UNIVERSITAT DE GIRONA
ESCOLA POLITÈCNICA SUPERIOR
Department of Electronics, Computer Engineering and Automatics

Research Work

Robust Combinatorial Auctions for Resource Allocation

Submitted by
Víctor Muñoz i Solà
to obtain:
Diploma of Advanced Studies

Supervisor:
Dr. Dídac Busquets i Font

Contents

1 Introduction **7**
- 1.1 Motivations . . . 7
- 1.2 Example application domains . . . 8
  - 1.2.1 Water quality improvement . . . 8
  - 1.2.2 Road transportation optimization . . . 10
- 1.3 Overview of the contributions . . . 11
- 1.4 Outline . . . 11

2 State of the Art **12**
- 2.1 CSP, COP and NP-Hard Problems . . . 12
- 2.2 Techniques for solving COPs . . . 14
  - 2.2.1 Global optimization . . . 15
  - 2.2.2 Local Optimization . . . 24
- 2.3 Auctions . . . 27
  - 2.3.1 Auctions Classification . . . 28
  - 2.3.2 Combinatorial Auctions . . . 29
  - 2.3.3 The Winner Determination Problem . . . 30
  - 2.3.4 Optimal Algorithms for Combinatorial Auctions . . . 31
  - 2.3.5 Summary of combinatorial auctions solvers . . . 35
- 2.4 Recurrent Auctions . . . 36
- 2.5 Robustness . . . 37
- 2.6 Conclusions . . . 38

3 Exploratory work **40**
- 3.1 Robustness in recurrent combinatorial auctions . . . 40
  - 3.1.1 Coordinating Schedules . . . 41
  - 3.1.2 Conflict Resolution with Recurrent Combinatorial Auctions 42
  - 3.1.3 Adding Robustness . . . 43
- 3.2 The Waste Water Treatment Problem . . . 47
  - 3.2.1 Implementation . . . 48
  - 3.2.2 Experimentation results . . . 49
- 3.3 Design of an efficient algorithm for the WDP of combinatorial auctions . . . 53
  - 3.3.1 Notation . . . 54

3.3.2 The Algorithm . . . 54
3.3.3 First phase: Preprocessing . . . 55
3.3.4 Second phase: Upper and Lower Bounding . . . 58
3.3.5 Third phase: Search . . . 59
3.3.6 Experiments . . . 60
3.3.7 Results . . . 62
3.4 Conclusions . . . 68

4 Thesis planning **70**
4.1 Summary . . . 70
4.1.1 Robustness mechanism . . . 70
4.1.2 Winner determination algorithm . . . 71
4.1.3 Case studies . . . 71
4.2 Schedule . . . 72

5 Contributions **74**
5.1 Articles in conferences . . . 74
5.2 Articles in conferences (submitted) . . . 75
5.3 Journals . . . 75
5.4 Awards . . . 76

Bibliography **77**

List of Figures

1.1 Water treatment system 9

2.1 Complexity Classes 14
2.2 Constraint optimization techniques 16
2.3 Backtracking (continuous arrows indicate branches, and discontinuous arrows represent backtracks). 18
2.4 Graphical representation of the mechanism used by the Simplex algorithm to find the optimal solution. 22
2.5 Graphical representation of an example solution using the Karmarkar's Algorithm. 23
2.6 Auctions classification 29
2.7 Partition into bins. 32
2.8 Branch on items (left) versus branch on bids (right) formulation. Figure extracted from [PCS06]. 35

3.1 Coordination system 41
3.2 Trust model. 45
3.3 Risk attitude function: (a) averse, (b) proclive. 46
3.4 Example of tasks with associated trust values. 46
3.5 Disobey probability function. 50
3.6 Examples of (a) dominated item (it_1) and (b) solution bid (b_1). . 55
3.7 Example of (a) dominated and (b) 2-dominated bids. In (a) b_1 dominates b_2, and in (b) b_3 is dominated by the union of b_1 and b_2. 56
3.8 Example of pseudo-dominated bid (b_1 is pseudo-dominated). . . . 56
3.9 Pseudo-code algorithm of lower bound function 57
3.10 Example of compatibility-dominated bid (b_2 is compatibility-dominated by b_1). 58
3.11 Pseudo-code algorithm of iCabro procedure 60
3.12 L1 Distribution. 63
3.13 L2 Distribution. 63
3.14 L3 Distribution. 63
3.15 L4 Distribution. 63
3.16 L5 Distribution. 64
3.17 L6 Distribution. 64
3.18 L7 Distribution. 64

3.19 Arbitrary Distribution. 64
3.20 Matching Distribution. 65
3.21 Paths Distribution. 65
3.22 Scheduling Distribution. 65
3.23 Transports Distribution. 65
3.24 Overall comparative. 67
3.25 Comparative over distributions. 67

4.1 Thesis time schedule. 72

List of Tables

2.1 Auction solvers comparative. 36

3.1 First environment (all industries behaving similarly), with coordination but without robustness. 50
3.2 First environment (all industries behaving similarly), with both coordination and robustness. 51
3.3 Second environment (one industry always disobeying), with coordination but without robustness. 52
3.4 Second environment (one industry always disobeying), with both coordination and robustness. 53
3.5 Solved (S) and unsolved ($\neg S$) problems before the timeout. . . . 68

Chapter 1

Introduction

1.1 Motivations

In Artificial Intelligence research, optimization problems appear in many areas conceptualized as Constraint Optimization Problems (COP) [Kum92]. This class of problems has produced a growing research interest in the computer science community because of its high difficulty (NP-Complete) and many real-world applications.

Many real life problems such as resource allocation, planning, scheduling, routing and auctions can be formalized as COPs. These problems have been widely studied for many years and by a lot of researchers. Very efficient algorithms have been developed for solving some of these problems (by exploiting their characteristics). Moreover, constraint programming is a technology that provides huge economical outcomes, since it can really improve production processes, logistics, etc.

There are a lot of techniques for solving optimization problems, such as Backtracking, Branch & Bound, Genetic Algorithms, Linear Programming, etc. Furthermore, about any constraint optimization problem can be modeled with any technique. Some of the techniques are able to find the optimal solution, that is, the solution fulfilling a set of constraints, while maximizing or minimizing a given objective function. However, this task is known to be NP-Complete [PS98] and therefore unfeasible with large and generic constraint optimization problems. On the other hand, the methods that focus on finding a solution quickly, usually cannot assure that the solution found is optimal neither compute how much time it will take to find it.

Combinatorial auctions have arisen in recent years to become one of the most studied constraint optimization modeling systems by researchers, and the problem of determining the winners of the auction, called the Winner Determination Problem (WDP) has been a prolific source of new and powerful algorithms [dVV03].

Although finding the optimal solution is the main goal of constraint opti-

mization solvers, sometimes it is not the best choice, since it could fail in case the environment changed (a machine breaking down, a process taking longer than expected, etc.). In such cases, it would be much better to have a *robust* solution that could still be applicable even if unexpected events occurred. This quality is desirable in many systems providing the ability to maintain functionality even with unpredictable changes on the environment. Obviously, the price of robustness is optimality [BS04], since usually a robust solution will be suboptimal. Therefore, there is a need of balancing the optimality and the robustness of the solutions.

Although robustness has already been addressed in the field of planning and scheduling, unfortunately there are not so many robustness techniques for combinatorial auctions, which take into account this desirable robustness of the solution found.

The objective of this research is to tackle constraint optimization problems as combinatorial auctions taking into account the robustness of the resulting solution.

1.2 Example application domains

In the following sections we present two different environments where constraint optimization methods together with robustness are desirable:

- Water quality improvement
- Transport optimization

Both of them can be modeled as a combinatorial auction. Later on Chapter 3 we will show how to do this modeling and what are the advantages of this approach.

1.2.1 Water quality improvement

The treatment of the waste water discharged from industries is vital to assure the quality of the river. For this purpose, the water coming from the sewage is treated in a waste water treatment plant (WWTP). Its job is to remove contaminants and produce an (up to a certain degree) clean waterstream that can be put back into the river. In order to ensure that the treatment process is correctly performed two conditions must hold:

- Keep the incoming water flow below the WWTP hydraulic capacity (that is, the total amount of water the plant can absorb at any instant in time); otherwise, the overflowed water goes directly to the river without receiving any treatment, increasing its contamination level.
- Keep the contamination level of the incoming water below the WWTP treatment capacity. The contamination level is defined by a set of quality variables (oxygen demand, nitrogen level, etc.). If the level of any of these

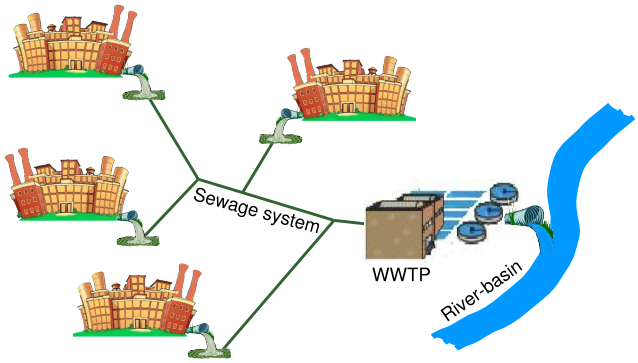

Figure 1.1: Water treatment system

variables is above the WWTP capacity, the water cannot be fully treated, and it increases the contamination of the river. Moreover, if the levels were too high, the micro-organisms used to treat the water may be damaged and the whole process could be stopped until these were regenerated. During this time, the plant could not accept any incoming water and it would be redirected to the river without any treatment.

The water entering the WWTP comes from three different sources: domestic use, rainfall and industries. Current regulations and legislations are in place so as to minimize the contaminating effects of industrial waste discharges. However this is not sufficient to guarantee the proper treatment of the water. The problem is that, although these regulations enforce industries to not contaminate more than some quantity, they do not take into account that simultaneous discharges by different industries may exceed these thresholds; for example if all the industries discharge at the same time, in such a case no industry would be breaking the rules, but the effect would be to have overflow or overcontaminated water going to the WWTP.

A typical water treatment system is depicted in Figure 1.1. The industries discharge their wastes to a sewage system, which directs the water to the WWTP. The plant, once the water has been treated, puts it back to the river. The main goal of the system is to ensure that the water flow entering the WWTP and its contamination levels are below some given thresholds, so that it can be correctly treated. It can be achieved by coordinating the discharges performed by the industries.

Through this coordination, the different discharges would be temporally distributed so that the WWTP is capable of processing all the incoming water. This would be beneficial at an environmental and health level, and is in the direction of having a more integrated management of the river-basin and all the involved systems.

The industries have some kind of working plan that allows them to foresee what discharges will be necessary in the near future, according to their produc-

tion process. This knowledge would permit the industries to inform the WWTP about the characteristics of their discharges (starting time, duration, flow and contaminants levels), so that coordination can be achieved.

Each industry has a tank where it can store its waste in case a discharge is not authorized. Obviously, if the industry is denied to discharge and its tank is full, it will be forced to realize the discharge anyway. As this situation can affect negatively the process in the WWTP, it should be avoided. It is assumed that an industry can perform two discharges at the same time: one coming from the production process, and another one coming from the retention tank.

The coordination mechanism tackles this problem by distributing the authorizations among the industries, trying to avoid that any of them has to perform unauthorized discharges.

A conflict arises when the set of discharges in a given instant exceeds the WWTP hydraulic capacity or the contamination levels. Once the discharges involved in a conflict are detected, the corresponding industry agents are informed about it, and the coordination process begins. The WWTP agent has to select a subset of the conflicting discharges that will be authorized, while some others will be asked to be delayed.

In this scenario, the robustness involves a resolution of the conflicts that produces solutions that are resistant to changes. These changes include modifications on the discharges times and flows of the industries, and also discharges of industries that are not allowed to do so. Changes can also occur in the WWTP, increasing or decreasing the hydraulic capacity or some of the components capacity, because of the micro-organisms variable life. And finally, changes can even arise with the weather variations, for example rain can create a sudden increase in the incoming flow entering the WWTP that will reduce its chances to treat industries discharges appropriately.

1.2.2 Road transportation optimization

In the road passenger transportation problem we are presented with a set of resources (drivers), and a set of tasks (services), to be performed using the resources. The problem consists in finding the best assignment of drivers to services given a cost function and subject to the constraints and preferences provided by administrations (local, national or European). In this constraint optimization problem (COP) we are trying to minimize the driver's costs, both in time and distance. The solution of the problem should be a complete allocation of the driver's activities that covers the complete set of services.

Each driver is characterized by a basic cost imposed by his contract, a cost per kilometer, a cost per time unit, a starting point and a final point (often the same), and the number of hours that he/she has driven during the last two weeks. The cost per kilometer refers to the cost of the vehicle associated to the driver.

A service consists in transporting passengers from a start location to a final location. Therefore, each service is characterized by the start location and the final location (where the service is), the start time and the final time (when).

There are several constraints regarding driving time which are quite complex in this transport domain. The problem is related to coach transportation and timetables are as strict as with other forms of transportation (train, buses); what is important is accomplishing the driving time regulated by law.

Here a robust assignation of services to drivers allows (up to a certain degree) variations in both the time schedules and driving constraints. It will turn into more relaxed drivers as they do not have so strict times, and less reschedules.

1.3 Overview of the contributions

The contributions of this work include the following:

- Study of the state of the art in both constraint optimization problems (focusing in auctions) and robustness.
- Modelization of some real-world constraint optimization problems using auctions.
- Development of robustness mechanisms to be applied in auctions.
- Development of a new algorithm for the Winner Determination Problem in combinatorial auctions.

1.4 Outline

This report is structured in five chapters as follows:

- Chapter 1 gives an introduction of this work and explains the two case examples (waste water treatment plant and road transportation problem) that have motivated this research.
- Chapter 2 discusses the related research on the areas where these kind of problems have been studied, mainly COP and auctions.
- Chapter 3 presents an exploratory work done in this area, composed by a robustness mechanism and a winner determination algorithm for combinatorial auctions.
- In Chapter 4 a proposal for the thesis together with a working plan are described.
- Finally, Chapter 5 shows the contributions of the work as publications in conferences and journals.

Chapter 2

State of the Art

This chapter surveys relevant research that has been done in the related areas. The first section discusses research in constraint optimization problems. After that, standard techniques for solving them are reviewed including both optimal and non-optimal methods. Next, auctions are overviewed and typified, followed by combinatorial and recurrent auctions. Finally, work made on robustness and particularly robustness in combinatorial auctions is addressed.

2.1 CSP, COP and NP-Hard Problems

Constraint satisfaction and optimization originated in the 1970s in the field of applied mathematics and computer science, related to operations research, artificial intelligence, software engineering, algorithm theory and computational complexity theory.

This kind of problems appear in many areas such as resource allocation, routing, planning and scheduling, and many other real-world and theoretical problems may be modeled using this generic framework. The objective of *Constraint Satisfaction* is to assign values to a set of decision variables such that the constraints (restrictions) are not violated. In *Constraint Optimization* the problem is extended, providing additionally a function of these variables to be maximized or minimized (while satisfying the constraints). A constraint optimization (maximization) problem is defined as [VJRS96, Bar99]:

$$\begin{aligned} &\text{maximize } f(x_1, ..., x_n) \\ &\text{subject to: } r_i, \quad i \in [1, m] \end{aligned} \tag{2.1}$$

where $X = \{x_1, ..., x_n\}$ is a set of *decision variables*, $R = \{r_1, ..., r_m\}$ is a set of *constraints* (or *restrictions*) limiting the values that the variables can simultaneously take and f is a general mathematical function. For each variable x_i some authors also define its *domain* D_i, which is a finite set of possible values. However, these domains can be easily converted to additional restrictions.

Mathematicians were the first to study Constraint Optimization Problems in the 1800s and early 1900s. Since then many researchers have been attracted for such problems due to its hardness to be solved in most cases, and have inverted much work trying to develop efficient methods. Unfortunately, nobody has been able to develop an "always-fast-and-optimal" algorithm for any of these problems, and that despite many decades of research effort invested on this subject by the most brilliant researchers worldwide. In fact, it seems to be the case that such problems are *inherently* difficult to solve, exhibiting an exponential growth in computing time when the size of the problem increases. The hypothesis that no efficient algorithm exists for solving these problems has been mathematically corroborated by advances in the field of computational complexity.

Complexity theory is the field of both mathematics and computer science (or theory of computation) concerned in classifying problems according to its intrinsic difficulty; dealing with the resources required during computation to solve a given problem, mainly execution time and space -memory- required. Many complexity classes exist, beginning with classes containing trivial problems (for example deciding if a number is, or is not multiple of 2) and ending with classes with problems that have been proved to be unsolvable or *undecidable* (for example the Halting problem [Tur36]).

The most cited complexity classes in complexity theory are P and NP, which separate respectively "easy" and "hard" problems. More precisely, P stands for *polynomial-time*, and problems are said to be in this class if they can be solved in a Deterministic Turing Machine [Tur36] using algorithms that require an execution time that is a polynomial function of the input's size. Alternatively, P is often taken to be the class of computational problems which are "tractable" or "efficiently solvable" by current computers. On the other hand, the class NP stands for "non-deterministic polynomial-time" and is the set of problems solvable in polynomial time on a Non-deterministic Turing Machine[1]. Practically, problems in NP become unfeasible because all known practical -deterministic- algorithms for them require an exponential amount of execution time to solve them.

An important notion in this context is the concept of *NP-Completeness*. The class of *NP-Complete* problems is a separate sub-class in NP and might be informally described as the "toughest" problems in NP in the sense that they are the ones most likely not to be in P. Figure 2.1 shows a diagram of all these complexity classes.

Formally, a problem is said to be *NP-Complete* if any problem in NP is polynomially transformable to it, and the problem itself belongs to NP. A problem is called *NP-Hard* if only the first condition holds. By definition, *NP-Complete* problems are *NP-Hard*, but also problems not contained in NP can be *NP-Hard*. All famous and classical combinatorial problems in its decisional version such as the Traveling Salesman Problem, Map Coloring Problem, Satisfiability, Graph

[1]Equivalently, the class NP can be defined as the set of problems whose solutions can be "verified" by a Deterministic Turing machine in polynomial time.

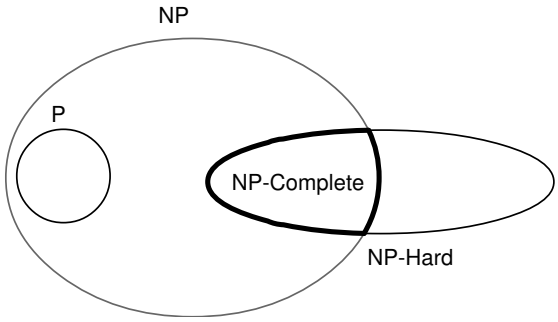

Figure 2.1: Complexity Classes

Partitioning, Vehicle Routing, Knapsack, Bin Packing, etc. belong to the *NP-Complete* class (see [CK95] and [Pap93] for a comprehensive list of *NP-Complete* problems).

All the *NP-Complete* problems can be mechanically transformed one to the other in polynomial time. Consequently, if there is a polynomial-time and deterministic algorithm for even one of them, then there is a polynomial-time algorithm for all the problems in *NP*, implying that *NP* is in fact equal to *P*. Because of this, and because dedicated research has failed to find a polynomial algorithm for any *NP-complete* problem, it is widely conjectured that no *NP-complete* problem is polynomially solvable deterministically[2] [Gas02]. Therefore, once a problem has been proved to be *NP-Complete* this is widely regarded as a sign that an efficient algorithm for this problem is unlikely to exist.

The interested reader is referred to classical texts on computational complexity by Papadimitriou and Steiglitz [PS98] and Garey and Johnson [GJ79] for a more formal and extensive description of these studies.

2.2 Techniques for solving COPs

Algorithms for solving COPs are generally faced with problems that are *NP-hard*. As mentioned in Section 2.1, such problems are not believed to be efficiently solvable in general. However, some approximations of complexity theory propose that some "small" instances of these problems can be solved efficiently. This is indeed the case, and such instances often lead to important practical ramifications.

Techniques used in constraint satisfaction and optimization problems depend on the kind of variables being considered. There exist two main types of

[2]Whether these problems are really undecidable in polynomial time is one of the greatest open questions in theoretical computer science. It is generally agreed to be one of the most important unsolved problems also in mathematics. In fact this problem, known as the P vs NP problem, is included in the Millennium Prize Problems proposed by the Clay Mathematics Institute, offering a $1 million US prize for the first correct answer to this question.

variables:

- Discrete variables.
- Continuous variables.

Problems that have to deal with variables on a discrete and finite domain (integer variables) are usually solved with search, in particular a form of backtracking or local search, trying to explore the habitually large search space. This search space is typically represented as a tree, or a graph, with the undesired feature of exponential growth with the input's size. For algorithms to be able to complete this search, methods for reducing the effective size of the space (pruning the tree), and procedures to explore efficiently the search space are applied.

Other considered kinds of variables are on continuous domains (real or rational numbers). Solving problems on these domains is usually done via variable elimination or Linear Programming using the simplex algorithm or other interior point methods.

A large number of methods proposed for solving optimization problems (including the majority of commercially available algorithms) are only concerned on finding a feasible solution quickly. But usually they are not capable to distinct between local optimal solutions and rigorous optimal solutions, treating the former as actual solutions to the problem. Research that focus on fast but non-optimal techniques is called *Local optimization*. Aside from this, the task of finding the absolutely best assignment of values to variables in order to achieve the optimal solution is called *Global Optimization*.

A diagram of the most common constraint optimization techniques divided by its category is shown in Figure 2.2. We next give an overview of these commonly used techniques.

2.2.1 Global optimization

Global optimization is the branch of applied mathematics and numerical analysis concerned with the development of deterministic algorithms capable of finding in finite time the real optimal solution of a problem with constraints, formulated in mathematical terms.

Some of the algorithms include the following:

- Generate and Test
- Uninformed Search
- Backtracking
- A*
- Branch & Bound
- Linear Programming

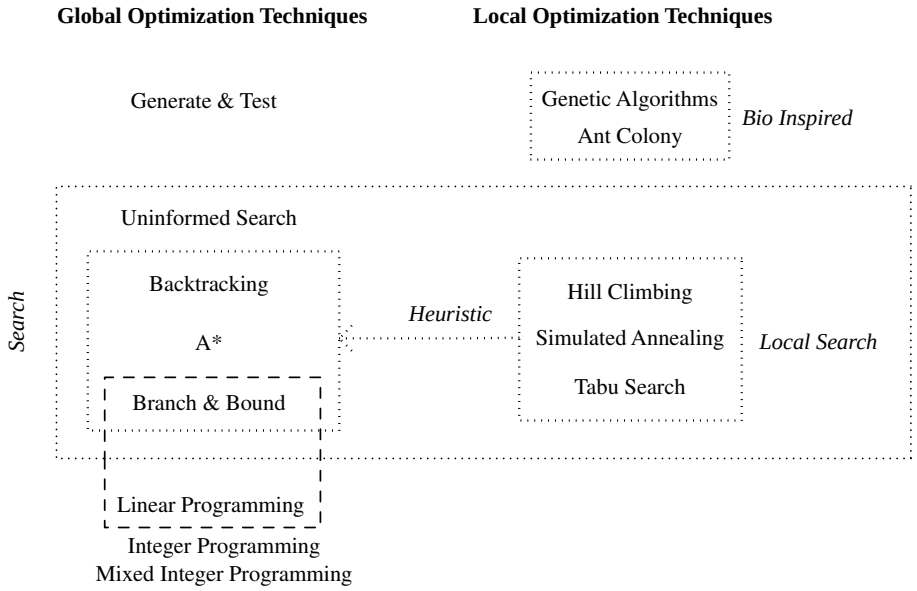

Figure 2.2: Constraint optimization techniques

Almost every global optimization technique can be modified to fit also in the local optimization section (for example stopping the execution at some point and returning the best current solution). Likewise, many of these optimal algorithms are able to utilize non-optimal techniques as an heuristic to explore efficiently the search space and achieve better anytime performance[3]. Therefore, albeit we will discuss separately global and local optimization techniques, they are actually quite connected.

Generate and Test

A first attempt to solve CSP and COP problems is using the generate-and-test paradigm, also known as *trial-and-error* or *brute force search*.

Solvers adhering to this paradigm use two basic modules.

- The *generator*. Enumerates possible solutions.
- The *tester*. Evaluates each proposed solution, either accepting or rejecting that solution.

The algorithm consists in creating a possible solution using the generator, applying it to the problem using the tester and, if it is not successful, generate subsequently the next possible solution. The process ends when a possibility yields a solution.

[3]Anytime performance is defined as the ability of an optimization algorithm to quickly generate good solutions.

Good generators combine these three properties:

- Complete: They eventually produce all possible solutions. Otherwise, the method is not complete and would turn into a local optimization technique.
- Non-redundant: They never compromise efficiency by proposing the same solutions twice.
- Informed: They use possibility-limiting information, restricting the proposed solutions accordingly.

With small or easy CSPs this is an easy-to-implement and fast algorithm, but in general this method will not be appropriate to find the solution in a reasonable amount of time with not-so-small instances. Furthermore, this method is only recommended for CSPs, given that with COPs (where the best solution has to be found) even though this paradigm is -rudimentarily- applicable, it is often useless, since all the possible candidates have to be tried, which is typically not feasible.

Uninformed Search

Uninformed search algorithms are exhaustive methods for systematically traversing or searching all the nodes in a tree or a graph to find a solution. They begin at the root node and continue expanding every node with some *strategy* until a solution is found. There exist mainly two strategies to explore the search space:

- Depth-first search (DFS).
- Breadth-first search (BFS).

BFS [THCS01] explores the search space in levels. It begins at the root node and expands all its child nodes. Then, for each of these nodes, it explores their unexplored child nodes, and so on, until it reaches the solution. Since all nodes discovered so far have to be stored, the space and time complexity of BFS is $O(|V|+|E|)$ where $|V|$ is the total number of nodes and $|E|$ the number of edges in the tree or graph. This method can be useful when the solution is located on the upper levels, but it is often impractical for harder problems due to its enormous demand for space.

In contrast, DFS [Pea84] explores the search space in branches. Starting at the root it goes as far as possible through each branch exploring continuously the first child node that appears and therefore going deeper and deeper until a solution is found, or until it reaches a node without children. Then the search returns to the most recent node it had not finished exploring. Time complexity is the same as BFS in the worst case but space complexity is much lower than BFS (as it only stores a single branch of the search tree).

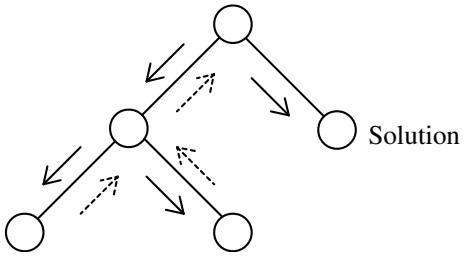

Figure 2.3: Backtracking (continuous arrows indicate branches, and discontinuous arrows represent backtracks).

Backtracking

Backtracking tis a generalization of recursive algorithms that performs a depth-first search with some improvements. Like DFS, all variables are initially unassigned and at each step, a variable is chosen, assigning all of its possible values to it in turn. To reduce the search size, for each attempted value, backtracking checks the correctness of the partial assignment, and only in case of consistency a recursive call is performed. When all possible values have been tried, the algorithm backtracks, i.e. continues with the previous variable. This consistency checking allows the algorithm to eliminate multiple solutions without being explicitly examined, with the saving of time that this implies.

Figure 2.3 shows a simple example with three variables to be assigned (three levels in the tree) and two possible values for each variable (two child nodes). The algorithm keeps on assigning values to variables until the third level, where it realizes that there is no possible solution this way. Then it backtracks to the previous variable and tries another value for it. The solution is neither in this branch, so the algorithms backtracks up to the first variable, to try the second value and finally find the solution.

In the basic backtracking algorithm, consistency is only concerned on satisfying all the constraints whose variables have been all assigned. Several variants of backtracking and consistency checking exists.

- *Backmarking* improves the efficiency of checking consistency by maintaining information about the last value assigned to a variable x_i and information about the values that caused inconsistency (i.e. violation of constraints).
- *Backjumping* allows going up more than one level in some cases when performing a backtrack.
- *Constraint learning* improves efficiency by inferring and storing new constraints whenever an inconsistency is found that can be later used to avoid part of the search.

- *Look-ahead* and *forward-checking* are sub-procedures that attempt to foresee the effects of an action (choosing a variable or a value to instantiate it).

Constraint propagation and *local consistency* techniques are methods to reduce variables' domain, therefore simplifying the problem. These techniques achieve this by checking the consistency of a subset of variables or constraints. This method in some kinds of problems and for some forms of constraint propagation may convert it to a triviality and a prove of either satisfiability or unsatisfiability would be straightforward. However, this is not guaranteed to happen in general. The most known and used forms of local consistency are:

- *Node consistency* requires the domain of every variable to be consistent with its unary constraints. For example, consider a variable x with a domain of $[0, 1, 2, 3]$ and a constraint $x > 1$, then its domain can be reduced to $[2, 3]$ and the constraint removed.

- *Arc consistency* enforces consistency between the domains of two variables connected with a binary constraint. For example, given two variables with a domain of $[0, 1, 2]$ and a constraint $a < b$. It is easy to see that value 2 can be removed from variable a's domain and value 0 from b's. The most popular arc consistency method is the *AC-3*[4] algorithm.

- *Path consistency* is a property similar to arc consistency, but considers a third variable that has to be consistent (path-consistent) with each consistent evaluation of the other two variables.

- *Hyper-arc consistency* or *generalized arc consistency* is a wider generalization of arc consistency, examining all the constraints (not only binary constraints) and assuring consistency in the domains of the variables involved.

There exist many other types of consistency such as *directional consistency*, *bucket elimination* and *relational consistency*, which carry out more processing in order to reduce even more the effective size of the space to be searched. However, although consistency techniques simplify problems, they generally do not produce solutions on their own.

A*

A* (pronounced "A star") was first described in 1968 by Peter Hart, Nils Nilsson, and Bertram Raphael. In their paper [HPE68], it was called algorithm A; applying to this algorithm an appropriate *heuristic* acquires optimal behavior, hence A*. This algorithm is a form of *best-first-search* as it expands the most promising node according to some rule, called an *heuristic*. The particularity

[4]The AC-3 (Arc Consistency Algorithm #3) algorithm, developed by Alan K. Mackworth in 1977 [Mac81], is one of the most used algorithms in the whole constraint propagation area.

of A* is that its heuristic also takes into account the distance already traveled ranking each node with an estimate of the best route that goes through that node.

To guarantee that the A* algorithm will find the optimal solutions, if one exists, the heuristic must be *admissible*, meaning that it must never *over-estimate* the distance from a state to the goal state. Formally:

$$f(x) \leq f(y) \qquad \forall y \in child(x)$$
$$f(x) = g(x) + h(x)$$

where $g(x)$ is the distance from the initial state to x, $h(x)$ is the estimation (heuristic) of getting from x to the goal node and, accordingly, $f(x)$ is the estimate of the best solution that goes through x.

A* is also *optimally efficient* for any heuristic h, in the sense that no algorithm employing the same heuristic will expand fewer nodes than A*, except when there are numerous partial solutions where h predicts the cost of the optimal path exactly.

Depth-first search and breadth-first search can be seen as two special cases of A* algorithm. Breadth-first search is the special case where $h(x) = 0\ \forall x$. To perform a Depth-first search, each time a new node is discovered we assign to $h(x)$ a value (beginning with a big value) that is subsequently decreased.

Time complexity of A* is highly dependent on the accuracy of the heuristic function. In the worst case, the algorithm runs in exponential time because it expands several nodes at each level, but it is able to reach polynomial time if the heuristic function h meets this condition [RN03]:

$$|h(x) - h^*(x)| \in O(\log h^*(x)) \tag{2.2}$$

where $h^*(x)$ is a *perfect* heuristic. In other words, the error of the used heuristic should not grow faster than the logarithm of an exact heuristic. However, since we are commonly facing *NP-Hard* problems, this condition is never met.

Branch & Bound

Branch and bound (B&B) is a rather general optimization technique to explore the search space partitioning recursively it, together with an smart procedure to cut the non-promising areas. The method was first proposed by A. H. Land and A. G. Doig in 1960 for integer programming [LD60].

Its design strategy is very similar to backtracking in that both use a state space tree to solve a problem. However, they differ in that the branch and bound method is not limited to an unique way of traversing the tree and it is used only for optimization problems. Its key features include:

- Branching: Divide the search space in separate regions.
- Bounding: Calculate bounds to reduce the search space.

- Pruning: Avoid exploring entire subregions.

The general idea of B&B approach is a Best-first-search for the optimal solution, except that not all nodes get expanded. Rather, a carefully selected criterion determines which node to expand and when, and another criterion notifies the algorithm when the optimal solution has been found. Criteria combine *lower-bounding* and *upper-bounding*. In maximization problems, a lower bound represents the minimum profit that a branch can provide (e.g. a feasible solution). On the other hand, an upper bound is an approximation of the best revenue that a branch can offer (possibly it will may not exist any solution with this income, as it is an optimistic estimation). For minimization problems, lower and upper bounds get interchanged. Another important tool of B&B is its effectiveness cutting called *pruning*. At any node of the tree it is checked whether the optimal solution can take place in any of its descendants. If the optimal solution can be proved to not be in a branch then the tree at that node can be "pruned". Note that it is always possible to find a feasible (sub-optimal) solution to a problem, and it can be used together with an upper bound (lower bound when minimizing) to discard entire regions of the tree.

The algorithm starts considering the original problem and the complete search space. Then it calculates a lower and an upper bound to the root problem. If they match, it means that the optimal solution has been found and the procedure terminates. Otherwise, the feasible search space is subdivided into many regions (ideally, splitting it into subregions), each one covering an strict subregion of the original, which together cover the whole feasible region. This is called *branching*, since this procedure is repeated recursively to each child, generating a tree of subproblems. If an optimal solution is found to a subproblem, it is actually a feasible solution to the full problem, although not necessarily globally optimal. However it can be used by the pruning mechanism. The search continues until all nodes have been either solved or pruned.

If the branches of the search tree can be pruned enough, we may be able to reduce the search space to a computationally manageable size. Note that solutions in the leaves of the pruned branches are not ignored, they are left out of consideration after we have made sure that the optimal solution cannot be at any one of these nodes. Therefore, the B&B approach is not an heuristic, or approximate, procedure, but it is an exact, optimal procedure.

Linear Programming

In mathematics, linear programming [Dan68] problems are defined as optimization problems consisting in an objective linear function and several equalities and inequalities (constraints). This subject dates back at least from Fourier (1768-1830), who created a -computationally expensive- method named *Fourier-Motzkin elimination* for solving a system of linear inequalities. However, it was in the 1940s where this discipline began to get important advances due to the models developed during the second world war to solve complex planning problems in wartime operations. Nevertheless, those mathematical models were kept

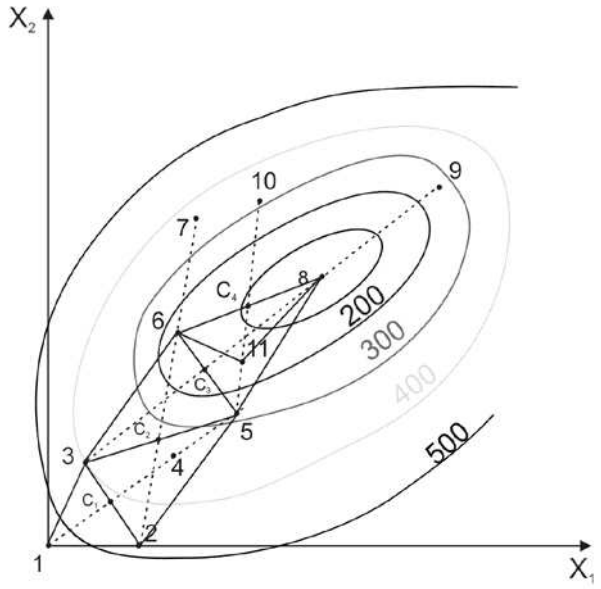

Figure 2.4: Graphical representation of the mechanism used by the Simplex algorithm to find the optimal solution.

in secret many years until they were made public postwar in 1947. Since then, its development accelerated enormously as many industries found valuable uses for linear programming.

The generally accepted founder of the subject is the mathematician George B. Dantzig, who created the *simplex* method in 1947 [Dan56], which is still now a popular method for solving linear programming models. However, Leonid Kantorovich, used similar techniques in economics before Dantzig (and won the Nobel prize in economics in 1975) [Kan59]. John von Neumann developed the theory of the duality, very important in linear programming, in 1947 [vN45]. The term "linear programming" was first proposed by T. J. Koopmans during a visit Dantzig made to the RAND corporation in 1948 to discuss his ideas.

Dantzig's simplex algorithm [Dan56] has been the standard technique for numerical solution of the linear programming problem since 1940's. In brief, the simplex method solves LP problems by constructing an admissible solution at a vertex of the polyhedron and then walking from vertex to vertex on the boundary of the feasible polyhedron with successively higher values of the objective function, until either an optimal solution is found, or it is established that no solution exists. Figure 2.4 shows a graphical representation of the simplex algorithm behavior.

In 1972, Klee and Minty gave an example [KM72] of a linear programming problem in which they showed that the simplex method as formulated by Dantzig visits all 2^n vertices before arriving at the optimal vertex. This

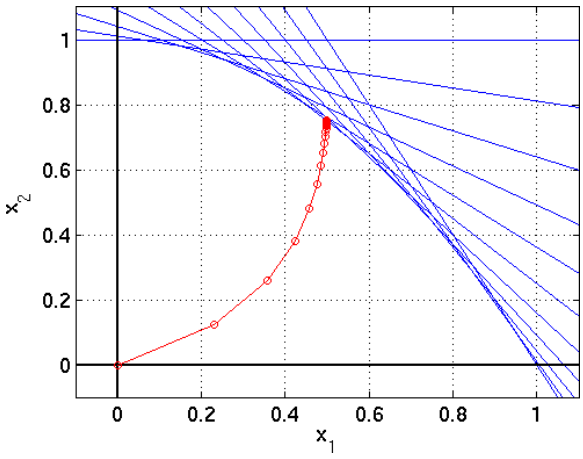

Figure 2.5: Graphical representation of an example solution using the Karmarkar's Algorithm.

shows that the complexity of the simplex algorithm in the worst-case is exponential time. However, in practice the method is remarkably efficient, typically requiring a number of steps which is just a small multiple of the number of variables.

In 1979 the Russian-born mathematician Leonid Khachiyan was the first to show that the linear programming problem is solvable in polynomial time. He presented the *Ellipsoid algorithm* [Kha79], guaranteed to solve any linear program in a number of steps which is a polynomial function of the amount of data defining the linear program. However, due to the high degree of the polynomial, in practice the simplex algorithm is far superior to the ellipsoid method. Even though, it has inspired other randomized algorithms for convex programming and is considered a significant theoretical breakthrough.

A larger major theoretical and practical progress in the field came in 1984 when Narendra Karmarkar introduced a new interior point method for solving linear programming problems, known as the Karmarkar's algorithm [Kar84], combining the desirable theoretical properties of the ellipsoid method and practical advantages of the simplex method. Its success initiated an explosion in the development of interior-point methods. These do not pass from vertex to vertex, but only through the interior of the feasible region as shown in Figure 2.5. Though this property is easy to state, the analysis of interior-point methods is a subtle subject which is much less easily understood than the behavior of the simplex method. Interior-point methods are now generally considered competitive with the simplex method in most, though not all, applications, and sophisticated software packages implementing them are now available. Whether they will ultimately replace the simplex method in industrial applications is not clear.

When in the linear program the unknown variables are all restricted to be integers instead of real, then this restricted version is called an *Integer Linear Programming problem.* Due to this, and because of the above algorithms are continuous by nature, they do not work so brilliantly with discrete variables. If

only some of the unknown variables are required to be integers, then the problem is called a *Mixed Integer Programming* problem. In contrast to linear programming, which can be solved efficiently, integer and mixed integer programming problems fall into the *NP-hard* class (not polynomially solvable), requiring additional search algorithms such as Branch & Bound. Binary programming is the special case of integer programming where variables are all required to be 1 or 0. This problem is also classified as *NP-hard*, in fact the decisional version of this problem was one of Karp's 21 *NP-complete* problems.

Concluding, Linear Programming in continuous domains is computationally tractable, either with an Interior Point method like the Karmarkar's algorithm which is guaranteed to run in polynomial time, or with the Simplex algorithm which has proved in practice to be a powerful mechanism solving highly efficiently almost every Linear Problem. However, when dealing with Integer Linear Programming this approach is deficient since the problems turns out to be non-polynomially solvable, needing the help of some other optimization techniques; rigorously, some kind of search.

2.2.2 Local Optimization

For *NP-Hard* problems it is in some cases not possible to find optimal solutions in a reasonable time, as optimal algorithms are only able to solve small problems exactly. For larger instances approximation algorithms must be applied. Finding only locally optimal solutions is considerably easier than finding the global optimum (of course depending on the degree of goodness of the desired solution). Some of the most used methods in local optimization include the following:

- Local Search
  - Hill Climbing
  - Simulated Annealing
  - Tabu Search
- Genetic algorithms
- Ant colony optimization

Local search techniques have been present in Artificial Intelligence for many decades, while Genetic algorithms and Ant Colony optimization methods are innovative ways to solve them creatively, besides achieving surprisingly good performance.

Local Search

Local search methods are nonsystematic, incomplete search algorithms. When running on CSPs they may find a solution of a problem, but they may fail even if the problem is satisfiable. With COPs they would find a "good" solution, but

unfortunately they will never know if the solution is optimal, neither know how far it is from the current one.

They work by iteratively improving a complete assignment over the variables. At each step, a small number of variables change their values, with the overall aim of increasing the goodness of this assignment. In practice, local search appears to work well when these changes are also affected by random choices. Interestingly, in many cases the quality of the solution can be arbitrarily good if an arbitrarily large number of repetitions is permitted.

Hill Climbing

Hill climbing is a rather simple local search strategy. It attempts to maximize (or minimize) a function $f(x)$, where x is an state of the search space. Hill climbing follows the graph from vertex to vertex, always trying to locally increase (or decrease) the value of *f*, until a local maximum x_m is reached.

In simple hill climbing, the first closer node is chosen, whereas in *steepest ascent hill climbing* all successors are compared and the closest to the solution is chosen.

Random-restart hill climbing is a meta-algorithm built on top of the hill climbing algorithm. It is also known as *Shotgun hill climbing*. Random-restart hill climbing simply runs an outer loop over hill climbing. Each step of the outer loop chooses a random initial condition x_0 to start hill climbing. The best x_m is kept: if a new run of hill climbing produces a better x_m than the stored one, it is replaced. Random-restart hill climbing is a surprisingly effective algorithm in many cases. It turns out that it is often better to spend CPU time exploring the space, rather than carefully optimizing from an initial condition.

Hill climbing together with backtracking turns into BFS, and therefore a global optimization technique.

Simulated Annealing

Simulated Annealing is another meta-algorithm built on top of the hill climbing algorithm. It was independently invented by S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi in 1983 [KGV83], and by V. Cerný in 1985 [Cer85]. It originated as a generalization of a Monte Carlo method for examining the equations of state and frozen states of n-body systems. The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects.

This method is similar to Hill Climbing in the way it finds a solution, but to overcome the local maxima problem, the process is iterated so that at each step of the Simulated Annealing algorithm the current solution is replaced by a random "nearby" solution, chosen with a probability that depends on the difference between the corresponding function values and on a global parameter T (called the temperature), that is gradually decreased during the process.

It can be shown that, for any given finite problem, the probability that the simulated annealing algorithm terminates with the global optimal solution

approaches 1 as the annealing schedule is extended. This theoretical result is, however, not particularly helpful, since the annealing time required to ensure a significant probability of success will usually exceed the time required for a complete search of the solution space.

Tabu Search

Tabu search is another local search algorithm, generally attributed to Fred Glover [Cer87]. It is similar to simulated annealing, in that both traverse the solution space by testing neighbors of the current solution. In order to prevent cycling, the Tabu search algorithm stores all visited solutions in a so-called "tabu list", in order to suppress moves to those solutions, and only accept neighbor solutions not contained in this list (i.e. not tabu solutions).

Due to memory restrictions it is not possible in general to store all visited solutions. Therefore, the list contains only solutions which have been visited in the last N iterations. Then only cycles with a length greater than the tabu list length may occur and if the tabu list is large enough, the probability of cycling becomes very small.

Different stopping criteria are possible. Usually, the whole tabu search procedure stops after a certain number of iterations, after a number of non-improving solutions, or when a given time-limit or solution fitness is reached.

Genetic Algorithms

The idea of evolutionary computing was introduced in the 1960s by Ingo Rechenberg in his work "Evolution strategies" [Rec60]. From then, many researchers where interested in this field. Genetic Algorithms (GA) as such were invented by John Holland and developed by him and his students and colleagues. This lead to Holland's book "Adaption in Natural and Artificial Systems" published in 1975 [Hol75].

A Genetic algorithm is a general search technique which mimics the biological evolution based on the principle "survival of the fittest". Genetic algorithms have been applied with growing success to combinatorial optimization problems.

Genetic algorithms are usually implemented as a computer simulation in which a population of abstract representations of candidate solutions (called individuals) to an optimization problem evolves toward better solutions. Traditionally, solutions are encoded as a sequence of symbols which is called a *chromosome*. Associated with an encoding s of a feasible solution is a measure of adaptation, the fitness value $fitness(s)$.

The evolution usually starts from an initial population of randomly generated individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected as "parents" from the current population based on their fitness, and new "child" solutions are generated by applying some modifications (crossover operators that recombinate or mix subsequences of the parent chromosomes, and mutation operators that perturbate a chromosome) to form a

new population, which is usually reduced to its original size by removing some solutions according to their fitness values. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached.

Ant Colony Optimization

The ant colony optimization algorithm (ACO), introduced by Marco Dorigo in 1992 in his thesis [Dor92], is a probabilistic technique for solving computational problems which can be reduced to finding good paths through graphs. It is inspired by the behavior of ants in finding paths from the colony to food.

Ant colony optimization algorithms have been used to produce near-optimal solutions to the traveling salesman problem. They have an advantage over simulated annealing and genetic algorithm approaches when the graph may change dynamically; the ant colony algorithm can be run continuously and adapt to changes in real time. This is of interest in network routing and urban transportation systems.

2.3 Auctions

So far we have reviewed several techniques to solve generic optimization problems. From now on, we will focus on an specific optimization problem: auctions.

Auctions are becoming popular in optimization problems where resources in a distributed system are shared in a competitive environment. Auctions provide useful mechanisms for resource allocation problems with autonomous and self-interested agents.

They were deeply studied first in economic theory as a way to establish prices in the market. Later they where also applied to game theory, and with the wide popularity of Internet and the emergence of electronic commerce, where auctions serve as the most popular mechanism, efficient auction design has become a subject of considerable importance for researchers in multi-agent systems.

Within the field of Artificial Intelligence there is a growing interest in using auction mechanisms to solve resource allocation problems in competitive multi-agent systems. For example, auctions and other market mechanisms are used in network bandwidth allocation, distributed configuration design, factory scheduling, and operating system memory allocation. Auctions are currently being used in several industrial scenarios ([BDHK06]), as the electricity market in which different kinds of energies are auctioned in order to favour the use of non pollutant sources of energy [PJ08].

An auction is a process where sellers offer objects, and the buyers notify their interest on them by submitting *bids*. The information contained in the bid is mainly the price that the buyer is ready to pay for the item. In the auction there

is usually one single auctioneer and a set of buyers. The auctioneer informs the buyers about the object/s that is/are going to be sold, then the bidders submit their bids and finally the auctioneer determines which are the winning bids to deliver the object to the corresponding buyers.

Now we survey and present a classification of the existing types of auctions, focusing and extending more deeply the combinatorial branch.

2.3.1 Auctions Classification

A diagram of the classification can be seen in Figure 2.6. Based on the number of items, auctions can be classified as single-item or multi-item auctions. The former are the most common where bidders compete for a good. There exist a lot of protocols for them, being the most common types English, Dutch, First Price Sealed Bid and Vickrey auctions.

In an *English auction* (also called an open-outcry auction) the auctioneer begins the auction with the reserve price (lowest acceptable price), then bidders are free to raise their bid higher than the actual price. When no more bids are risen the winner is the highest bidder. This is the typical auction used to sell artistic works. In a *Dutch auction* the auctioneer lowers the price until a bidder takes it. The first bidder to speak wins. This auction has been extensively used in fish markets. With *First Price Sealed Bid* each bidder submits a bid without knowing the other bidders' bids. The highest bid wins, paying his respective price. This differs from English auction because as bids are not open or called, bidders must submit valuations based upon supposed market value and their own willingness to pay, as opposed to engaging in competition through relative prices with other bidders. *Vickrey auction* [Vic61] (also known as *Second Price Sealed Bid* auction) is the same as first price sealed bid but here the winner pays the second highest price submitted.

When the quantity of items being sold is larger than one, auctions are called multi-unit. Single-item auctions with multi-unit items are differently classified based on pricing rules. For example, in a *Discriminatory Price Sealed Bid* (DPSB) auction, all the winners pay their bid price. Alternatively, in *Uniform Price Sealed Bid* (UPSB) auction, all winners pay the same price which is the highest bidding price of the losers. Finally, in *Generalized Vickrey Auction*, also called VCG[5], the price of a winner k is computed by deducting the sum of payments of all other bidders in the current resource allocation from the sum of all payments that would be obtained from those other bidders in the optimum allocation where the bidder k is removed from the allocation.

Multi-item auctions are known as Combinatorial Auctions. Here bidders can bid on bundles (combinations) of items. This enables the bidder to express dependencies and complementarities between goods. The auctioneer selects such a set of these combinatorial bids that results in the most revenue without assigning any object to more than one bidder. The computational complexity of the

[5]Where "V" stands for Vickrey [Vic61], "C" for Clarke [Cla71], and "G" for Groves [Gro73], three researchers who gave successively more general versions of the Vickrey auction.

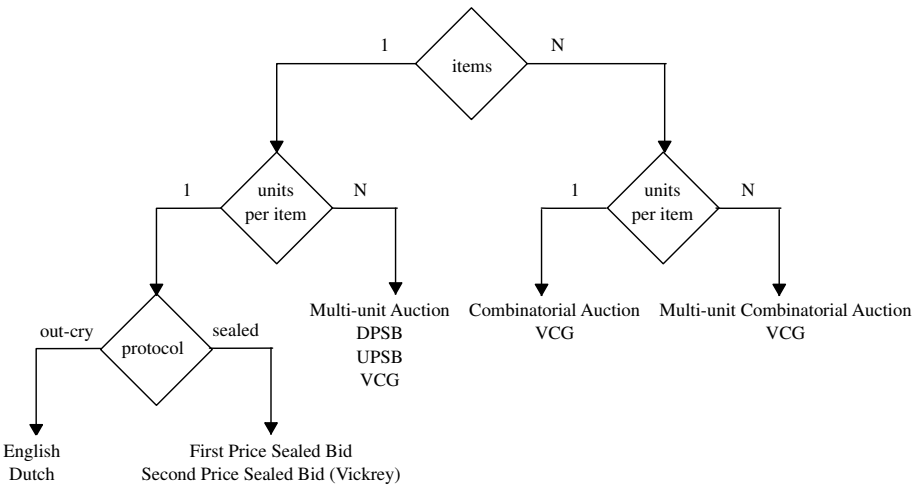

Figure 2.6: Auctions classification

optimal winner selection is very high compared with single-item auctions. Furthermore, we can also have multi-unit items in a combinatorial auction (known as Multi-unit Combinatorial Auction), making it even harder to solve. Clarke [Cla71] and Groves [Gro73] also extended the Vickrey concept to combinatorial auctions, although it is highly computationally intractable.

2.3.2 Combinatorial Auctions

The field of combinatorial auctions has grown rapidly in the past ten years. However, combinatorial auctions were first proposed by Rassenti, Smith, and Bulfin in 1982 [RB82], for the allocation of airport landing slots. This paper introduced many of the key ideas on combinatorial auctions, including the mathematical programming formulation of the auctioneers problem, the connection between the winner determination problem and the set packing problem, the issue of computational complexity, the use of techniques from experimental economics for testing combinatorial auctions, and consideration of issues of incentive compatibility and demand revelation in combinatorial auctions. The study of combinatorial auctions lies at the intersection of economics, operations research, and computer science.

Combinatorial auctions are those auctions in which bidders can place bids on combination (bundle, collection, package...) of items rather than just individual items. This allows bidders to express different types of dependency between goods [San02, FLBS99]:

- Substitutability: A player's value of getting for example two goods is less than the sum of its values for each individually (e.g., they are at least partially redundant). For example, a DVD reader and a DVD reader/writer are substitutable; a bidder may want one or another but not both.

- Complementarity: A player's value of getting for example two goods is greater than the sum of its values for each individually (e.g., they are at least partially co-dependent). For example, a bidder may value getting two shoes, but he probably does not want only one.

This expressiveness can lead to more economical allocations of these items because bidders do not get stuck with partial bundles of low value. The seller (auctioneer) is faced with the set of price offers for various bundles of goods, and his aim is to allocate the goods in a way that maximizes his revenue. However, designing good combinatorial auctions is much more challenging than designing good auctions for selling a single good. The computational complexity of determining the optimal winners of a combinatorial auction (maximizing auctioneers revenue), known as the *Winner Determination Problem* (WDP), is very high.

For a more extensive study of combinatorial auctions and the Winner Determination Problem, the interested reader is referred to the book by Peter Cramton, Yoav Shoham and Richard Steinberg [PCS06].

2.3.3 The Winner Determination Problem

The *Winner Determination Problem* WDP (also called the *auction clearing algorithm*) can be defined as: Given a set of bids in a combinatorial auction, find an allocation of items to bidders that maximizes the seller's revenue, subject to the constraint that each good can be allocated at most once.

The WDP is equivalent to weighted set-packing problem and the maximum weighted clique problem[6], and is therefore *NP-hard* even in its single-unit variant (see e.g., [RPH95]). Furthermore, it has been demonstrated that the WDP cannot even be approximated to a ratio of n^{1-e} (any constant factor) in polynomial time, unless $P = NP$ (see e.g., [San02]).

Formal Definition

Let $G = \{g_1, g_2, ..., g_m\}$ be a set of goods, and let $B = \{b_1, b_2, ..., b_n\}$ be a set of bids. Bid b_i is a pair ($price_i$, $goods_i$) where $price_i \in \mathbb{R}^+$ is the price offer of bid b_i and $goods_i \subseteq G$ is the set of goods requested by b_i. For each bid b_i define an indicator variable x_i that encodes the inclusion or exclusion of bid b_i from the allocation.

The single-unit WDP is the following constraint optimization problem:

$$\max \sum_{i=1}^{n} x_i \cdot price_i \tag{2.3}$$

$$\text{s.t.} \sum_{i|g \in goods_i} x_i \leq 1 \qquad \forall g \in G$$

[6]To model a combinatorial auction as a maximum weighted clique problem the problem has to be converted as a graph where nodes are bids and edges connect compatible bids, assigning the bids prices to the vertices weights.

In a multi-unit combinatorial auction instead of unique items we have a quantity $q(g)$ for each good, and the bids can request also different quantities of each item $q_{i,g}$. The WDP for multi-unit combinatorial auctions is the following:

$$\max \sum_{i=1}^{n} x_i \cdot price_i \tag{2.4}$$

$$\text{s.t.} \sum_{i|g \in goods_i} x_i \cdot q_{i,j} \leq q(g) \qquad \forall g \in G$$

The above problem formulation assumes the notion of *free disposal*. This means that the optimal solution need not necessarily sell all of the items. If the auction rules stipulate that all items must be sold, the problem becomes a Set Partition Problem [dVV03], which is *NP-Complete* as well.

If bids could be accepted partially, the problem would become a linear program which can be solved in polynomial time. This is absolutely not the case, but we keep this idea in mind since it is an interesting feature that some algorithms can smartly exploit.

2.3.4 Optimal Algorithms for Combinatorial Auctions

Since 1998 there has been a surge of research into the combinatorial auction winner determination problem. For a more extended survey, see [dVV03] and [PCS06]. We will brief some of the most known algorithms that provably find an optimal solution to the general problem where bids are not restricted. Since the problem is *NP-Hard*, any optimal algorithm for the problem will be slow on some problem instances. However, in practice, modern search algorithms can optimally solve winner determination in the large.

Now we describe some specific algorithms for solving combinatorial auctions (CA). Then we will give details on how to model combinatorial auctions with Integer Linear Programming to be run with a generic commercial LP solver like CPLEX, which has nowadays become the generally accepted solving method for CAs.

CASS

Combinatorial Auction Structured Search [FLBS99] is a branch and bound search algorithm with a befitting heuristic. It has been developed in the Stanford University by Yuzo Fujishima, Kevin Leyton-Brown and Yoav Shoham.

The crucial detail about CASS is that it structures the search space using *bins* (see Figure 2.7). A bin is created for each good, and every bid is placed into the bin corresponding to its lowest-order good. Rather than always trying to add each bid to the allocation, at most one bid from every bin is added since all bids in a given bin are mutually exclusive. Often entire bins can be skipped. However, the main benefit of bins is not the ability to avoid consideration of conflicting bids. Bins are powerful because they allow the pruning function

to consider context without significant computational cost, and allowing the generation of very fast and tight upper bounds.

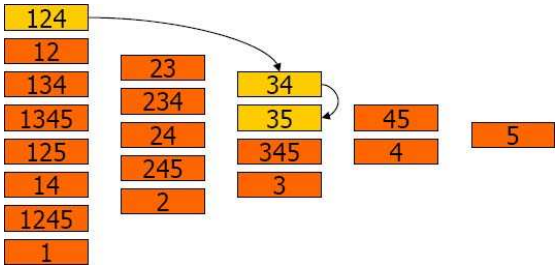

Figure 2.7: Partition into bins.

The search method is based on the branch on bids formulation. Each path in the search tree consists of a sequence of disjoint bids, that is, bids that do not share items with each other. A path ends when no bid can be added to it. As the search proceeds down a path, a tally, g, is kept of the sum of the prices of the bids accepted on the path. At every search node, the revenue g from the path is compared to the best g-value found so far in the search tree to determine whether the current path is the best solution so far. If so, it is stored as the new incumbent. Once the search completes, the incumbent is an optimal solution.

However, care has to be taken to treat the possibility that the auctioneer's revenue can increase by keeping items. The auctioneer's possibility of keeping items can be implemented by placing *dummy* bids of price zero on those items that received no 1-item bids.

A preprocessing step is performed in order to remove in polynomial-time *dominated* bids to not enter into the search process. For each pair of bids (b_1,b_2) where all items in b_1 are contained in b_2, if the price of b_1 is greater or equal to the price of b_2, b_2 may be removed from the bids list to explore, as b_2 is never preferable to b_1 (hence we say that b_1 *dominates* b_2).

CASS also caches the results of partial searches. This caching scheme is a form of dynamic programming that allows the algorithm to use experience from earlier in the search to tighten its upper bound function. In terms of computational complexity, it is easy to see that even in the worst case, the size of the explored tree is polynomial in the number of bids, but exponential in the number of items. However, CASS may be used as an anytime algorithm, as it tends to find good allocations quickly.

CASS is a free and open source algorithm that can be unrestrictedly downloaded from Kevin Leyton-Brown's web page[7].

[7]http://www.cs.ubc.ca/~kevinlb/downloads.html

BidTree

Bidtree [San02] is the other special-purpose WDP algorithm that has been most widely studied and cited in the literature. It was presented in the same conference proceedings as CASS. The Bidtree algorithm is similar to CASS in several ways, but important differences hold. In particular, Bidtree performs a secondary depth-first search to identify non-conflicting bids, whereas CASS's structured approach provides context to the upper bound function as well as allowing it to avoid considering most conflicting bids. Bidtree performs no caching or cache pruning. On the other hand, Bidtree uses an IDA* search strategy rather than CASS's branch-and-bound approach, and does more preprocessing.

The Bidtree algorithm has never been publicly available, neither to researchers. However the creators of CASS affirm that overall, CASS dramatically outperforms Bidtree, being between 2 and 500 times faster than Bidtree, and never slower.

Linear Programming

The Winner Determination Problem can be easily modeled as an Integer Programming Problem. To make so, bids are converted to binary variables X, and the function to be maximized f is the weighted sum of the bids multiplied by its price. Restrictions are constructed in order to assure that bids sharing an item cannot both win (their sum must be less or equal to 1).

$$\text{maximize f} = \sum_{i \in Bids} price_i \cdot X(i) \tag{2.5}$$

$$\forall j \in Items \sum_{i \in C_j} X(i) \leq 1 \tag{2.6}$$

where C_j is the set of bids containing item j. Note that the constraint is ≤ 1 instead of $= 1$ because an optimal allocation may leave some items unsold. If all the items are required to be sold then the equality condition should be set.

ILOG's CPLEX, is the most used LP optimization software worldwide. Universities and researchers have extensively used it to solve most of the COP's and every new algorithm or technique that comes out is habitually compared versus CPLEX.

When CASS and Bidtree were proposed, ILOG's CPLEX 5 mixed integer programming package (the industry standard) was unable to solve most WDP problems within a reasonable amount of time. Since that time, however, CPLEX's mixed integer programming module improved substantially with version 6 (released 2000), and considerably again with version 7 (released 2001). In version 8 (released 2002), with the MIP optimizer achieving an average 40% speed increase to optimality, with a 70% increase on difficult problems, there was a general convergence in the research community towards using CPLEX as the default approach for solving the WDP. Once again, CPLEX with version 9 (released 2003) improved the MIP optimizer to be 50% faster on average, for

a set of difficult customer models. Now that CPLEX is in version 10, which has improved the time to optimality by an average of 30% and improvements average 70% for particularly difficult models, the only drawback of this software against others' may be its elevated price.

Another possibility is to use the free GLPK (GNU Linear Programming Kit) solver. ILOG claims that its CPLEX solver is 100 times faster than GLPK; instead GLPK is free, and still a good choice for solving medium-size instances.

CABOB

When CPLEX began to release amazingly efficient LP and MIP solvers, researchers gradually switch to using CPLEX as the default approach for solving the WDP. The only ongoing effort at competition with CPLEX came from the authors of Bidtree, who wrote an updated algorithm called CABOB which they claim is much faster [SSGL01].

The CABOB (Combinatorial Auction Branch on Bids) algorithm is a depth first branch and bound search that branches on bids. The main difference is that instead of branching on items, CABOB uses the branch on bids formulation. A graphical representation of the search space generated with both formulations is shown in Figure 2.8. When branching on a bid, the children in the search tree are the world where that bid is accepted, and the world where that bid is rejected. The branching factor is 2 and the depth is at most n (number of bids). No dummy bids are needed: the items that are not allocated in bids on the search path are kept by the auctioneer. Given the branching factor and tree depth, a naive analysis shows that the number of leaves is at most 2^n. However, a deeper analysis establishes a drastically lower worst-case upper bound reaching a polynomial growth in bids, while exponential in items (the same as CASS).

The algorithm maintains a conflict graph structure called the bid graph. The nodes of the graph correspond to bids that are still available to be appended to the search path, that is, bids that do not include any items that have already been allocated. Two vertices in the graph share an edge whenever the corresponding bids share items. CABOB uses a technique for pruning across independent subproblems (components of the graph).

CABOB uses Linear programming for upper bounding. This usually leads to faster search times than any of the other special-purpose upper bounding methods proposed for winner determination. This is likely due to better bounding, better bid ordering, and the effect of the INTEGER special case, i.e an integer solution provided by the Linear Programming solver, implying that no more search is needed in the respective branch. The time taken to solve the linear program is greater than the per-node time with the other bounding methods, but the reduction in tree size usually amply compensates for that. However, on a non-negligible portion of instances the special-purpose bounding heuristics yield faster overall search time.

Like Bidtree, CABOB is neither available publicly. Its reported performance is apparently similar to CPLEX's, and as discussed above, CABOB is also similar to CPLEX in its construction: it makes use of CPLEX's linear programming

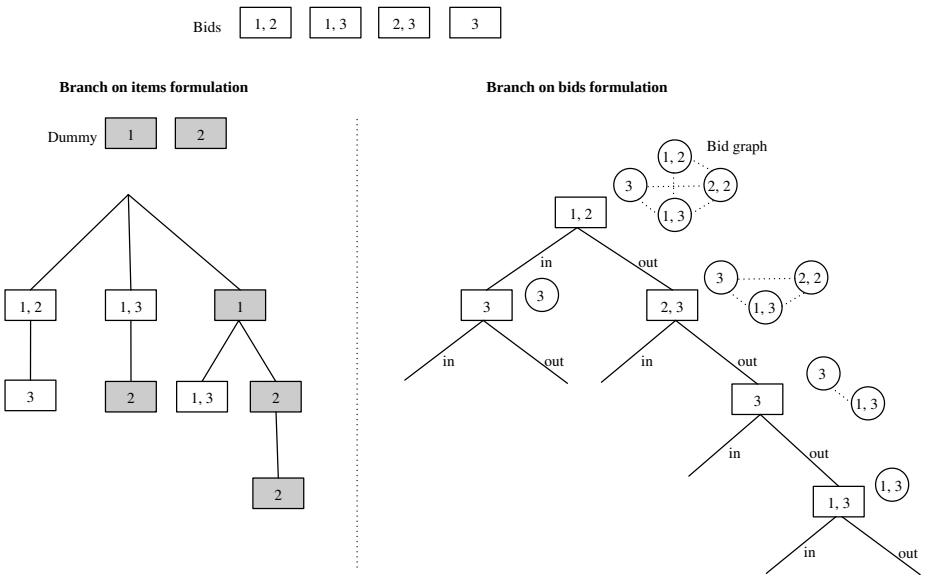

Figure 2.8: Branch on items (left) versus branch on bids (right) formulation. Figure extracted from [PCS06].

package as a subroutine and uses a similar search strategy.

2.3.5 Summary of combinatorial auctions solvers

We have described four different methods to solve a combinatorial auction. Table 2.1 shows a comparison of these methods, focusing on the following characteristics:

- Performance. How fast the algorithm ends giving the absolute optimal solution.
- Anytime performance. How fast the algorithm produces a valid solution.
- Input & output. This field describes whether the algorithm receives as input the list of bids directly or it needs some conversion.
- Preprocessing. How much preprocessing the algorithm executes.
- Economical cost. Price of the software providing the method.

Regarding the overall performance, CPLEX is clearly the best product followed by CABOB and GLPK, and at some distance CASS and finally BidTree. Instead, concerning anytime performance, CASS is the method requiring less amount of time to produce a first solution. This is for two reasons: firstly because it performs less preprocessing and secondly because LP-based algorithms

need to solve first the LP problem (which does not generally produce a valid solution) in order to begin the search of valid solutions. Therefore, although the first (non-optimal) proposed solution of CPLEX is probably much better than the CASS first solution, CASS obtains it earlier, so we state that CASS exhibits a better anytime performance.

CPLEX and GLPK need a transformation from the set of bids to a linear programming problem. This transformation requires a small amount of time (polynomial) compared to the total time of the execution. However, for small problems it may be faster to use a method that does not require any transformation. Of course, when dealing with huge problems, MIP solvers will be much faster since the transformation time would be insignificant compared to the improvement in overall execution time obtained.

Method	Perf.	Anytime	Input&Output	Preproc.	Econ. Cost
CASS	Slow	Good	Direct	Very Fast	Free
BidTree	Very Slow	Good*	Direct*	Fast*	Unavailable
CPLEX	Very Fast	Bad	Transformation	Fast	Expensive
GLPK	Fast	Bad	Transformation	Fast	Free
CABOB	Fast	Bad*	Direct*	Slow*	Unavailable

*Unknown (just presumed values).

Table 2.1: Auction solvers comparative.

Regarding the cost of the product, CPLEX is quite expensive, and CABOB is not available publicly, therefore unless we need to solve huge problems, GLPK and CASS would be interesting tools.

As a conclusion, CASS should be the first option to try, as it is free and easy to use (receiving as input directly the list of bids). If CASS is not able to solve the problems because of its large size, then a transformation to LP should be considered to test whether or not GLPK is able to solve it. Otherwise, CPLEX is the last resource if its cost can be afforded.

2.4 Recurrent Auctions

The previously presented solvers are only concerned in solving a concrete combinatorial auction. However, in some domains the allocation of resources to bidders are made for an specific time only [LS06], and when the time has expired the auction is repeated in what is known as a *recurrent* auction, where bidders are continuously competing for the same resources. This kind of auctions have received little attention [LS06, PDJS06, LS05b], but they are gaining importance, since there are many applications where this recurrence takes place, such as e-service oriented marketplaces.

Recurrent auctions present the *Bidder Drop Problem* because during such auctions, bidders can drop out of an auction at any time. This problem occurs when a bidder participating in many auctions is always losing, and then he

may decide to leave the auction, and this is not good for the auctioneer, since reducing the number of bidders gradually decreases the price competition given that the probability of winning increases for the remaining bidders. Therefore, one of the main concerns in recurrent auctions is to keep the agents interested in participating in the auction.

In order to avoid, or somehow decrease, the bidder drop problem, the recurrent auction process should have some degree of *fairness*. A fair solution means that at long term, all the participants accomplish their goals in the same degree, independently of their wealth. The inclusion of this fairness can be somewhat acting against optimality, since the result of an auction could differ from the optimal solution if a suboptimal solution is fairer. However, its long-term effect has better performance than a pure utilitarian view, since the duration of the recurrent auction may be longer with satisfied agents and the final outcome could be much higher.

Some approaches of recurrent auctions using fairness can be found in the literature [LS06, LS05a, MMBL07].

2.5 Robustness

In real-world applications where there exists the possibility that unpredictable changes occur that modify the initial conditions of the problem, the obtained solution may not remain applicable [ABT00]. In such situations robustness would be preferable to optimality, since in the case that changes occur, a robust solution would remain applicable while an optimal solution could not be applicable and therefore it should be re-calculated from scratch.

However, optimality is not completely left out of consideration, instead, the robust solution should be as close to the optimal as possible. Therefore a trade off between optimality and robustness is addressed, as usually the degree of robustness would be inversely proportional to the solution revenue. Some authors have named this drawback "the price of robustness" [BS04].

There are two general approaches for dealing with robustness. Whereas *reactive* techniques address the problem of how to recover from a disruption once it has occurred, *pro-active* methods constructs solutions that account for statistical knowledge of uncertainty.

The first step in incorporating robustness (pro-active) in optimization was taken by Soyster [Soy73], who proposed a linear optimization model to construct a solution that is feasible for all data that belong in a convex set. Unfortunately, the resulting model produces solutions that are too conservative giving up much of optimality in order to ensure robustness (see [ABT00]).

A significant step forward for developing a theory for robust optimization addressing the problem of overconservatism was taken independently by Ben-Tal and Nemirovski [ABT98, ABT99, ABT00], and El-Ghaoui and Lebret [EG97, EG98]. These papers proposed less conservative models by considering uncertain linear problems with ellipsoidal uncertainties. With properly chosen ellipsoids, such formulation can be used as a reasonable approximation to more complicated

uncertainty sets. However, a practical drawback of this approach is that it leads to nonlinear models, which are more demanding computationally than easier linear models by Soyster [Soy73].

Andrew J. Davenport proposed a slack-based technique for robust pro-active scheduling [DGB01]. The central idea behind slack-based techniques is to provide each activity with extra time to execute so that some level of uncertainty can be absorbed without rescheduling.

On the reactive field, Kentaro Tsuchida [TOIK04] presented a robust scheduling method for job-shop problem which consisted on a method to produce robust schedules obtained by iteratively generating new schedules together with appropriate adjustment rules. An adjustment rule is a modification of the schedule, and is used when an environmental change happens, by shifting or replacing jobs. They calculate an expectation evaluation value of each robust solution and keep the best solution based on various initial situations.

In terms of combinatorial auctions, Alan Holland and Barry O'Sullivan incorporated reactive robustness to them in what they call the "Bid-taker Exposure Problem" defined as: "Given a bid withdrawal in a combinatorial auction, finding an alternative repair solution of adequate revenue without causing undue disturbance to the remaining winning bids in the original solution". Then the problem is to find a *robust solution* which is defined as an allocation that can withstand bid withdrawal (a *break*) by making changes easily to form a repair solution of adequate revenue [HO05], instead of *brittle solution* in which an unacceptable loss in revenue is unavoidable if a winning bid is withdrawn. He proposes an approach to address the Bid-taker Exposure Problem using the Weighted Super Solutions framework [HO04] developed by himself, from the field of constraint programming, to find a robust solution. The weighted super solution guarantees that any subset of bids likely to be withdrawn can be repaired to form a new solution of at least a given revenue by making limited changes. The limitations of this approach is that it only considers bid withdrawal, but does not consider any other kind of change such as variations in the bids, or disobedience of the bidders (for example in an auction-based resource allocation problem the bidders could use more resources capacities than requested).

2.6 Conclusions

Auctions have been one of the most popular mechanisms in economics to resource allocation problems, and are nowadays also a popular method for AI researchers as well. The problem of deciding the winners of an auction is known as the Winner Determination Problem and it can be modeled as a Constraint Optimization Problem (COP).

COPs have been studied for many years and a wide variety of powerful techniques and methods already exist. We can distinguish between optimal and non-optimal techniques. The former can provably find the optimal solution although it usually takes an unfeasible amount of time given that this is a

NP-Hard problem, while the latter are not generally able to guarantee that a solution can be found, neither prove that it does not exist.

Most of the work found in the literature for both optimal and non-optimal methods is focused on the utilitarian principle, and is therefore deficient in environments where the the allocation process is repeated several times as each problem is solved independently.

Another desired feature that is difficult to find in most of the existing methods is robustness. Robustness is gaining importance in the last years because in real, dynamic environments there take place changes and the classical techniques are generally not prepared to deal with such changes in the initial data. The research in this field has already began, but the field of combinatorial auctions is still on its infancy and much work can be done within it.

The research presented in the rest of this work deals with (i) the design of a robustness mechanism to be used in recurrent combinatorial auctions, and (ii) the development of an efficient and versatile combinatorial auction solver.

Chapter 3

Exploratory work

In this chapter we present two separate works, the first is a robustness mechanism for combinatorial auctions, and the second is the development of an efficient algorithm for the Winner Determination Problem for Combinatorial Auctions.

To test the former we will use the case example introduced in the first chapter about coordinating water discharges in a treatment plant. The latter algorithm will be compared, using a popular combinatorial auctions benchmark, against the best solvers existing nowadays.

3.1 Robustness in recurrent combinatorial auctions

In some domains an interesting feature on auctions is to incorporate robustness. Robustness, as it has been introduced in Chapter 2, represents the ability of a solution to overcome unexpected changes in the environment. Under this approach, we are willing to accept a suboptimal solution in order to ensure that the solution remains feasible and near optimal even when the data changes. There are two general approaches for dealing with uncertainty: proactive and reactive. Roughly speaking, proactive robustness means that the obtained solution is robust by itself, being able to absorb some level of unexpected events, while reactive robustness addresses the problem of how to recover from a disruption once it has occurred, providing an alternative solution in case that the primary solution becomes unapplicable.

We will focus only on proactive robustness (for the interested reader on reactive robustness an extended work on that subject can be found in [HO05]), mainly related to scheduling problems that use recurrent combinatorial auctions to distribute the available resources among all the agents in order to not exceed the resources capacities at any time and minimize the makespan (i.e. the time at which all tasks are completed). In the following section a generic recurrent auction procedure to solve scheduling problems is described, and the subsequent section will explain how to add the robustness capability to this framework.

After that, a concrete real-world problem, the waste water treatment system, will be used to evaluate this mechanism.

3.1.1 Coordinating Schedules

Scheduling problems could be solved using a centralized approach, where given all the tasks, it generates a new schedule for each agent, without any conflicts between them. Such centralized approach implies that the central scheduler would made all the decisions. However, such decisions should be made distributedly by each of the agents, since they may not be willing to disclose private information related with the production process upon which their decisions are based. Thus, in order to preserve privacy, a distributed approach is preferable [CDE+06]. Therefore we have designed a multi-agent system. In this scenario there is a single agent representing the shared resources of certain capacities, and a set of agents competing for some resources needed to execute a set of tasks, where each task is defined by a given duration, release time, deadline and resources capacity requirements. From the point of view of an auction, the agent that controls the resources is the auctioneer (seller) which we will call the *coordinator*, the agents performing tasks are the bidders (buyers), and the items being sold are the capacities of the resources.

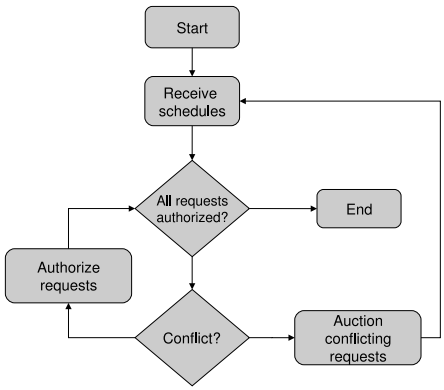

Figure 3.1: Coordination system

The process for coordinating the different schedules is depicted in Figure 3.1. Firstly, the agents inform the coordinator about their scheduled tasks. These schedules contain the set of tasks that they plan to perform in a given period of time, and for each task the information about its starting time, duration, resources capacities (and any other information) is also included. Then a schedule from an agent k is described as $S_k = \{t_1, ..., t_n\}$ where n is the number of tasks contained in this schedule and each task t_i is defined as $t_i = \{s_i, d_i, \overline{q_i}\}$, where s_i stands for the start time, d_i is the duration and $\overline{q_i}$ is the set of resources capacity requirements for this task.

The coordinator, upon reception of these schedules, starts checking for conflicts (that is, whether the resources capacities are exceeded at any time). A conflict arises if the tasks being performed at time t, $T(t) = \{t_i | t \in [s_i, s_i + d_i]\}$, violate the following restriction:

$$\sum_{i=1}^{|T(t)|} q_{i,j} \leq Q_j \quad \forall j \in C \tag{3.1}$$

where $q_{i,j}$ is the capacity requirement of the resource j by the task i, Q_j is the maximum capacity of the resource j, and C is the set of resources. Whenever a conflict is detected, the involved agents (the agents whose tasks are scheduled at the time of the conflict) are informed about it, and an auction is started in order to select which of them will be authorized to use the resources and which will not. Once the auction is completed, the agents are informed about the resolution, in a way that the losers of the auction should modify their schedules. This process is repeated until all the tasks have been authorized, and the result is that each agent has a new schedule, and the resulting schedules do not produce any conflicts.

Next we describe in more detail the conflict resolution method using combinatorial auctions.

3.1.2 Conflict Resolution with Recurrent Combinatorial Auctions

As discussed earlier, we use an auction mechanism to mediate the conflicts between the scheduled tasks (of the respective agents). Once the involved tasks in a conflict have been detected, their corresponding agents are informed about the conflict and the auction process begins. The goal of this process is to select a subset of them, which will be authorized to perform their tasks, while the remaining tasks should be delayed.

The selection criteria is based in the bids submitted by the agents. These bids represent the urgency that each of them has to perform the task. A high bid indicates that the agent really needs (or wants) to perform the task, while a low bid indicates that the agent could delay its task and therefore it can miss the opportunity to perform it at the auctioned time.

Formally, the problem to solve by the coordination agent is the Winner Determination Problem (WDP) for multi-unit combinatorial auctions [KP05] (similar to the multi-dimensional knapsack problem [Kel05]):

$$\begin{aligned} &\max \sum_{i=1}^{NC} x_i \cdot v_i \\ &\text{s.t.} \sum_{i=1}^{NC} x_i \cdot q_{i,j} \leq Q_j \quad \forall j \in C \end{aligned} \tag{3.2}$$

where:

- NC is the number of conflicting tasks,
- $x_i \in \{0, 1\}$ represents whether task i is denied or authorized,
- $v_i \in \mathbb{R}^+$ is the bid value for task i,
- $q_{i,j}$ is the capacity requirement of the resource j for the task i,
- Q_j is the resource j capacity,
- and C is the set of resources.

The auction process is repeated every time a new conflict is detected. This leads to a *recurrent auction*, where the same bidders are continuously competing for the same resources. As explained in the previous chapter, in this kind of auctions the Bidder Drop Problem comes out: in this coordination scenario, it can cause the agents to stop obeying the outcome of the coordination and start behaving on its own, which could conflict with the behavior of the agents agreeing with the coordination.

The bidder drop problem has been typically addressed in the literature using fairness mechanisms [LS05a, MMBL07]. However, although fairness incentivizes agents to participate in the auctions, it does not produce robust solutions by itself. Robustness is a desired feature in these situations, as it would produce solutions taking into account those agents which are most likely to disobey the decisions of the auctioneer if unauthorized, thus preventing overuse of the resources.

3.1.3 Adding Robustness

Now we will describe how to add a robustness model to this coordinating mechanism with recurrent combinatorial auctions. The robustness model consists in three separated components:

- **Trust model** of the agents performing tasks
- **Risk function** of the auctioneer selling the resources (i.e. the coordinator)
- **Robust solution generation**

The first component is concerned with the agents performing tasks. It models their behavior by learning from their actions the circumstances in which an agent does not obey the decisions of the coordinator. Then the coordinator uses these models in order to know in advance which agents are going to disobey if they are unauthorized to perform the task, and act correspondingly. The second component is related to the coordinator and its risk function, as the robustness mechanism varies depending on the risk attitude of this agent. Finally, with the inputs coming from all the agents, the robustness is achieved by combining the risk of the coordinator with the trust on the agents involved in each conflict.

Trust model

An agent performing tasks can disobey the decisions of the auctioneer for several reasons. It is not usually the case that an agent disobeys every decision of the auctioneer independently of the characteristics of the task to perform. Normally, an agent would disobey only the decisions that deny some tasks that it really needs to perform for some reason. Therefore the trust model should not not contain only a unique global value for the degree of trust of an agent, but the trust value should be related to a given task features, as an agent probably disobeys differently as a function of the characteristics of a task. Therefore, the trust model maintains concrete information about the distinct kinds of tasks, in order to learn which tasks are most likely for the respective agent to be disobeyed in case they were denied. Possible task features to build the trust model with include the resources capacity requirements, the task duration, etc.

The information stored about the trust itself is not only the probability of disobeying, but it is generalized with a lie magnitude, as an agent may request to perform some tasks using a given capacity of resources and later use a higher capacity than requested. Consequently, the trust model also stores for each of the considered characteristics these two trust measures:

- **Probability of disobeying**. This value $\in [0..1]$ can be measured in different ways, being the most intuitive the average of disobediences in relation to the total number of auctions the agent has been involved in. However, it could be measured not only counting the number of times that the agent has performed the task when unauthorized, but counting also the times where the agent has performed the authorized task but using a higher amount of capacity than requested.

- **Lie magnitude**. This value $\in [0..\infty]$ represents the degree of the disobedience. For example a value of 1 would represent that when the agent disobeys, it uses the quantity of resources requested for the task, while a value of 1.5 would represent that it uses the 150% of the requested capacity.

A graphical representation of this trust model using only one characteristic of the task is shown in Figure 3.2 (to use more task characteristics, additional dimensions would be added). Note that this model is general enough to allow including even the case where an industry does never disobey the auctioneer (it only performs the task when it is authorized to, so it has a disobey probability of 0 for all task characteristics), but it uses a higher amount of capacity than requested (having a lie magnitude greater than 1 at disobey probability of 0). This is particularly useful in problems where the resource capacity requirements of the agents are quite dynamic.

The trust model is learned by the auctioneer agent at execution time. Every time a task is performed the trust model of the respective agent is updated with the new trust values obtained and related to the characteristics of the current task, i.e. if the task has been performed after the authorization of the auctioneer

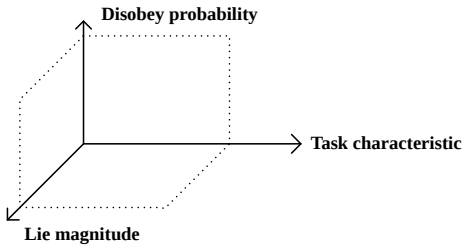

Figure 3.2: Trust model.

or not (the agent has disobeyed), and checking if the resource capacity used is the same as what was requested. The trust model is also consulted each time an agent requests to perform a task; the auctioneer looks for the trust model of the agents willing to perform a task at a given time and gets their trust values in order to solve the current conflict taking them into account.

Risk function

The auctioneer's risk attitude characterizes its willingness to face dangerous situations. Risk attitudes are generally categorized in three distinct classes: risk aversion, neutrality and proclivity. Risk aversion is a conservative attitude for individuals who do not want to be at stake. Risk neutral agents display an objective predilection for risk, whilst agents with a proclivity for risk are willing to engage in gambles where the utility of the expected return is less than the expected utility.

To produce a robust solution the risk attitude of the auctioneer is considered together with the trust models of the agents. For example, a risk-averse auctioneer would consider that every task with a probability of disobeying greater than 0 is going to be performed even if unauthorized, and thus it would auction only the remaining resources capacities over the rest of the tasks. On the other hand a risk-proclive auctioneer would consider that if a task has a low probability of being disobeyed, it would not be the case at this time and hence the auctioneer would auction a bigger amount of resources capacities, although with a higher risk of being overused.

The risk function f_{risk} gives the risk attitude of the coordinator (between 0 and 1) as a function of the probability of disobeying of a given agent and a given task. An example of a risk function is shown in Figure 3.3(a). In this case it represents a risk-averse auctioneer, since the resulting *risk value* is almost always 1 (it considers risky tasks as if they are going to be surely performed even if unauthorized), regardless of the probability of disobeying. On the other hand, a risk-proclive auctioneer would have the final value almost always set to 0 (as seen in Figure 3.3(b)), and a risk-neutral one would have it set accordingly to the probability of disobeying.

We can guess that a risk-averse coordinator will face fewer overuses of the

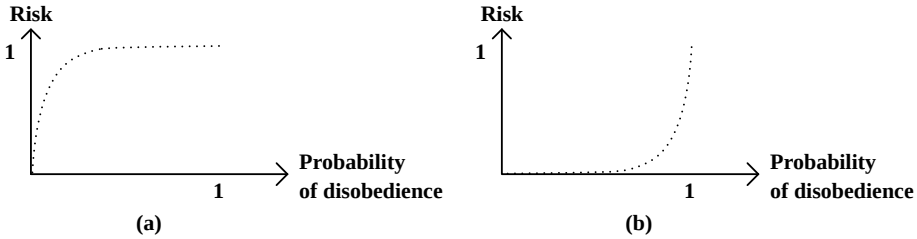

Figure 3.3: Risk attitude function: (a) averse, (b) proclive.

resources. However, as the agents will have less access to the resources, there will be more tasks delayed and thus the makespan will be longer. Instead, with a risk-proclive coordinator the makespan will be shorter, although the resource may be overused. However, in the results section we will observe that this fact is not always happening.

Robustness resolution

Once the auctioneer has defined its risk function and the trust model about the agents performing tasks with different resources requirements has been learned, all of this information can be used to solve any forthcoming conflict in the resources. When a conflict is detected, the auctioneer is faced with a set of tasks, each associated with a set of trust features, obtained from the trust model. Figure 3.4 shows an example, where boxes represent tasks and the number inside them is its capacity requirement, P stands for the probability of disobeying of the respective agent, and M stands for the lie magnitude.

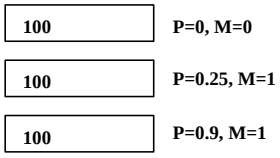

Figure 3.4: Example of tasks with associated trust values.

Then the auctioneer, taking into account the trust levels associated to each task decides which to authorize and which not in function of its risk. To solve this situation a new constraint, the *robustness constraint*, is added to the constraint optimization problem previously formulated in Equation 3.2, where we have n variables $X = \{x_1 ... x_n\}$ (one for each task involved in the conflict), each one representing whether the task is authorized or denied.

The robustness constraint is formulated in a way that the solution finds a balance between the amount of resources required by the authorized tasks and the assumed risk from the unauthorized tasks (appropriately weighted by its probability of disobeying, lie magnitude and the risk function f_{risk} of the

auctioneer). The objective is to not exceed the capacities of the resources (Q_j). This constraint is defined as follows:

$$\sum_{i\in[1,n]} x_i{\cdot}c_i + \sum_{i\in[1,n]} (1-x_i){\cdot}c_i{\cdot}f_{risk}(P_i){\cdot}M_i \leq Q_j \quad \forall j \in C \tag{3.3}$$

The first summatory represents the resources used by the authorized tasks, and the second characterizes the resources potentially used by the unauthorized tasks. Then the unauthorized tasks are considered as if they where performed in the cases where the probability of disobeying of the associated agent (P_i) is high. However this value (appropriately weighted with the lie magnitude M_i) is considered as a function of the risk attitude of the coordinator f_{risk}. In this case we have considered that the lie magnitude is directly multiplied by the *risk value*, but another function could be used as well.

Another way of understanding this equation is by moving the second summatory to the right side. Then it can be read as if the total capacities of the resources get diminished in some degree by the unauthorized tasks that are likely to be performed anyway. Then the tasks are auctioned normally although witl less resources capacities available.

In the next section we will apply the presented mechanism to a real world problem and we will evaluate it in adverse environments to assess how useful this robustness mechanism is.

3.2 The Waste Water Treatment Problem

The Waste Water Treatment Problem, described in Section 1.2.1, can be modeled as a recurrent combinatorial auction, where the auctioneer is the treatment plant, the resource being sold is its capacity, and the agents performing tasks are the industries whose tasks are to perform discharges. Every discharge is defined as $D_i = \{industry\_id_i, s_i, d_i, q_i, \overline{c_i}\}$, where s_i and d_i are the start time and the duration of the discharge, and q_i and $\overline{c_i}$ are the flow and contaminant levels of the discharge. In this case the resource consumption (as well as the individual discharges) does not have only a global capacity limit (hydraulic capacity), but it is extended with many thresholds, one for each contaminant type. Then the goal of the auctioneer is not only to not exceed the hydraulic capacity of the plant but also to have each of the contaminant levels under its thresholds. To this end the discharge flow as well as each of the contaminants are considered as separated resources.

The unauthorized discharges should not cause problems in the production processes of the industries, however if the discharges can not be delayed, there is no coordination possible. Therefore we assume that each industry has a retention tank (of a given capacity) where it can store a discharge whenever it is not authorized, and empty it later on.

With these adjustments, the coordinating scenario described in the previous section can be easily adapted to be applied to this problem, so the robustness mechanism can also be used.

Regarding the proposed robustness mechanism, it is easier to understand more clearly in this concrete problem why is it useful. In this scenario it is conceivable that industries may sometimes disobey the decisions of the plant. The most obvious reason is when an industry has its retention tank completely full; in this case if the forthcoming discharge is not authorized, the industry will be forced to discharge it anyway, thus disobeying the plant. However, an industry could disobey the decisions of the plant for other uncontrolled and unpredictable reasons, for example when the industry needs for some reason to have the retention tank empty (for maintenance purposes, for instance), or when a concrete discharge cannot be stored in the tank because of its high level of contamination, etc. That is the reason why the robustness mechanism has been designed to take into account the characteristics of the task in the trust model.

Here the disobeying probability can be defined as a function of the characteristics of the discharge (or the industry), for example:

- The flow of the discharge.
- Duration.
- Volume (amount of liters of the discharge).
- Contaminant levels.
- Retention tank occupation.

3.2.1 Implementation

To evaluate the coordination mechanism we have implemented a prototype of the system, using Repast[1], a free open source software framework for creating agent based simulations using Java language. The simulation reproduces the coordination process and the communication between the plant and the industries performing discharges. We have created an agent to represent the plant and another one for each one of the industries. So far we have only considered the hydraulic capacity.

To calculate the bid, each industry agent takes into account the urgency for performing the discharge, based on the retention tank occupation. Thus, the bid value of agent i, v_i, is computed as:

$$v_i = \frac{\text{tank occupation}_i}{\text{total tank capacity}_i} \tag{3.4}$$

In case an industry agent has to reschedule its discharges, its behavior is the following: it first tries to store the rejected discharge into the tank; the discharge of the tank is then scheduled as the first activity of the agent after

[1]REPAST Agent Simulation Toolkit, http://repast.sourceforge.net

the current conflict finishes. Otherwise, if the industry has its tank already full, the discharge will be performed anyway.

The objective function to maximize in the auction clearing is the sum of the winning bids values. The free linear programming kit GLPK [GLP] has been used to solve the winner determination problem appropriately modeled as a mixed integer programming problem (MIP). The robustness constraint is added to the constraint optimization problem as an additional constraint.

The trust models of the industries have been implemented using only one characteristic of the discharges: the flow. The models of the industries are learned during the execution by storing, for each different value of flow of a discharge from an industry, two counters for the total number of lies and truths (that is, disobedient and obedient actions), and another value to compute the lie magnitude. These values are updated after each performed discharge in the following way: if the respective industry was authorized by the plant, then the number of truths of the corresponding flow is incremented; alternatively if the performed discharge was not authorized, then the number of lies is incremented. Independently, the value regarding the average lie magnitude (of this concrete flow) is updated with the lie magnitude of the current discharge computed as the division between the used capacity in relation with the requested capacity.

3.2.2 Experimentation results

In order to evaluate the results we have considered some quality measures based on different characteristics of the solution:

- **number of overflows (NO)** occurred during the execution of the schedules
- **maximum flow overflowed (MFO)**, measured in m^3/day
- **total volume overflowed (VO)**, in liters
- percentage of discharge denials being **obeyed** by the industries **(%IO)**

The experiments consisted of simulations using a set of real data provided by the Laboratory of Chemical and Environmental Engineering (LEQUIA). This data is composed of the discharges of 5 industries in two weeks. The first one is a pharmaceutical industry; it is increasing its discharge flow during the week and does not discharge during the weekend. The second one is a slaughterhouse that discharges a constant flow, except at the end of the day when it increases. The third one is a paper industry that discharges a constant flow during the seven days of the week. The fourth one is a textile industry, whose discharges flow oscillates during the day. The fifth one is the waste water coming from the city, whose flow is fixed. The hydraulic capacity of the plant is 32000 m^3/day.

We have tested the mechanism in different scenarios. In the first scenario there is no coordination among the industries (without coordination the treatment plant does never unauthorise a discharge). The second scenario uses the

coordination mechanism and assumes that the industries always obey the decisions of the plant, as long as they have enough tank capacity. The third scenario also uses coordination and we introduce a probability of disobeying the outcome of the coordination mechanism. This probability depends on the occupation of the tank (the higher the occupation, the higher the chances of disobeying); a graphical representation of this function is shown in Figure 3.5. The scenarios with coordination have been tested with and without robustness.

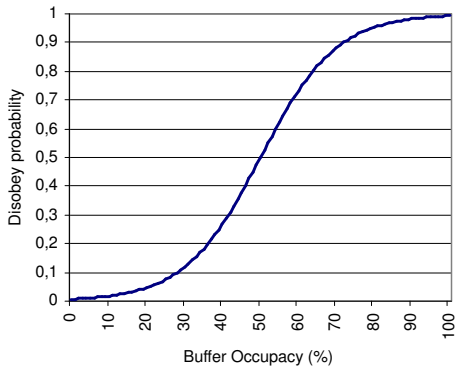

Figure 3.5: Disobey probability function.

Additionally, we have tested the system in two different environments. In the first environment all the industries behave in the same way as explained before, whereas in the second environment there is an industry (the textile, chosen randomly), that will always disobey the decisions of the plant if any of its discharges is unauthorized. We expect that, although the robustness mechanism should improve the outcome in both environments, it would improve more significantly the results of the second environment, as it fits better into its playing field.

	NO	**MFO**	**VO**	IO
No coordination	80	9826	$15.21 \cdot 10^6$	-
Obey	28	4996	$3.74 \cdot 10^6$	98.95
Disobey (0)	77.60 (4.12)	14432 (865.93)	$11.5 \cdot 10^6$ (216866)	98.55 (0.12)
Disobey (0.1)	113.40 (7.55)	14357 (1077.02)	$13.4 \cdot 10^6$ (319429)	97.23 (0.21)

Table 3.1: First environment (all industries behaving similarly), with coordination but without robustness.

Table 3.1 shows the results of the first environment where there is no always-disobeying industries. The first row of the table shows the results obtained without any coordination, i.e. all the discharges proposed by the industries

are authorized. The second row shows the results with coordination where all the industries obey the decisions of the plant (unless its tank is full). The two following rows show the results where the industries disobey using the function shown in Figure 3.5, and setting a minimum probability of disobeying of 0 in the third row and 0.1 in the last.

Comparing these values we can see that with coordination the results have been highly improved, as the volume overflowed is drastically reduced from $15 \cdot 10^6$ to $3.74 \cdot 10^6$ in the case in which all the industries obey. Furthermore the number of overflows and the maximum flow overflowed is also considerably reduced in the case where all the industries obey. In the two cases where the industries disobey we observe that the maximum flow overflowed and the total number of overflows is notably incremented. Instead, the volume overflowed, which is the most important indicator for the goodness of the results, is reduced (yet getting slightly incremented as the probability of disobeying augments).

Disobey	**Risk**	**NO**	**MFO**	**VO**	IO
0	0	78.70 (7.15)	14360 (1522)	$11.3{\cdot}10^6$ (261362)	98.27 (1.57)
	0.5	79 (7.83)	13531 (1396)	$11.4{\cdot}10^6$ (260669)	98.19 (0.24)
	1	84.8 (5.16)	14052 (1006)	$11.3{\cdot}10^6$ (251712)	98.15 (0.17)
0.1	0	126.60 (6.13)	14398 (1604)	$13.3{\cdot}10^6$ (363484)	96.48 (0.31)
	0.5	122.9 (6.84)	13966 (803)	$13.2{\cdot}10^6$ (403934)	96.61 (0.32)
	1	121.3 (7.94)	14233 (1358)	$13.2{\cdot}10^6$ (374673)	96.58 (0.41)

Table 3.2: First environment (all industries behaving similarly), with both coordination and robustness.

Table 3.2 shows the results of adding the robustness mechanism to the coordination system, also experimenting with different disobedience levels of the industries. The robustness has been analyzed with three different risk attitudes of the plant, namely, a completely risk-averse plant (risk=0), a risk-neutral plant (risk=0.5) and a completely risk-proclive plant (risk=1).

We can observe that all the results with disobeying industries are improved using robustness since both the volume overflowed and the maximum flow overflowed is reduced compared without robustness. However the difference is not much relevant, and the number of overflows is generally higher within the robust approach. Regarding the different risk attitudes of the plant, we see that there are not much changes between them. This is not surprising as the industries in this environment do not disobey in the way at which the robustness mechanism

has been designed to work on. Finally considering the percentage of won auctions, the minimum percentage of won auctions and the percentage of industry obeyings, we see that there are neither many changes between all the cases.

	NO	**MFO**	**VO**	**IO**
No coordination	80	9826	$15.21 \cdot 10^6$	-
Obey	112	6523	$6.89 \cdot 10^6$	90.84
Disobey (0)	112 (6.09)	14955 (1201.58)	$12.6 \cdot 10^6$ (233076)	90.98 (0.2)
Disobey (0.1)	119.70 (4.72)	14819 (1373.74)	$14.3 \cdot 10^6$ (263955)	89.96 (0.28)

Table 3.3: Second environment (one industry always disobeying), with coordination but without robustness.

Now we are going to consider the second environment, where there is one industry always disobeying the decisions of the plant. We can see the results without using robustness in Table 3.3 (results without coordination are the same as in the first environment, as there are no unauthorized discharges). Of course in this environment all the values are worse than in the first one. However, as it can be seen in Table 3.4, the robustness mechanism improves the results more significantly than in the first scenario. All the indicators, volume overflowed, number of overflows and maximum flow overflowed are reduced. Regarding the percentage of discharge denials obeyed (%IO), we can note that in Table 3.2, that is, without robustness, the values are lower than in the previous scenario; instead, in the robust approach the values remain almost as high as then. Nevertheless, here again it is not clear the differences produced by different risk attitudes in the plant.

Penalty policy

It should be noted that the robustness mechanism may induce the agents to disobey, as doing so they are going to be always authorized by a risk-averse auctioneer. To avoid such a situation another mechanism should be incorporated to the system. Different mechanisms to achieve that have already been studied, as for example the addition of *fines* (or *penalties*) [SL02] to be paid whenever an agent does not obey the decision of the coordinator; another method would be to stipulate a deposit to be paid for the participants before beginning the coordination, and returned later only to the ones that have obeyed the coordinator. However, the price of these fines or deposits should be studied in more detail in order to make it not too cheap so an agent would prefer to pay it instead of obeying the coordinator, neither too expensive so that a poor agent would have more problems than a rich one to pay it.

Disobey	Risk	NO	MFO	VO	IO
No	*	58	6590	$5.47{\cdot}10^6$	96.77
0	0	77.70 (3.68)	14225 (1212)	$11.8{\cdot}10^6$ (205150)	96.69 (1.57)
	0.5	82.5 (7.66)	15110 (997)	$11.9{\cdot}10^6$ (199074)	96.66 (0.16)
	1	81.2 (4.44)	14018 (1596)	$11.8{\cdot}10^6$ (133988)	96.68 (0.18)
0.1	0	109.50 (3.95)	14150 (1310)	$13.6{\cdot}10^6$ (242619)	95.19 (0.17)
	0.5	113.5 (5.5)	13708 (1040)	$13.6{\cdot}10^6$ (445501)	95.16 (0.37)
	1	110.9 (8.16)	14522 (1571)	$13.6{\cdot}10^6$ (338985)	95.31 (0.29)

Table 3.4: Second environment (one industry always disobeying), with both coordination and robustness.

3.3 Design of an efficient algorithm for the WDP of combinatorial auctions

We have seen in chapter 2 three ways to solve a combinatorial auction. For example we can use general purpose optimization algorithms such as backtracking, branch and bound, etc. We also have specific algorithms to solve combinatorial auctions like CASS and CABOB. We finally showed that Integer Programming (IP) could be used as well to solve this kind of problems and concluded that although in some (few) concrete cases those specific algorithms perform better, IP is nowadays the most used method. An important drawback of IP Solvers is that they are usually commercial packages with an expensive economic cost. However there also exist free open source IP solvers available publicly, being the most used GLPK (Gnu Linear Programming Kit).

In this section we present CABRO, an efficient algorithm for solving the winner determination problem related to combinatorial auctions. It uses a new large variety of simplification techniques, together with upper and lower bounding methods combined with dynamic bid ordering heuristics. Experiments against current methods show that CABRO is in average the fastest free solver (i.e. not including CPLEX), and in some hard instances drastically faster than any other solver.

The idea is to extend in the future this fast solver to solve also multi-unit combinatorial auctions, and to make it suitable for dealing with robustness.

3.3.1 Notation

Here we introduce a few notation that is going to be used through this paper. In a combinatorial auction the auctioneer receives a set of bids $B = \{b_1, ..., b_n\}$, each of them composed by a price $p(b_i)$ and a subset of items $g(b_i)$ of size $n(b_i)$ (such that $n(b_i) = |g(b_i)|$). The complete set of items is $I = \{it_1, ..., it_m\}$.

Useful relations between bids include $b(it_i)$ as the set of bids that contain the item it_i, and $C(b_i)$ as the set of bids compatible with bid b_i (i.e. the set of bids that do not contain any item in $g(b_i)$). Additionally, $C(b_i, b_j)$ and $\neg C(b_i, b_j)$ represent whether bids b_i and b_j are compatible or incompatible.

3.3.2 The Algorithm

CABRO (Combinatorial Auction BRanch and Bound Optimizer) is mainly a branch and bound depth-first search algorithm with a specially significative procedure to reduce the size of the input problem. The algorithm is basically divided in three main phases:

- The first phase performs a fast preprocessing step with the aim of removing as many bids as possible. Bids removed in this phase may be either bids that are surely not in the optimal solution, or bids that surely are. To decide which bids to remove, the algorithm uses different criteria, explained below.

- The second phase consists in calculating upper and lower bounds for each bid. In order to compute the upper bound for a given bid, a relaxed linear programming problem (LP) is formulated and solved using linear programming techniques. Then the upper bound of the bid is computed generating a solution quickly containing it. This phase may also remove a notable amount of bids.

- The third phase completes the problem by means of search, concretely a branch and bound depth first search. This phase uses also linear programming techniques as heuristic and for pruning the search space.

In some instances it is not necessary to execute all the three phases of the algorithm, for example when the optimal solution is already found before the search phase (it happens more usually than expected). The algorithm is able to end prematurely either when all of the bids have been removed or when at some point of the execution the global lower bound reaches the global upper bound.

This algorithm also provides anytime performance, giving the possibility to be stopped at any time during the execution and providing the best solution found so far.

In the following sections each of the three phases of the algorithm are explained in detail.

3.3.3 First phase: Preprocessing

This phase uses fast algorithms (with polynomial-time complexity) to reduce the size of the problem by deleting bids and items that either cannot be present at the optimal solution or that surely belong to it. This phase consists of 8 separate strategies (steps), each of them using a different criteria to remove either bids or items.

- **Step 1: Bids with null compatibility.** In this step all the bids that do not have any compatible bid are deleted, except for the bid with the highest price b_h. These bids are surely not in the optimal solution since the maximum benefit of a solution containing any of them would be its own price, yet it still does not surpass the price of the bid b_h.

- **Step 2: Dominated items.** Items give information about incompatible bids. Still in some cases the information given by an item is already included into another's: the item is *dominated*. Then, the former can be removed without any loss of information. Hence, this step deletes (leaves out of consideration) dominated items.

  More formally, for each pair of items (it_1, it_2) such that $b(it_1) \subseteq b(it_2)$, it_1 may be deleted from the problem since the information given by it_1 is redundant. Figure 3.6 (a) shows an example of this situation; here item it_1 can be deleted since the information given by it_1 ($\neg C(b_2, b_3)$) is already included in the information given by it_2 ($\neg C(b_1, b_2)$, $\neg C(b_2, b_3)$ and $\neg C(b_1, b_3)$).

- **Step 3: Bids of the solution.** In some instances there may exist bids such that all of its items are unique (the bid is the only one containing them), and therefore the bid does not have any incompatible bid. In such situations the bid is surely part of the optimal solution.

  This step finds all the bids complying with this condition, adding them to the optimal solution and being removed from the remaining set of bids. Figure 3.6 (b) shows an example of this situation, where bid b_1 is added to the optimal solution given that its item i_1 is unique.

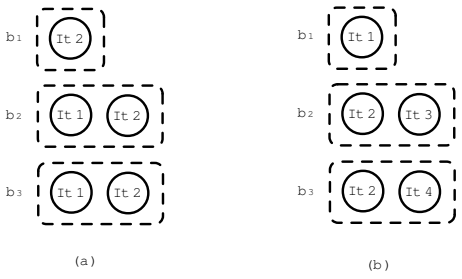

Figure 3.6: Examples of (a) dominated item (it_1) and (b) solution bid (b_1).

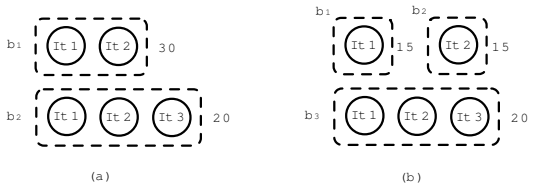

Figure 3.7: Example of (a) dominated and (b) 2-dominated bids. In (a) b_1 dominates b_2, and in (b) b_3 is dominated by the union of b_1 and b_2.

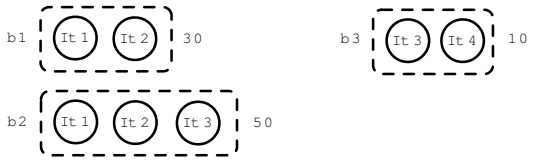

Figure 3.8: Example of pseudo-dominated bid (b_1 is pseudo-dominated).

- **Step 4: Dominated bids.** This is the same pre-processing step that CASS [FLBS99] and CABOB [SSGL01] perform: the elimination of dominated bids. A bid is dominated by another when its set of items includes another bid's items and its price is lower. More formally, for each pair of bids (b_i, b_j) where $g(b_i) \subseteq g(b_j)$ and $p(b_i) \geq p(b_j)$, b_j may be removed as it is never preferable to b_i. Figure 3.7 (a) shows an example of a dominated bid (b_1 dominates b_2).

- **Step 5: 2-Dominated bids.** This is an extension of the previous technique, checking whether a bid is dominated by a pair of bids. In some cases a bid is not dominated by any single bid separately, but the union of two bids together (joining items and adding prices) may dominate it. Figure 3.7 (b) shows an example of a 2-dominated bid (the union of b_1 and b_2 dominates b_3).

  This step can be easily generalized to check *n-dominated* bids. However, the probability of a bid being dominated by n bids is very low for higher values of n, still requiring much more processing (finding all subsets of size n), so this generalization is not useful at all for $n > 2$.

- **Step 6: Pseudo-dominated bids.** This step is a quite more complex generalization of the dominating techniques. Here we deal again with pairs of bids (b_i, b_j) such that not all of the items in b_i are contained in b_j, but there is one single item it_k not included. In this situation the bid b_i can be removed only if adding to its price the price of its best (highest price) compatible bid containing item it_k is not higher than the price of the bid b_j.

  In such a situation b_j is always preferable to b_i even when taking b_j together with its best compatible bid; therefore b_i does definitely not belong

```
{1}  function lowerBound(b)
{2}     C ← compatible(b)
{3}     s ← p(b)
{4}     sort(C)
{5}     while C is not empty do
{6}        x ← first(C)
{7}        s ← s + p(x)
{8}        C ← C ⋂ compatible(x)
{9}     end-while
{10} return s
```

Figure 3.9: Pseudo-code algorithm of lower bound function

to the optimal solution and might be removed.

Figure 3.8 illustrates this situation: here b_2 pseudo-dominates b_1 since its price (50) is higher than the sum of bid b_1's price (30) plus the price of its best compatible bid containing the item it_3, in this case b_3 (10), therefore b_1 can be removed.

Again, this step can be generalized to check for bids with more than one item not included. Unfortunately, like in the previous case, the computational cost required together with the reduced rate of success dissuades this extension.

- **Step 7: Upper and lower bound values.** In this step, fast upper and a lower bounds are assigned to each bid with the aim of deleting bids with its upper bound lower than a *global lower bound* (GLB)[2], since they cannot improve the best solution already found.

  The upper bound u of a bid b_x is calculated according to Equation 3.5 where $C'(b_x, it_k)$ is the set of compatible bids of b_x including item it_k.

$$u(b_x) = p(b_x) + \sum_{\forall i \notin g(b_x)} \max_{\forall j \in C'(b_x, it_k)} \frac{p(b_j)}{n(b_j)} \tag{3.5}$$

  Roughly, it computes the upper bound of a bid b_i by adding to its price the best possible prices of the bids containing the items not included in $g(b_i)$.

  After that, the lower bound of the bids is then calculated with the algorithm shown in figure 3.9, which roughly constructs a solution of a bid by iteratively attempting to add all of its compatible bids to the solution. Its compatible bids are ordered in descending order according to the upper bound previously calculated (line 4 of the algorithm).

[2]The global lower bound (GLB) is the best (maximum) lower bound found, associated to a valid solution.

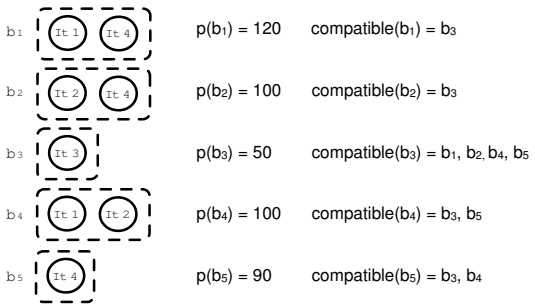

Figure 3.10: Example of compatibility-dominated bid (b_2 is compatibility-dominated by b_1).

All the solutions obtained with this algorithm are valid solutions and update the GLB accordingly. Note that GLB actually stores the best solution to the problem found so far (although it may not be the optimal one), therefore it can be returned immediately if the user decides to stop de execution, thus providing anytime performance.

- **Step 8: Compatibility-Dominated bids.** This step is another generalization of dominated bids. A bid b_i is compatibility-dominated by another bid b_j if the set of compatible bids of b_i is a subset of the set of compatible bids of b_j and its price is lower.

  More formally, for each pair of bids (b_i, b_j) where $C(b_i) \subseteq C(b_j)$ and $p(b_i) \geq p(b_j)$, b_j may be removed as it is never preferable to b_i. Figure 3.10 shows an example where b_2 is not dominated by b_1 but it is compatibility-dominated.

Once all of these steps have been executed, since the problem has changed, it may be the case that some bids and items previously undeleted can now be removed. For example the deletion of a bid may cause the appearance of dominated items and vice-versa. Therefore phase 1 is repeated until it does not remove any more bid or item.

3.3.4 Second phase: Upper and Lower Bounding

In the second phase, the algorithm calculates improved upper and lower bounds for each bid. In order to compute the upper bound for a given bid b_i, a relaxed linear programming (LP) problem is formulated. This relaxed formulation defines the bids in such a way that they can be accepted partially (a real number in the interval [0, 1]), therefore it can be solved using the well-known simplex algorithm [Dan56], which solves almost every instance in polynomial-time. The relaxed version does not contain the current bid b_i neither none of the bids with items included in b_i (i.e. its incompatible bids). Adding the price of the bid b_i to the solution of the relaxed LP problem gives a new upper bound that is usually much more precise than the one obtained in step 7 of phase 1.

This step firstly performs an ordering of the bids according to the upper bound value calculated in step 7 of phase 1 in ascending order. Then the process of calculating new upper bounds using the simplex method starts with the bid with the lower upper bound, and each time a bid's upper bound is lower that the GLB, it is deleted, thus decreasing the size of the subsequent bids' simplex. Note that the chosen ordering, beginning with the "worst" bids, may seem inappropriate, but this is in fact a good strategy since the worst bids' upper bounds are usually much faster to compute than the "best", hence we quickly obtain accurate upper and lower bounds that may allow to remove lots of bids rapidly, thus decreasing the size of the problem and making "best" bids also faster to be computed. This fact has been verified experimentally.

Regarding the lower bound for each bid b_i, it is computed using the values returned by the LP solver, and updates the GLB accordingly. The solution is constructed by firstly considering any value greater than 0.5 to be actually 1; that is, part of the solution. This assumption is not inconsistent (it does not produce solutions containing incompatible bids) because compatible bids are restricted to sum at most 1, therefore two incompatible bids cannot have both values larger than 0.5. After that, the remaining bids (with values smaller or equal to 0.5) are attempted to be put into the solution in descending order. Of course if the solution of the LP was integer this process is not required, as it is the optimal solution for that bid.

3.3.5 Third phase: Search

The third phase (*iCabro*) performs a branch-and-bound depth-first search with the remaining bids of the previous phases (L). The full algorithm can be seen in Figure 3.11. The value of the best solution found so far (GLB) is stored in the global variable *bSolution*. Initially *bSolution*=0, and the search starts by calling *iCabro*(L,0).

The *iCabro* procedure processes the incoming list of bids L performing the following steps:

- The algorithm begins getting the first bid b of the list L (recall that L is sorted according to the upper bound computed in phase 2). A new list $L2$ is created as the intersection between L and $C(b)$ (compatible bids of b). In deeper nodes (as it is a recursive function) the set $L2$ represents the compatible bids with the current solution.

- After that, the algorithm formulates and solves the Linear Programming (LP) problem related to the current solution. If the result of the LP problem is integer then the algorithm finishes (prunes) the current branch, as the optimal solution of the branch has been found.

- At line 10 the algorithm verifies if the upper bound of the current solution is greater than the GLB (the best solution found so far). If this is the case the search continues through this branch updating the best current solution if necessary. Otherwise, the branch is pruned.

```
{1}  procedure iCabro(L,cSolution)
{2}    for each element b of L
{3}        L2 ← L ⋂ compatible(b)
{4}        cSolution2 ← cSolution ⋃ b
{5}        LPSol ← simplex(cSolution2)
{6}        if LPSol is integer then
{7}            cSolution2 ← cSolution2 ⋃ LPSol
{8}            L2 ← ∅
{9}        end-if
{10}       if v(LPSol) > v(bSolution) then
{11}           if v(cSolution2) > v(bSolution) then
{12}               bSolution ← cSolution2
{13}           end-if
{14}           if L2 is not empty then
{15}               sort(L2)
{16}               iCabro(L2, cSolution2)
{17}           end-if
{18}       end-if
{19}   end-for
{20} end-procedure
```

Figure 3.11: Pseudo-code algorithm of iCabro procedure

- At line 14 the algorithm verifies that the $L2$ set is not empty, given that if it is empty then it means that the current solution does not have any more compatible bids and consequently the branch can be pruned. Alternatively, if this condition does not happen, then the following action is to sort the list $L2$ according to the upper bound of each bid, in order to perform a recursive call to *iCabro* with the list $L2$.

3.3.6 Experiments

To evaluate the CABRO algorithm we have compared it against both specific algorithms and MIP solvers. We have chosen CASS for the specific solver instead of CABOB because although their authors claim that it outperforms CASS, it is not available publicly, neither for academic research purposes. For the MIP solver, both GLPK (free) and CPLEX 10.1 (commercial) have been tested.

Test examples have been generated using the popular benchmark for combinatorial auctions CATS (Combinatorial Auctions Test Suite) [LBPS00]. Since its first release in 2000, the CATS suite has become the standard tool to evaluate and compare WDP algorithms [SSGL01, PCS06]. It generates realistic combinatorial auction instances, following a set of real-world situations as well as many previously published distributions (called legacy) [LBPS00, LBNS02]. Given a required number of goods and bids, all the distributions select which

goods to include in each bid uniformly at random without replacement.

For most of the real-world distributions a graph is generated representing adjacency relationships between goods, and it is used to derive complementarity properties between goods and substitutability properties for bids. Two of them concern complementarity based on adjacency in (physical or conceptual) space, while the others concern complementarity based on correlation time. The characteristics of each distribution are the following [LBPS00]:

- Paths (PATHS). This distribution models shipping, rail and bandwidth auctions. Goods are represented as edges in a nearly planar graph, with agents submitting a set of bids for paths connecting two nodes.
- Arbitrary (ARB). In this distribution the planarity assumption is relaxed from the previous one in order to model arbitrary complementarities between discrete goods such as electronics parts or colectables.
- Matching (MAT). This distribution concerns the matching of time-slots for a fixed number of different goods; this case applies to airline take-off and landing rights auctions.
- Scheduling (SCH). This distribution generates bids for a distributed job-shop scheduling domain, and also its application to power generation auctions.

The legacy distributions are the following [LBPS00]:

- L1, the *Random* distribution from [San02], chooses a number of items uniformly at random from [1,m], and assigns the bid a price drawn uniformly from [0, 1].
- L2, the *Weighted Random* distribution from [San02], chooses a number of items g uniformly at random from [1, m] and assigns a price drawn uniformly from [0, g].
- L3, the *Uniform* distribution from [San02], sets the number of items to some constant c and draws the price offer from [0, 1].
- L4, the *Decay* distribution from [San02] starts with a bundle size of 1, and increments the bundle size until a uniform random drawn from [0, 1] exceeds a parameter α.
- L5, the *Normal* distribution from [HB00], draws both the number of items and the price offer from normal distributions.
- L6, the *Exponential* distribution from [FLBS99], requests g items with probability $C{\cdot}e^{-g/q}$, and assigns a price offer drawn uniformly at random from [0.5g, 1.5g].

- L7, the *Binomial* distribution from [FLBS99], gives each item an independent probability of p of being included in a bundle, and assigns a price offer drawn uniformly at random from $[0.5g, 1.5g]$ where g is the number of items selected.

We have also created a new distribution called TRANSPORTS (TRANS) based on the second example application domain introduced in chapter 1.1, the road transportation optimization. The problem roughly consists of finding the best assignment of available drivers to a set of requested services given a cost function and subject to a set of constraints (see [**?**] for more details). To model this problem as an auction the bids represent journeys (a set of services) associated with a driver, therefore its items represent the services performed as well as the driver used. Note that the original problem consists in minimizing the final cost of doing all the services, while an auction is concerned on maximizing. Therefore, the costs associated to the bids are appropriately transformed so that the maximized solution corresponds to the real (minimizing) solution.

3.3.7 Results

We have generated 100 instances of each distribution with different amounts of bids and items. Each instance has been solved using CABRO, CASS, GLPK 4.9 and CPLEX 10.1 with a timeout of 300 seconds. The first three methods have been run in a 2.4GHz Pentium IV with 2Gb of RAM running under Windows XP SP2, whilst CPLEX has been run on a 3.2GHz Dual-Core Intel Xeon 5060 machine with 2 Gb of RAM running under GNU/Linux 2.6.

Results for each distribution

Figure 3.12 shows the results of each method on distribution L1 (the random distribution). In this distribution CPLEX gets the best results since it solves all the instances, while CABRO fails as the number of items increases. On the other hand GLPK is surprisingly getting the worst results although it uses MIP like CPLEX; even CASS is performing better, yet not as good as CABRO, both far from CPLEX.

In distribution L2, CABRO is clearly the best method as it can be seen in Figure 3.13. In this distribution CASS is the second best method, performing even better than CPLEX, and leaving GLPK at the last position. It is overwhelming the efficiency of CABRO, given that the increase of execution time as the number of items and bids augments is imperceptible.

The results of distribution L3 can be seen in Figure 3.14. Here, CPLEX is the best method, followed closely by GLPK, at some distance CABRO and lastly CASS obtaining the worst results as it only solves 3 instances. Note that in this distribution, opposedly to L2, the complexity augments as the number of items increases but the number of bids decreases, while in L2 (and the majority of distributions) the complexity augments when both items and bids are increased.

In L4, CPLEX and GLPK are clearly the best solvers, as it can be observed in Figure 3.15. Their execution times are always very low, while CABRO finds

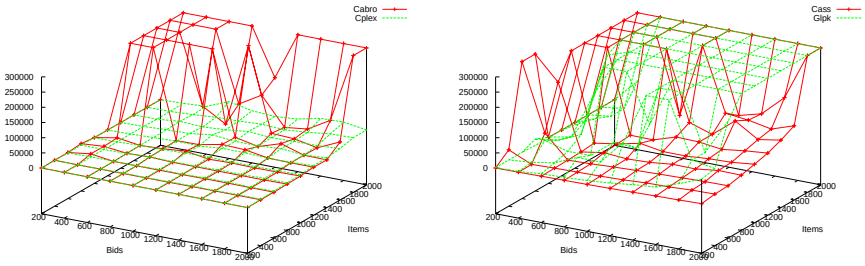

Figure 3.12: L1 Distribution.

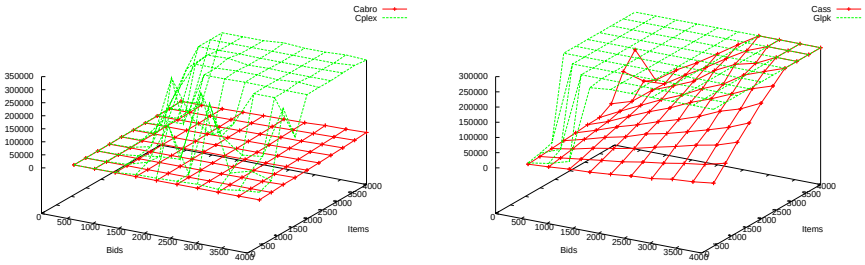

Figure 3.13: L2 Distribution.

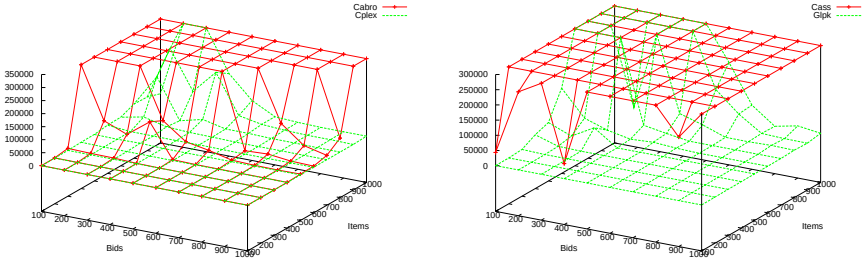

Figure 3.14: L3 Distribution.

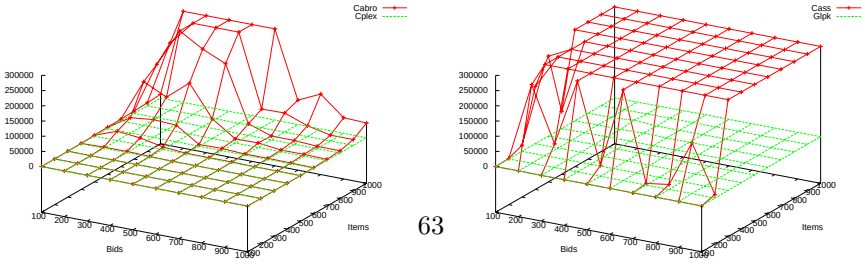

Figure 3.15: L4 Distribution.

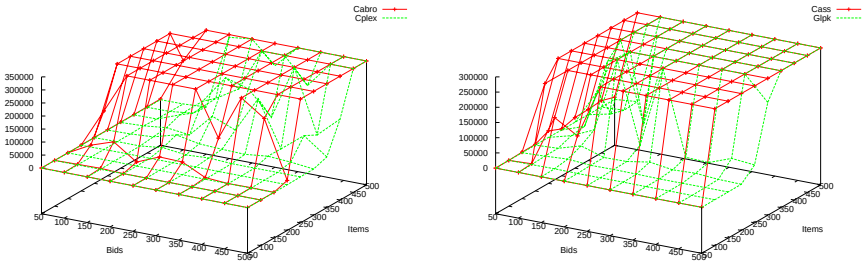

Figure 3.16: L5 Distribution.

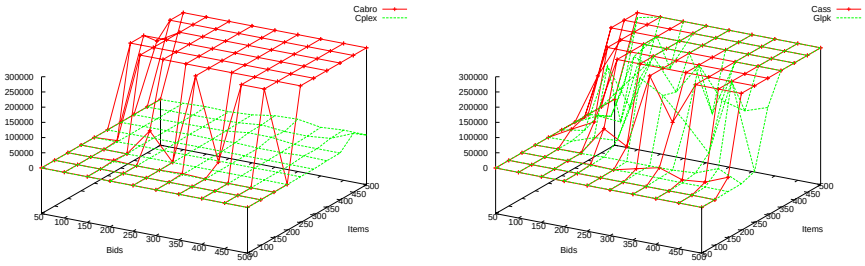

Figure 3.17: L6 Distribution.

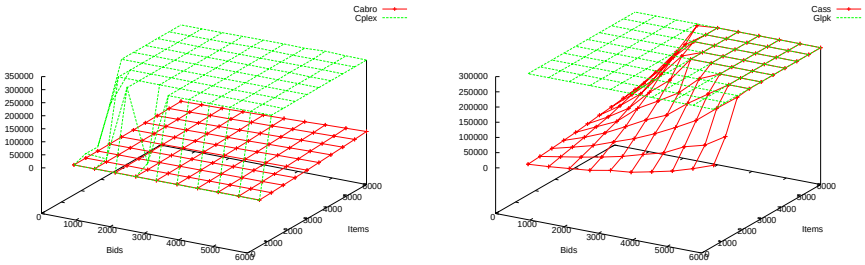

Figure 3.18: L7 Distribution.

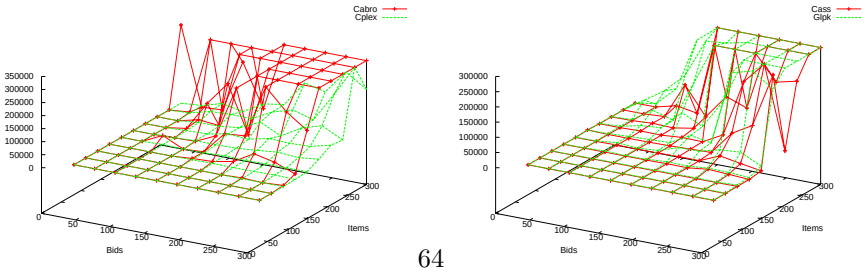

Figure 3.19: Arbitrary Distribution.

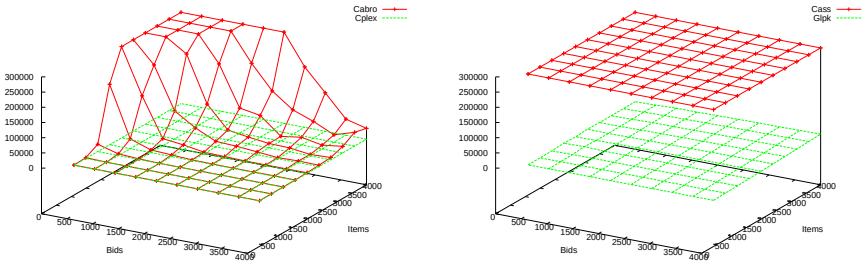

Figure 3.20: Matching Distribution.

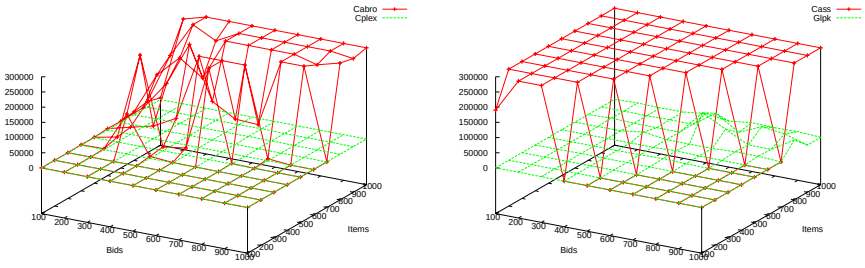

Figure 3.21: Paths Distribution.

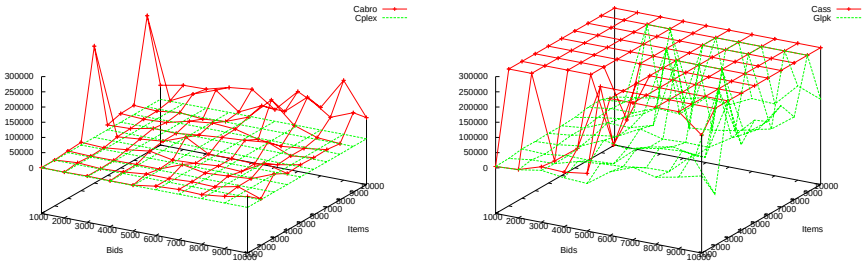

Figure 3.22: Scheduling Distribution.

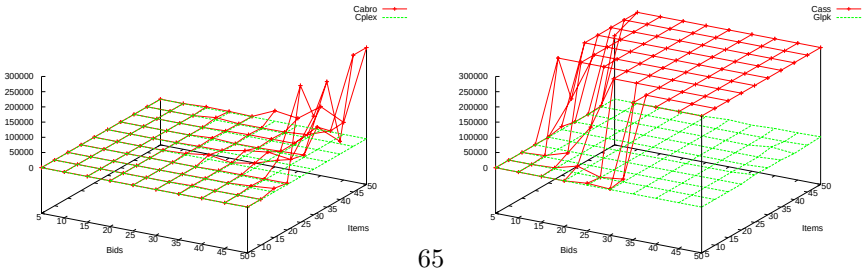

Figure 3.23: Transports Distribution.

difficulties to solve the instances with a large number of items. Here, CASS only solves the instances with a very few number of items.

Regarding distribution L5, shown in Figure 3.16, again CPLEX is the best method, followed by GLPK and CABRO, whilst CASS is the last once more. A similar situation occurs in the next distribution L6, shown in Figure 3.17. However, in that distribution CASS and CABRO obtain practically the same results.

A completely different situation happens in distribution L7, shown in Figure 3.18. Here, CABRO is far more efficient than any other method, being able to solve all the instances in very little time, whereas CPLEX and GLPK fail to solve the majority of the cases. In this distribution CASS obtains the second position, still far from CABRO.

With regard to the other set of distributions, we can observe in Figure 3.19 that in the arbitrary (ARB) distribution, there is not a clear winner, given that the differences are not as big as in the other distributions. However, in absolute results, CPLEX obtains the first position followed by CASS, GLPK and finally, CABRO. This is the one and only distribution in which CABRO gets the last position.

Instead, in the matching (MAT) distribution, shown in Figure 3.20, the LP-based solvers CLPEX and GLPK are clearly superior to CABRO and still more than CASS. In this distribution, like in L3, the most difficult instances are curiously the ones having lots of items but few bids.

In the paths (PAT) distribution shown in Figure 3.21, CPLEX is obtaining the best results, followed closely by GLPK, at some distance CABRO and CASS far away.

Conversely, in the scheduling (SCH) distribution shown in Figure 3.22, both CPLEX and CABRO are getting the best results, still CPLEX seems slightly faster than CABRO. GLPK is the third best and CASS the worst.

Finally, Figure 3.23 shows the results in the transports (TRANS) distribution. In this distribution, CLPEX and GLPK obtain the best results, followed by CABRO, and ultimately CASS which requires much more processing than the other methods.

Overall results

Figure 3.24 shows the total time (in seconds) required for each method to solve all the instances of all the distributions. Here we can observe that CPLEX is in average the fastest solver since it solves all the instances (1167 auctions) in considerably less time than CABRO. Recall that the machine used for CPLEX is considerably faster than the one used for the others; however, we believe that the results in equal machines would not change significantly. Yet CABRO spends less time than the free solvers GLPK and CASS.

Figure 3.25 shows the results in each of the distributions comparing the total time required (in seconds) to solve all the instances of each distribution with the four methods. Note that in the cases in which the execution time is very little the bar is not visible. Here we can observe that in two distributions (L2

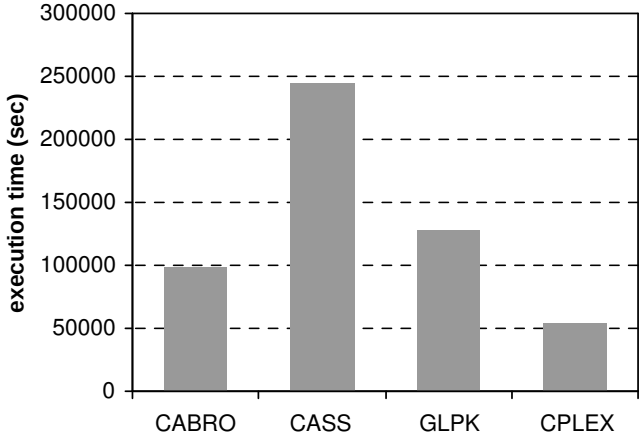

Figure 3.24: Overall comparative.

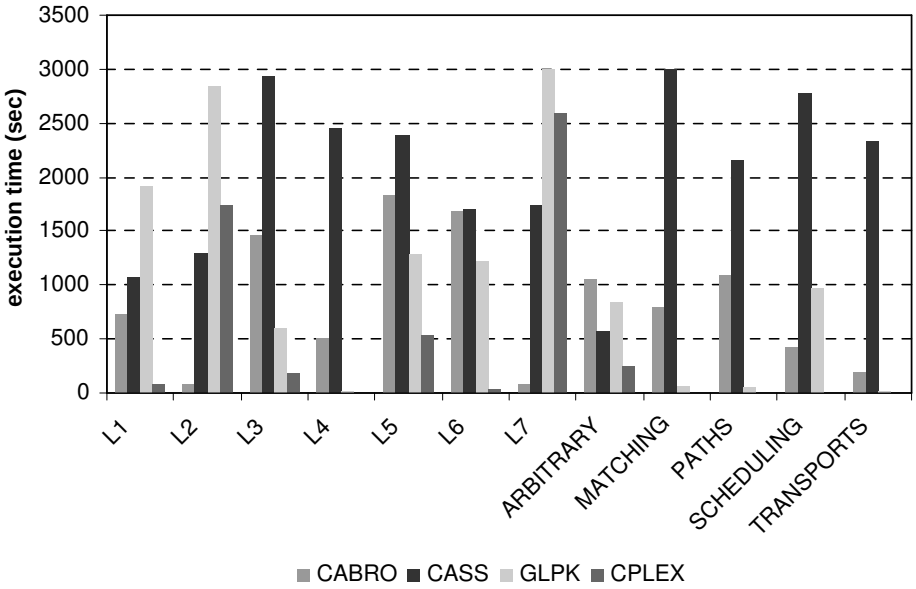

Figure 3.25: Comparative over distributions.

	CABRO		*CASS*		*GLPK*		*CPLEX*	
	S	$\neg S$	S	$\neg S$	S	$\neg S$	S	$\neg S$
$L1$	74	21	67	28	39	56	95	0
$L2$	100	0	90	10	7	93	50	50
$L3$	54	46	3	97	84	16	98	2
$L4$	91	9	22	78	100	0	100	0
$L5$	41	59	23	77	61	39	90	10
$L6$	46	55	46	54	70	30	100	0
$L7$	100	0	68	32	0	100	15	85
ARB	71	29	86	14	81	19	99	1
MAT	83	17	0	100	100	0	100	0
$PATHS$	45	27	1	71	72	0	72	0
SCH	98	2	9	91	84	16	100	0
$TRANS$	97	3	24	76	100	0	100	0
$TOTAL$	899	268	439	728	798	369	1019	148

Table 3.5: Solved (S) and unsolved ($\neg S$) problems before the timeout.

and L7) CABRO is clearly the best algorithm, while CPLEX is the best in the remaining distributions.

Regarding the free solvers, GLPK is the best solver in the majority of distributions, still CABRO outperforms GLPK in L1, L2, L7 and SCH. CASS is only rather competitive in L1, L2, L7 and ARB distributions, although achieving the best performance in the latter.

Table 3.5 shows the number of problems solved (S) and not solved (¬S) before the timeout, for each method and each distribution. The results are similar to the execution time results, with CPLEX being the best method in absolute results, as it solves up to 1019 instances (87%). However, CABRO solves 899 instances (77%) and in two distributions it performs better than CPLEX. Therefore, there is not any method that can be claimed to be the best, since it depends on the kind of data that the auction is processing. Particularly, CABRO is performing better for the weighted random and binomial distributions, solving 100% of the instances, while CPLEX only solves 15% in L7 and 50% in L2.

3.4 Conclusions

In the state of the art chapter we realized that there was not so much work concerned with adding proactive robustness to combinatorial auctions. To fill the gap, we have designed a robustness mechanism to be used in this kind of auctions. Moreover, it is general enough to be used on any kind of auctions and furthermore any problem related to resource allocation. The proposed mechanism is concerned mainly with preventing possible overuses in the resources caused by disobeying users. This approach has been applied in a real-world problem obtaining the expected results.

Additionally, a new efficient algorithm for solving combinatorial auctions has been implemented with the idea of being used in the robustness auction resolution. This algorithm uses many simplification techniques, together with a

smart heuristic function that provides more pruning. From the tests performed we observe that this algorithm outperforms in average the best current free solvers and in some instances even the best comercial solvers.

Chapter 4

Thesis planning

4.1 Summary

In the previous chapter we have presented a **robustness mechanism** for recurrent combinatorial auctions together with a new efficient **winner determination algorithm**. In this initial proposal the robustness mechanism is focused in creating a robust solution according to the risk attitude of the auctioneer and it works in a real-world scenario as we have seen in the results. However a deeper study and analysis is needed in order to improve this mechanism so it can be more extensively used.

Regarding the winner determination algorithm, its results are encouraging since it outperforms in average the best actual free solvers. However there are some concrete distributions where the algorithm is inefficient and more work can be done to improve this. Furthermore the algorithm only deals with combinatorial auctions of single items, and an extension to multi-unit auctions would be useful since as we have seen in the state of the art chapter, there are not so many specific algorithms for this kind of auctions.

The following sections describe in more detail the work that should be done during the thesis period:

4.1.1 Robustness mechanism

The robustness mechanism has been designed in such a way that different risk attitudes should produce different outcome. However, we have not noticed significant changes in the results of the case study when varying the risk attitude of the treatment plant from risk-averse to risk-proclive. Maybe this case study is not appropriate to deal with different risk attitudes or maybe the risk function should be fixed. Therefore, further work needs to be done on this matter.

The presented trust model considers tasks with different characteristics independently. This feature needs to be improved as in problems where the tasks characteristics were too dynamic it would be useless as there would never be two identical tasks. One option to overcome this problem could be to consider

fuzzy tasks characteristics, so that two tasks with slight different characteristics would share the trust values.

Another important aspect to take into account is that the robustness mechanism may induce the agents to disobey, as doing so they are going to be always authorized by a risk-averse coordinator. To avoid such a situation another mechanism should be incorporated to the system. Different mechanisms to achieve that have already been studied, as for example the addition of *fines* (or *penalties*) to be paid whenever an agent does not obey the decision of the coordinator [SL02]; another method would be to stipulate a deposit to be paid for the participants before beginning the coordination, and returned later only to the ones that have obeyed the coordinator. However, the price of these fines or deposits should be studied in detail in order to make it not too cheap so an agent could still be willing to pay if the benefit of unlawfully using a resource is higher than the penalty for doing so, neither too expensive so that a poor agent would have more problems than a rich one to pay it.

4.1.2 Winner determination algorithm

The presented winner determination algorithm is very efficient in some distributions. Nevertheless, it is not so efficient in others as there are distributions in which other solvers are drastically faster than ours. In such cases we believe that it is possible to create new techniques to improve the performance of our algorithm on those distributions.

In fact, we have already devised new reduction strategies roughly based on finding an upper bound drawn from a concrete lower bound (a valid solution), obtaining much better estimations than those obtained in the first phase of the algorithm that estimates using only a single bid. Additionally, other sorting criteria to obtain better lower bounds could be examined.

We also plan to integrate the pre-processing phases into the search phase, so that at each node of the tree the reduction phases would be done on each sub-problem. We expect that these concrete changes may significantly improve the overall algorithm performance.

Another interesting point would be to extend this algorithm to deal also with multi-unit combinatorial auctions, since as we have seen in Chapter 2 there are not many specific algorithms for this kind of auctions.

Finally, we also would want to study other criteria to evaluate the algorithm as we have dealt only with the execution time. Other indicators, such as the anytime behavior and the memory consumption could be studied, since concretely the latter is a known drawback of MIP solvers.

4.1.3 Case studies

So far we have only tested the robustness mechanism within the waste water treatment system domain. Furthermore, we have used only one restriction: the hydraulic capacity constraint. Therefore, two different areas should be explored:

- Firstly, extend the waste water treatment problem to deal with the contaminants levels restrictions. This will turn the internal auction to a multi-unit combinatorial auction, which should be solved with the previously explained planned multi-unit version of the CABRO algorithm.
- The research should also be done in other domains, as for example the road transportation domain presented in Section 1.1.

The evaluation of the algorithms for the waste water treatment problem has been done comparing the results in different pre-defined scenarios with and without coordination, with and without robustness, and varying the risk attitude of the auctioneer. However, a comparison against the best possible solution to the problem (obtained for example using a centralized approach) has not been considered and it would be interesting as it would help to extract more objective conclusions about our methods.

4.2 Schedule

Figure 4.1 shows an approximate schedule to elaborate the thesis within two years.

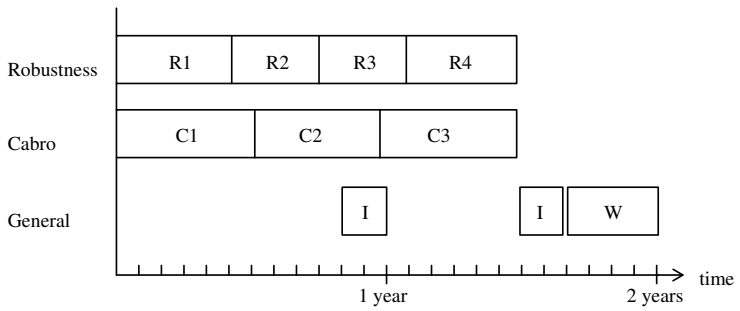

Figure 4.1: Thesis time schedule.

Our work will include alternatively enhancements on the robustness mechanism and improvements on the winner determination algorithm. Next we explain each of the tasks in the two areas.

Work on the robustness mechanism:

- R1. Duration: 5 months. Study and test with new case studies.
- R2. Duration: 4 months. Address the risk attitudes issue.
- R3. Duration: 4 months. Development of a fuzzy algorithm to deal with task characteristics.
- R4. Duration: 5 months. Study penalties policies.

Work on the winner determination algorithm:

- C1. Duration: 6 months. Study new reduction strategies and heuristics to improve overall efficiency.
- C2. Duration: 6 months. Integration of the two pre-processing phases into the search phase.
- C3. Duration: 6 months. Extension to multi-unit combinatorial auctions.

General work:

- I. Duration: 2 months. Integration of the robustness mechanism with the winner determination algorithm.
- W. Duration: 4 months. Write the thesis.

In the course of time, various papers will be written and published with the obtained results in national and international conferences and journals.

Chapter 5

Contributions

The work developed in the last two years within the eXiT group[1] at the University of Girona has led to several publications in the field of Artificial Intelligence. Although some of them have not specifically focused on robustness, they constitute an indicator of the acquired knowledge concerning optimization problems, mainly combinatorial auctions.

5.1 Articles in conferences

- **Víctor Muñoz**, Javier Murillo, Dídac Busquets and Beatriz López. Improving Water Quality by Coordinating Industries Schedules and Treatment Plants. *AAMAS Workshop on Coordinating Agent Plans and Schedules (CAPS). Honolulu, Hawaii, USA. May 16-18, 2007.*

- Javier Murillo, **Víctor Muñoz**, Beatriz López and Dídac Busquets. Dynamic configurable auctions for coordinating industrial waste discharges. *Fifth German conference on Multi-Agent System Technologies MATES. Leipzig, Germany. September 24-26, 2007.*

- Javier Murillo, **Víctor Muñoz**, Dídac Busquets and Beatriz López. Coordinating Agents' Schedules through Auction Mechanisms. *Planning, Scheduling and Constraint Satisfaction, The Conference of the Spanish Association for Artificial Intelligence (CAEPIA). Salamanca, Spain. November 12-16, 2007.*

- Javier Murillo and **Victor Muñoz**. Agent UNO: Winner in the 2007 Spanish ART Testbed competition. *Workshop on Competitive agents in Agent Reputation and Trust Testbed, The Conference of the Spanish Association for Artificial Intelligence (CAEPIA). Salamanca, Spain. November 12-16, 2007.*

[1] Control Engineering and Intelligent Systems Group

- Josep Lluís de la Rosa, Ricardo Mollet, Miquel Montaner, Daniel Ruiz and **Víctor Muñoz**. Kalman Filters to Generate Customer Behavior Alarms. *Artificial Intelligence Research and Development, 10th Catalan Congress on Artificial Intelligence CCIA 07. Sant Julià de Lòria, Andorra. October 25-26, 2007.*

- **Víctor Muñoz**, Miquel Montaner and Josep Lluís de la Rosa. Seat Allocation for Massive Events Based on Region Growing Techniques. *Artificial Intelligence Research and Development, 9th Catalan Congress on Artificial Intelligence CCIA 06. Perpignan, France. October 26-27, 2006.*

5.2 Articles in conferences (submitted)

- **Víctor Muñoz**, and Dídac Busquets. Managing Risk in Recurrent Auctions for Robust Resource Allocation. *Submitted to ECAI 2008.*

- **Víctor Muñoz**, and Javier Murillo. CABRO: Winner Determination Algorithm for Combinatorial Auctions. *Submitted to ECAI 2008.*

- **Víctor Muñoz**, and Javier Murillo. Improving Waste Water Treatment Quality through an auction-based management of discharges. *Submitted to IEMSS 2008.*

- Miquelo Bofill, **Víctor Muñoz**, and Javier Murillo. A Comparison of Techniques for solving the Waste Water Treatment Problem. *Submitted to AAAI 2008.*

5.3 Journals

- **Víctor Muñoz**, Javier Murillo, Dídac Busquets and Beatriz López. Schedule Coordination through Recurrent Multi-Unit Combinatorial Auctions (Extended version of the CAPS workshop paper). *Multiagent and Grid Systems. An International Journal. Under review.*

- Beatriz López, **Víctor Muñoz**, Javier Murillo, Federico Barber, Miguel A. Salido, Montserrat Abril, Mariamar Cervantes, Luis Fernando Caro and Mateu Villaret. Experimental Analysis of optimization techniques on the road passenger transportation problem. *Engineering Applications of Artificial Intelligence (EAAI) Journal. Under review.*

- Javier Murillo and **Víctor Muñoz**. Agent UNO: Winner in the 2007 Spanish ART Testbed competition. *Ibero-American Journal of Artificial Intelligence. To appear in 2008.*

5.4 Awards

- Winner in the Second Spanish ART (Agent Reputation and Trust) Testbed Competition. Valencia, Spain. March 26-27, 2007.
- 7th classified in the 2007 International ART (Agent Reputation and Trust) Testbed Competition. Honolulu, Hawaii, USA. May 14-18, 2007.
- 2006 Catalan Association for Artificial Intelligence (ACIA) Award to the best final career project 2006. Project title: Allocation algorithm for distributing attendants at F1 Grands Prix.

Bibliography

[ABT98] Arkadi Nemirovski Aharon Ben-Tal. Robust convex optimization. *Math Operations Research*, 23:769–805, 1998.

[ABT99] Arkadi Nemirovski Aharon Ben-Tal. Robust solution to uncertain programs. *Operations Research*, 25:1–13, 1999.

[ABT00] Arkadi Nemirovski Aharon Ben-Tal. Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical Programming*, 88(3):411–424, 2000.

[Bar99] R. Bartak. Constraint programming: In pursuit of the holy grail. *In Proceedings of the Week of Doctoral Students (WDS), Prague, Czech Republic, June 1999.*, 1999.

[BDHK06] Martin Bichler, Andrew Davenport, Gail Hohner, and Jayant Kalagnanam. *Combinatorial Auctions*, chapter Industrial Procurement Auctions, pages 593–612. MIT Press, 2006.

[BS04] Dimitris Bertsimas and Melvyn Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.

[CDE+06] Y. Chevaleyre, P.E. Dunne, U. Endriss, J. Lang, M. Lemaître, N. Maudet, J. Padget, S. Phelps, J.A. Rodríguez-Aguilar, and P. Sousa. Issues in multiagent resource allocation. *Informatica*, 30:3–31, 2006.

[Cer85] V. Cerny. A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[Cer87] V. Cerny. Tabu search methods in artificial intelligence and operations research. *ORSA Artificial Intelligence*, 1, No. 2, 6, 1987.

[CK95] P. Crescenzi and V. Kann. A compendium of np optimization problems. *Technical report SI/RR-95/02, Dipartimento di Scienze dell'Informazione, Università di Roma "La Sapienza"*, 1995.

[Cla71] Edward H. Clarke. Multipart pricing of public goods. *Public Choice*, 11(1):17–33, 1971.

[Dan56] George B. Dantzig. The simplex method. *RAND Corp*, 1956.

[Dan68] George B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1968.

[DGB01] A.J. Davenport, C. Gefflot, and J.C. Beck. Slack-based techniques for robust schedules. In *Proceedings of the Sixth European Conference on Planning (ECP-2001)*, pages p–q, Xyz. 2001.

[Dor92] Marco Dorigo. *Optimization, Learning and Natural Algorithms (in Italian)*. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, 1992.

[dVV03] Sven de Vries and Rakesh V. Vohra. Combinatorial auctions: A survey. *INFORMS Journal on Computing*, (3):284–309, 2003.

[EG97] H. Lebret El-Gahoui, L. Robust solutions to least-square problems to uncertain data matrices. *SIAM J. Matrix Anal. Appl.*, 18:1035–1064, 1997.

[EG98] F. Oustry H. Lebret El-Gahoui, L. Robust solutions to uncertain semidefinite programs. *SIAM J. Optim.*, 9:33–52, 1998.

[FLBS99] Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 548–553, 1999.

[Gas02] William Gasarch. The p = np poll. *SIGACT News 36, 33(2)*, 2002.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, January 1979.

[GLP] GLPK. GNU Linear Programming Kit, http://www.gnu.org/software/glpk/.

[Gro73] T. Groves. Incentives in teams. *Econometrica*, 41(4):617–631, 1973.

[HB00] Holger H. Hoos and Craig Boutilier. Solving combinatorial auctions using stochastic local search. In *Procedings of AAAI'00*, pages 22–29, 2000.

[HO04] A. Holland and B. O'Sullivan. Weighted super solutions for constraint programs. *Technical Report: No. UCC-CS-2004-12-02.*, 2004.

[HO05] A. Holland and B. O'Sullivan. Robust solutions for combinatorial auctions. *In ACM Conf. on Electronic Commerce. (to appear).*, 2005.

[Hol75] John H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, 1975.

[HPE68] Raphael B Hart P. E., Nilsson N. J. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics SSC4*, (2):100–107, 1968.

[Kan59] Leonid Vitaliyevich Kantorovich. The best use of economic resources. *Moscow, Acad. Nauk SSSR*, 1959.

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[Kel05] Terence Kelly. Generalized Knapsack Solvers for Multi-unit Combinatorial Auctions: Analysis and Application to Computational Resource Allocation. *LNAI*, 3435:73â–86, 2005.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598*, 220, 4598:671–680, 1983.

[Kha79] Leonid G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Math. Dokl.*, 20(1):191–194, 1979.

[KM72] V. Klee and G.J. Minty. How good is the simplex algorithm? *Inequalities*, (3):159–175, 1972.

[KP05] J. Kalagnanam and D. Parkes. *Handbook of Supply Chain Analysis in the E-Business Era*, chapter Auctions, bidding, and exchange design. Kluwer Academic Publishers, 2005.

[Kum92] Vipin Kumar. Algorithms for constraint-satisfaction problems: a survey. *AI Mag.*, 13(1):32–44, 1992.

[LBNS02] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *Constraint Programming (CP)*, pages 556–572, 2002.

[LBPS00] Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *ACM Conference on Electronic Commerce*, pages 66–76, 2000.

[LD60] A. H. Land and A. G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, (28):497–520, 1960.

[LS05a] Juong-Sik Lee and Boleslaw K. Szymanki. A novel auction mechanism for selling time-sensitive e-services. *Proc. 7th International IEEE Conference on E-Commerce Technology (CEC'05), Munich, Germany, July 2005, pp. 75-82.*, 2005.

[LS05b] Juong-Sik Lee and Boleslaw K. Szymanski. An analysis and simulation of a novel auction=based pricing mechanism for network services. *Technical report, Department of Comupter Science, Rensselaer Polytechnic Institute*, 2005.

[LS06] Juong-Sik Lee and Boleslaw K. Szymanski. Auctions as a dynamic pricing mechanism for e-services. In *Service Enterprise Integration*, pages 131–156. Cheng Hsu (ed.), Kluwer, New York, 2006.

[Mac81] A. K. Mackworth. Consistency in networks of relations. In B. L. Webber and N. J. Nilsson, editors, *Readings in Artificial Intelligence*, pages 69–78. Kaufmann, Los Altos, CA, 1981.

[MMBL07] Víctor Munoz, Javier Murillo, Dídac Busquets, and Beatriz López. Improving water quality by coordinating industries schedules and treatment plants. In Mathijs Michiel de Weerdt, editor, *Proceedings of the Workshop on Coordinating Agents' Plans and Schedules (CAPS)*, pages 1–8. IFAAMAS, may 2007.

[Pap93] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley, November 1993.

[PCS06] Yoav Shoham Peter Cramton and Richard Steinberg. *Combinatorial Auctions*. MIT Press, 2006.

[PDJS06] Terry R. Payne, Esther David, Nicholas R. Jennings, and Matthew Sharifi. Auction mechanisms for efficient advertisement selection on public displays. In *ECAI*, pages 285–289, 2006.

[Pea84] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

[PJ08] Y.K. Penya and N.R. Jenning. Optimal combinatorial electricity markets. *International Journal of Web Intelligence and Agent Systems 6 (1)*, 2008.

[PS98] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization : Algorithms and Complexity*. Dover Publications, July 1998.

[RB82] Vernon L. Smith Rassenti, Stephen J. and Robert L. Bulfin. A combinatorial auction mechanism for airport time slot allocation. *Bell Journal of Economics*, (13):402–417, 1982.

[Rec60] I. Rechenberg. *Evolution strategies*. 1960.

[RN03] St. Russell and P. Norvig. *Artificial Intelligence - A Modern Approach*. Prentice Hall International Series in Artificial Intelligence. Prentice Hall, 2003.

[RPH95] M. H. Rothkopf, A. Pekec, and R. M. Harstad. Computationally manageable combinatorial auctions. Technical Report 95-09, 19, 1995.

[San02] Tuomas Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1-2):1–54, 2002.

[SL02] Tuomas Sandholm and Victor Lesser. Leveled commitment contracting: A backtracking instrument for multiagent systems. *AI Magazine*, 23(3):89–100, 2002.

[Soy73] A. L. Soyster. Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research*, 21(5):1154–1157, 1973.

[SSGL01] Tuomas Sandholm, Subhash Suri, Andrew Gilpin, and David Levine. CABOB: A fast optimal algorithm for combinatorial auctions. In *IJCAI*, pages 1102–1108, 2001.

[THCS01] Ronald L. Rivest Thomas H. Cormen, Charles E. Leiserson and Clifford Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2001.

[TOIK04] Kentaro Tsuchida, Hiroaki Oka, Yoshitomo Ikkai, and Norihisa Komoda. A robust scheduling method for a job shop problem in production by using data carriers. In *SMC (2)*, pages 1464–1468, 2004.

[Tur36] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936.

[Vic61] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16(1):8–37, 1961.

[VJRS96] C. R. Reeves G. D. Smith V. J. Rayward-Smith, I. H. Osman. *Modern Heuristic Search Methods*. Wiley, December 1996.

[vN45] John von Neumann. A model of general equilibrium. *Review of Economic Studies 13: 1-9.*, 1945.