro, Denis Trystram, and Frédéric Wagner. Analysis of multi-organization scheduling algorithms. In *Euro-Par 2010 - Parallel Processing, 16th International Euro-Par Conference, Ischia, Italy, August 31 - September 3, 2010, Proceedings, Part II*, pages 367–379, 2010.

[52] Johanne Cohen, Daniel Cordeiro, Denis Trystram, and Frédéric Wagner. Coordination mechanisms for selfish multi-organization scheduling. In *18th International Conference on High Performance Computing, HiPC 2011, Bengaluru, India, December 18-21, 2011*, pages 1–9, 2011.

[53] Johanne Cohen, Daniel Cordeiro, Denis Trystram, and Frédéric Wagner. Multi-organization scheduling approximation algorithms. *Concurrency and Computation: Practice and Experience*, 23(17):2220–2234, 2011.

[54] Johanne Cohen, Christoph Dürr, and Thang Nguyen Kim. Non-clairvoyant scheduling games. *Theory of Computing Systems*, 49(1):3–23, July 2011.

[55] Richard Cole, José R. Correa, Vasilis Gkatzelis, Vahab Mirrokni, and Neil Olver. Inner product spaces for MinSum coordination mechanisms. In *STOC 2011*, pages 539–548. ACM, 2011.

[56] Yann Collette and Patrick Siarry. *Multiobjective Optimization.* Springer, 2004.

[57] J. Colomer. Ramon Llull: from 'Ars electionis' to social choice theory. *Social Choice and Welfare*, 40(2):317–328, 2013.

[58] Daniel Cordeiro, Pierre-François Dutot, Grégory Mounié, and Denis Trystram. Tight analysis of relaxed multi-organization scheduling algorithms. In *25th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2011, Anchorage, Alaska, USA, 16-20 May, 2011 - Conference Proceedings*, pages 1177–1186, 2011.

[59] José R. Correa and Maurice Queyranne. Efficiency of equilibria in restricted uniform machine scheduling with total weighted completion time as social cost. *Naval Research Logistics (NRL)*, 59(5):384–395, 2012.

[60] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 1st edition, 2015.

[61] Jean-Charles de Borda. *Mémoire sur les élections au scrutin.* Histoire de l'Académie Royale des Sciences, Paris, 1781. `http://gerardgreco.free.fr/IMG/pdf/MA_c_moire-Borda-1781.pdf`.

[62] Sheng Di, Derrick Kondo, and Franck Cappello. Characterizing and modeling cloud applications/jobs on a Google data center. *The Journal of Supercomputing*, 69(1):139–160, 2014.

[63] Sheng Di, Derrick Kondo, and C Wang. Optimization of composite cloud service processing with virtual machines. *IEEE Transactions on Computers*, 64:1755–1768, 2015.

[64] S. Dobzinski and S. Dughmi. On the power of randomization in algorithmic mechanism design. *SIAM Journal on Computing*, 42(6):2287–2304, 2013.

[65] Maximilian Drees, Matthias Feldotto, Sören Riechers, and Alexander Skopalik. On existence and properties of approximate pure nash equilibria in bandwidth allocation games. In *Algorithmic Game Theory - 8th International Symposium, SAGT 2015, Saarbrücken, Germany, September 28-30, 2015, Proceedings*, pages 178–189, 2015.

[66] Johann Dréo, Alain Petrowski, Éric Taillard, and Patrick Siarry. *Metaheuristics for Hard Optimization.* Springer, 2006.

[67] David Easley and Jon Kleinberg. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World.* Cambridge University Press, 2010.

[68] Matthias Ehrgott. *Multicriteria Optimization.* Springer, 2005.

[69] Bruno Escoffier, Laurent Gourvès, and Jérôme Monnot. Fair solutions for some multiagent optimization problems. *Autonomous Agents and Multi-Agent Systems*, 26(2):184–201, 2013.

[70] Hossein Esfandiari and Guy Kortsarz. A bounded-risk mechanism for the kidney exchange game. *Discrete Applied Mathematics*, 243:46 – 53, 2018.

[71] David Gale et Lloyd S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69:9–14, 1962.

[72] Eyal Even-Dar, Alex Kesselman, and Yishay Mansour. Convergence time to nash equilibrium in load balancing. *ACM Trans. Algorithms*, 3(3), August 2007.

[73] Eyal Even-Dar, Alexander Kesselman, and Yishay Mansour. Convergence time to nash equilibria. In *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, June 30 - July 4, 2003. Proceedings*, pages 502–513, 2003.

[74] B. Fain, A. Goel, and K. Munagala. The core of the participatory budgeting problem. In *WINE 2016*, pages 384–399, 2016.

[75] F. Fischer, O. Hudry, and R. Niedermeier. Weighted tournament solutions. In F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. D. Procaccia, editors, *Handbook of Computational Social Choice*, chapter 2. Cambridge University Press, 2016.

[76] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms.* Springer, 2011.

[77] Dimitris Fotakis, Piotr Krysta, and Carmine Ventre. Combinatorial auctions without money. *Algorithmica*, 77(3):756–785, 2017.

[78] Dimitris Fotakis and Christos Tzamos. Winner-imposing strategyproof mechanisms for multiple facility location games. In *Proc. 6th Workshop on Internet and Network Economics (WINE)*, pages 234–245, 2010.

[79] Dimitris Fotakis and Christos Tzamos. On the power of deterministic mechanisms for facility location games. *ACM Trans. Economics and Comput.*, 2(4):15:1–15:37, 2014.

[80] Dimitris Fotakis and Christos Tzamos. Strategyproof facility location for concave cost functions. *Algorithmica*, 76(1):143–167, 2016.

[81] Dimitris Fotakis, Christos Tzamos, and Manolis Zampetakis. Mechanism design with selective verification. In *Proceedings of the 2016 ACM Conference on Economics and Computation, EC '16, Maastricht, The Netherlands, July 24-28, 2016*, pages 771–788, 2016.

[82] Dimitris Fotakis and Emmanouil Zampetakis. Truthfulness flooded domains and the power of verification for mechanism design. *ACM Trans. Economics and Comput.*, 3(4):20:1–20:29, 2015.

[83] Panagiotis Garefalakis, Konstantinos Karanasos, Peter R Pietzuch, Arun Suresh, and Sriram Rao. Medea: scheduling of long running applications in shared production clusters. In *EuroSys*, pages 4–1, 2018.

[84] M. R. Garey and D. S. Johnson. " strong " np-completeness results: Motivation, examples, and implications. *J. ACM*, 25(3):499–508, July 1978.

[85] Joseph Geunes, H. Edwin Romeijn, and Wilco van den Heuvel. Improving the efficiency of decentralized supply chains with fixed ordering costs. *European Journal of Operational Research*, 252(3):815–828, 2016.

[86] Allan Gibbard. Manipulation of voting schemes: A general result. *Econometrica*, 41(4):587–601, 1973.

[87] Ashish Goel and Piotr Indyk. Stochastic load balancing and related problems. In *IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 579–586. IEEE, 1999.

[88] Laurent Gourvès, Jérôme Monnot, and Lydia Tlilane. Approximate tradeoffs on weighted labeled matroids. *Discrete Applied Mathematics*, In Press, 2014.

[89] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling : a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[90] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAP*, 17(2):416–429, 1969.

[91] Fabrizio Grandoni, R. Ravi, Mohit Singh, and Rico Zenklusen. New approaches to multi-objective optimization. *Math. Program.*, 146(1-2):525–554, 2014.

[92] Peter Hansen. Bicriterion path problems. *Multiple Criteria Decision Making Theory and Application*, 177, 1980.

[93] Christoph Hansknecht, Max Klimm, and Alexander Skopalik. Approximate pure nash equilibria in weighted congestion games. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, pages 242–257, 2014.

[94] Jason Hartline. Mechanism design and approximation. http://jasonhartline.com/MDnA/, 2016.

[95] Refael Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations Research*, 17(1):36–42, 1992.

[96] Birgit Heydenreich, Rudolf Müller, and Marc Uetz. Decentralization and mechanism design for online machine scheduling. In *Algorithm Theory - SWAT 2006, 10th ScandinavianWorkshop on Algorithm Theory*, pages 136–147, 2006.

[97] Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *J. ACM*, 34(1):144–162, 1987.

[98] Dorit S. Hochbaum and David B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM J. Comput.*, 17(3):539–551, 1988.

[99] Ruben Hoeksma and Marc Uetz. The price of anarchy for minsum related machine scheduling. In *WAOA*, number 7164 in LNCS, pages 261–273. Springer, 2012.

[100] John Bartholdi III, Craig A. Tovey, and Michael A. Trick. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6(3):227–241, 1989.

[101] John Bartholdi III, Craig A. Tovey, and Michael A. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165, 1989.

[102] Nicole Immorlica, Li (Erran) Li, Vahab S. Mirrokni, and Andreas S. Schulz. Coordination mechanisms for selfish scheduling. *Theoretical Computer Science*, 410(17):1589–1598, 2009.

[103] Pawel Janus and Krzysztof Rzadca. SLO-aware colocation of data center tasks based on instantaneous processor requirements. In *ACM SoCC Proc.*, pages 256–268. ACM, 2017.

[104] Ehud Kalai and Meir Smorodinsky. Other solutions to nash's bargaining problem. *Econometrica*, 43(3):513–518, 1975.

[105] Ehud Kalai and Meir Smorodinsky. Proportional solutions to bargaining situations: Intertemporal utility comparisons. *Econometrica*, 45(7):1623–1630, 1977.

[106] Melanie Kambadur, Tipp Moseley, Rick Hank, and Martha A Kim. Measuring interference between live datacenter applications. In *International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, page 51. IEEE, 2012.

[107] Alexis C. Kaporis and Paul Pavlos Spirakis. Stackelberg games: The price of optimum. In *Encyclopedia of Algorithms*, pages 2078–2083. Springer, 2016.

[108] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[109] Gunjan Khanna, Kirk Beaty, Gautam Kar, and Andrzej Kochut. Application performance management in virtualized server environments. In *Network Operations and Management Symposium, (NOMS), Proc.*, pages 373–381. IEEE, 2006.

[110] Suntae Kim, Eunji Hwang, Tae-Kyung Yoo, Jik-Soo Kim, Soonwook Hwang, and Young-Ri Choi. Platform and co-runner affinities for many-task applications in distributed computing platforms. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Proc.*, pages 667–676. IEEE CS, 2015.

[111] Thang Nguyen Kim. On (group) strategy-proof mechanisms without payment for facility location games. In *Proc. 6th Workshop on Internet and Network Economics (WINE)*, pages 531–538, 2010.

[112] Dalibor Klusácek and Hana Rudová. Multi-resource aware fairsharing for heterogeneous systems. In *Job Scheduling Strategies for Parallel Processing - 18th International Workshop, JSSPP 2014.*, pages 53–69, 2014.

[113] Konstantinos Kollias. Nonpreemptive coordination mechanisms for identical machines. *Theory of Computing Systems*, 53(3):424–440, 2013.

[114] Anshul Kothari, David C. Parkes, and Subhash Suri. Approximately-strategyproof and tractable multiunit auctions. *Decision Support Systems*, 39(1):105 – 121, 2005. The Fourth ACM Conference on Electronic Commerce.

[115] Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. In *STACS, Proc.*, pages 404–413. Springer, 1999.

[116] J. Lesca and Patrice Perny. Almost-truthful mechanisms for fair social choice functions. *Frontiers in Artificial Intelligence and Applications*, 242:522–527, 01 2012.

[117] J. Levin and B. Nalebuff. An introduction to vote-counting schemes. *Journal of Economic Perspectives*, 9(1):3–26, 1995.

[118] Pinyan Lu, Xiaorui Sun, Yajun Wang, and Zeyuan Allen Zhu. Asymptotically optimal strategy-proof mechanisms for two-facility games. In *ACM Conference on Electronic Commerce*, pages 315–324, 2010.

[119] Zoltán Ádám Mann. Resource optimization across the cloud stack. *IEEE Transactions on Parallel and Distributed Systems*, 29(1):169–182, 2018.

[120] Bodo Manthey. Deterministic algorithms for multi-criteria max-tsp. *Discrete Applied Mathematics*, 160(15):2277–2285, 2012.

[121] Bodo Manthey. On approximating multicriteria tsp. *ACM Transactions on Algorithms*, 8(2):17, 2012.

[122] Nicholas Mattei and Toby Walsh. Preflib: A library for preferences http://www.preflib.org. In *Algorithmic Decision Theory - Third International Conference (ADT 2013)*, pages 259–270, 2013.

[123] Kenneth O. May. A set of independent necessary and sufficient conditions for simple majority decision. *Econometrica*, 20(4):680–684, 1952.

[124] Aranyak Mehta and Vijay V. Vazirani. Randomized truthful auctions of digital goods are randomizations over truthful auctions. In *Proceedings 5th ACM Conference on Electronic Commerce (EC-2004)*, pages 120–124, 2004.

[125] Lili Mei, Minming Li, Deshi Ye, and Guochuan Zhang. Facility location games with distinct desires. *Discrete Applied Mathematics*, 264:148–160, 2019.

[126] Dov Monderer and Lloyd S.Shapley. Potential games. *Games and Economic Behavior*, 14(1):124–143, May 1996.

[127] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[128] Hervé Moulin. *Axioms of Cooperative Decision Making*. Cambridge University Press, 1991.

[129] John Nash. The bargaining problem. *Econometrica*, 18(2):155–162, 1950.

[130] John Nash. Non-Cooperative Games. *The Annals of Mathematics*, 54(2):286–295, September 1951.

[131] Noam Nisan. Introduction to mechanism design (for computer scientists). In Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani, editors, *Algorithmic Game Theory*, pages 209–241. Cambridge University Press, 2007.

[132] Noam Nisan and Amir Ronen. Algorithmic mechanism design. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing (STOC)*, pages 129–140, 1999.

[133] Fukuhito Ooshita, Tomoko Izumi, and Taisuke Izumi. A generalized multi-organization scheduling on unrelated parallel machines. In *2009 International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT 2009, Higashi Hiroshima, Japan, 8-11 December 2009*, pages 26–33, 2009.

[134] Fukuhito Ooshita, Tomoko Izumi, and Taisuke Izumi. The price of multi-organization constraint in unrelated parallel machine scheduling. *Parallel Processing Letters*, 22(2), 2012.

[135] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*, volume 1 of *MIT Press Books*. The MIT Press, June 1994.

[136] Christos Papadimitriou. The complexity of finding nash equilibria. In Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani, editors, *Algorithmic Game Theory*, pages 517–542. Cambridge University Press, 2007.

[137] Christos H. Papadimitriou and Tim Roughgarden. Computing correlated equilibria in multi-player games. *J. ACM*, 55(3):14:1–14:29, August 2008.

[138] Christos H. Papadimitriou and Mihalis Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 86–92, 2000.

[139] The participatory budgeting project. `www.participatorybudgeting.org/`, 2016.

[140] Andrej Podzimek, Lubomir Bulej, Lydia Y. Chen, Walter Binder, and Petr Tuma. Analyzing the impact of CPU pinning and partial CPU loads on performance and energy efficiency. In *IEEE/ACM CCGrid), Proc.*, pages 1–10. IEEE CS, 2015.

[141] Emmanouil Pountourakis and Guido Schäfer. Mechanisms for hiring a matroid base without money. In *Algorithmic Game Theory - 7th International Symposium, SAGT 2014, Haifa, Israel, September 30 - October 2, 2014. Proceedings*, pages 255–266, 2014.

[142] Ariel D. Procaccia and Moshe Tennenholtz. Approximate mechanism design without money. In *ACM Conference on Electronic Commerce*, pages 177–186, 2009.

[143] Tim Püschel, Guido Schryen, Diana Hristova, and Dirk Neumann. Revenue management for Cloud computing providers: Decision models for service admission control under non-probabilistic uncertainty. *European Journal of Operational Research*, 244(2):637–647, 2015.

[144] April Rasala, Clifford Stein, Eric Torng, and Patchrawat Uthaisombut. Existence theorems, lower bounds and algorithms for scheduling to meet two objectives. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2002)*, pages 723–731, 2002.

[145] R. Ravi, Madhav V. Marathe, S. S. Ravi, Daniel J. Rosenkrantz, and Harry B. Hunt III. Many birds with one stone: multi-objective approximation algorithms. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing (STOC 1993)*, pages 438–447, 1993.

[146] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *ACM Symposium on Cloud Computing, SOCC '12, San Jose, CA, USA, October 14-17, 2012*, page 7, 2012.

[147] Robert W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.

[148] Robert W. Rosenthal. A class of games possessing pure-strategy nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.

[149] Tim Roughgarden. Stackelberg scheduling strategies. *SIAM J. Comput.*, 33(2):332–350, 2004.

[150] Tim Roughgarden. Routing games. In Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani, editors, *Algorithmic Game Theory*, pages 459–481. Cambridge University Press, 2007.

[151] Tim Roughgarden and Éva Tardos. How bad is selfish routing? *J. ACM*, 49(2):236–259, 2002.

[152] Erik Saule and Denis Trystram. Multi-users scheduling in parallel systems. In *23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS*, pages 1–9, 2009.

[153] Andreas S. Schulz and Nicolás E. Stier Moses. On the performance of user equilibria in traffic networks. In *Proc. 14th ACM-SIAM Symp. on Discrete Algorithms (SODA'03)*, pages 86–87, 2003.

[154] James Schummer and Rakesh V. Vohra. Strategy-proof location on a network. *Journal of Economic Theory*, 104, 2001.

[155] James Schummer and Rakesh V. Vohra. Mechanism design without money. In Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani, editors, *Algorithmic Game Theory*, pages 243–266. Cambridge University Press, 2007.

[156] Petra Schuurman and Gerhard J Woeginger. Approximation schemes - a tutorial. In R.H. Möhring, C.N. Potts, A.S. Schulz, G.J. Woeginger, and L.A. Wolsey, editors, *Lectures on Scheduling*. L.A. Wolsey, to appear.

[157] Stefano Sebastio, Giorgio Gnecco, and Alberto Bemporad. Optimal distributed task scheduling in volunteer clouds. *Computers & Operations Research*, 81:231–246, 2017.

[158] Amartya Sen. *Collective Choice and Social Welfare*. Holden-Day, 1970.

[159] Lloyd Shapley. A value for n-person games. *Contributions to the Theory of Games. Annals of Mathematical Studies*, 28:307–317, 1953.

[160] Alexander Skopalik and Berthold Vöcking. Inapproximability of pure nash equilibria. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 355–364, 2008.

[161] Louis P. Slothouber. A model of web server performance. In *International World Wide Web Conference (WWW)*, 1996.

[162] Weijia Song, Zhen Xiao, Qi Chen, and Haipeng Luo. Adaptive resource provisioning for the cloud using online bin packing. *IEEE ToC*, 63(11):2647–2660, 2014.

[163] Cliff Stein and Joel Wein. On the existence of schedules that are near-optimal for both makespan and total weighted completion time. *Operations Research Letters*, 21(3):115 – 122, 1997.

[164] Joseph E. Stiglitz. The theory of "screening," education, and the distribution of income. *The American Economic Review*, 65(3):283–300, 1975.

[165] M. Stillwell, F. Vivien, and H. Casanova. Virtual machine resource allocation for service hosting on heterogeneous distributed platforms. In *IPDPS, Proc.*, pages 786–797. IEEE, 2012.

[166] Xueyan Tang, Yusen Li, Runtian Ren, and Wentong Cai. On first fit bin packing for online cloud server allocation. In *IPDPS, Proc.*, pages 323–332, 2016.

[167] Eva Tardos and Tom Wexler. Network formation games and the potential function method. In Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani, editors, *Algorithmic Game Theory*, pages 487–516. Cambridge University Press, 2007.

[168] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin, Heidelberg, 2001.

[169] Berthold Vöcking. Selfish load balancing. In Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani, editors, *Algorithmic Game Theory*, pages 517–542. Cambridge, 2007.

[170] Berthold Vöcking. Selfish load balancing. In Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani, editors, *Algorithmic Game Theory*, pages 517–542. Cambridge University Press, 2007.

[171] Harvey M. Wagner and Thomson M. Whitin. Dynamic version of the economic lot size model. *Management Science*, 5(1):89–96, 1958.

[172] Meng Wang, Xiaoqiao Meng, and Li Zhang. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *INFOCOM, Proc.*, pages 71–75. IEEE, 2011.

[173] Yunjing Xu, Zachary Musgrave, Brian Noble, and Michael Bailey. Bobtail: Avoiding long tails in the cloud. In *USENIX NSDI, Proc.*, pages 329–341, 2013.

[174] Hui Zhao, Jing Wang, Feng Liu, Quan Wang, Weizhan Zhang, and Qinghua Zheng. Power-aware and performance-guaranteed virtual machine placement in the cloud. *IEEE Transactions on Parallel and Distributed Systems*, 29(6):1385–1400, 2018.