Universitat de Girona

ROBUSTNESS ON RESOURCE ALLOCATION PROBLEMS

Víctor MUÑOZ SOLÀ

ISBN: 978-84-694-2594-7
Dipòsit legal: GI-372-2011
http://hdl.handle.net/10803/7753

UNIVERSITAT DE GIRONA

ESCOLA POLITÈCNICA SUPERIOR

PhD Thesis

Robustness in Resource Allocation Problems

by

Víctor Muñoz i Solà

Advisors

Dr. Dídac Busquets i Font

Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Major subject: Computer Science) at the University of Girona

2010

Robustness in Resource Allocation Problems

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy at the Universitat de Girona

Author:

Víctor Muñoz i Solà

Advisor:

Dr. Dídac Busquets i Font

Programa de Doctorat: Tecnologies de la Informació

Departament d'Electrònica, Informàtica i Automàtica

Contents

Abstract

Real world resource allocation problems, where a set of tasks with different requirements have to be assigned a set of resources, usually include uncertainties that can produce changes in the data of the problem. For instance, in scheduling the duration or the start time of the tasks could change (due to delays and tasks taking more time than expected). These changes may cause difficulties in the applicability of the solution. Research in approaches that consider data uncertainty while looking for a solution is gaining importance recently, due to the wide use of such problems in many domains.

Solutions that are able to overcome (up to a given degree) the possible changes that can occur in the environment are called *robust solutions*. Auctions are a typical application of resource allocations problems, and uncertainty in this scenario may cause some of the auctioned resources to be not available.

This thesis is focused in finding robust solutions against resource unavailability for auctions. Our notion of robustness is based on resources that become unavailable once an allocation of the auctioned items has been found. We begin with an extensive study with the aim of quantifying the effects of unavailable resources in the revenue of the auction. Then, we mathematically formalize this concept of robustness and propose an approach for finding such robust solutions in combinatorial auctions. In our approach, we first extend previous works on encoding auctions as weighted Max-SAT formulas. Particularly, our approach is sustained in propositional logic and that in turn allows the model to be able to be applied in any Boolean formula, being auctions a particular case. We then present a mechanism to add flexibility to robust solutions, based on supermodels for the Boolean satisfiability framework (SAT). Our approach is flexible in the sense that it allows to easily set the desired degree between optimality and robustness. In addition, we study the balance between optimality and robustness of the solutions and discuss the reasoning that could be made in order to choose the desired trade-off between optimality and robustness.

We also analyze the strategy-proofness of our approach in order to force the participants to express their true valuations, and therefore avoid possible manipulations in the prices of the auction. We will show that although our robustness mechanism is not incentive compatible in a restricted case where only a subset of the resources can fail (the breakage set does not include all the items), for the generic case the same proof does not hold and furthermore we provide two procedures to support the idea that our robust approach is actually incentive compatible in practical cases.

Resum

Els problemes d'assignació de recursos realistes, on un conjunt de tasques amb diferents característiques han d'ésser assignades a un conjunt de recursos, normalment incorporen incerteses que poden produir canvis en les dades del problema. Per exemple, en alguns problemes de planificació, la durada o el temps inicial de les tasques pot canviar (degut a retards o tasques que triguen més del previst). Aquests canvis poden dificultar l'aplicabilitat de la solució. La recerca en tècniques que consideren la incertesa alhora de cercar les solucions està guanyant importància darrerament gràcies a l'àmplia utilització d'aquest tipus de problemes en múltiples dominis.

Les solucions que siguin capaces de superar (dins d'uns límits) els possibles canvis en l'entorn són anomenades *solucions robustes*. Les subhastes són una aplicació típica dels problemes de satisfacció de restriccions, i la incertesa en aquest escenari pot causar la indisponibilitat d'alguns dels recursos subhastats.

Aquesta tesi es centra en trobar solucions robustes en subhastes on els recursos poden passar a ser no disponibles. La nostra noció de robustesa es basa en recursos que esdevenen indisponibles un cop aquests ja han estat subhastats i assignats. Comencem amb un estudi que té com a objectiu quantificar l'efecte que els recursos indisponibles produeixen en els ingressos de la subhasta. En la nostra aproximació, primer estenem treballs anteriors per codificar subhastes en fórmules del tipus "weighted Max-SAT". Particularment, la nostra aproximació es basa en lògica proposicional i aquest fet permet al model ser aplicable a qualsevol fórmula booleana, essent les subhastes només un cas particular. També presentem un mecanisme per a afegir flexibilitat a les solucions robustes basat en supermodels, en el marc de la satisfactibilitat Booleana (SAT). La nostra aproximació és flexible en el sentit de que permet establir fàcilment el punt desitjat entre robustesa i optimalitat. Addicionalment, estudiem el balanç entre la optimalitat i la robustesa de les solucions i discutim el raonament que es podria fer per tal d'escollir el punt més apropiat entre optimalitat i robustesa.

També analitzem la manipulabilitat de la nostra aproximació amb l'objectiu de forçar als participants a informar de les seves valuacions reals, i així evitar que disminueixin els preus de la subhasta pel seu propi benefici. Demostrarem que tot i que el nostre mecanisme de robustesa no impedeix la manipulabilitat en un cas restringit on només un subconjunt dels recursos poden fallar (el conjunt de trencament no inclou tots els recursos), la prova no es pot estendre al cas genèric, i de fet presentem dos procediments que reforcen el fet que la nostra aproximació de robustesa no és manipulable en casos pràctics.

Acknowledgements

I would like to give thanks to all the people that has helped me during my PhD. First of all, I want to thank my supervisor Dídac Busquets, who has guided my research and revised all my work. He has been always available whenever I have needed his help. I could not have written this thesis without his priceless help. Secondly, to Beatriz López, who was my supervisor only during my first year of PhD, but has continued always encouraging my work and providing useful comments.

I also wish to express my gratitude for the support of the eXiT (Control Engineering and Intelligent Systems) research group, and to all the people working there, specially to my PhD mate Javier Murillo. All of them have contributed to make these years better.

My most sincere appreciation to the Cork Constraint Computation Centre (4C) at the University of Cork, Ireland. Specially to Dr. Alan Holland, who accepted me as a research visitor and helped in many aspects of my work.

Miquel Bofill and Mateu Villaret also deserve special thanks for introducing me to SAT and SMT, and specially, for their inestimable help in the mathematical parts of this thesis (proofs, theorems, lemmas, etc.).

This thesis has been done with the support of the Commissioner for Universities and Research of the Department of Innovation, Universities and Company of Generalitat of Catalonia and of the European Social Fund, with the support of the University of Girona BR Grant program, with the support of Spanish Ministry of Science and Innovation MICINN project SuRoS (TIN2008-04547) and Desarrollo y aplicación de técnicas de resolución de problemas de scheduling en entornos multi-agente (TIN2004-06354-C02-02).

List of Figures

List of Tables

List of Acronyms and Abbreviations

CA : Combinatorial Auction

CABOB : Combinatorial Auction Branch On Bids

CASS : Combinatorial Auction Structured Search

CATS : Combinatorial Auctions Test Suite

CNF : Conjunctive Normal Form

COP : Constraint Optimization Problem

CP : Constraint Programming

CSP : Constraint Satisfaction Problem

DCOP : Distributed Constraint Optimization Problem

DPSB : Discriminatory Price Sealed Bid

FPSB : First Price Sealed Bid

GLB : Global Lower Bound

GLPK : Gnu Linear Programming Kit

IDA : Iterative Deeping A*

ILP : Integer Linear Programming

LIA : Linear Integer Arithmetic

LP : Linear Programming

LRA : Linear Real Arithmetic

MIP : Mixed Integer Programming

MUCA : Multi-Unit Combinatorial Auction

NP : Non-deterministic Polynomial-Time

PB : Pseudo-Boolean

QF : Quantifier Free

RAP : Resource Allocation Problem

SAT : Satisfiability

SMT : SAT Modulo Theories

SPSB : Second Price Sealed Bid

SS : Super Solutions

UPSB : Uniform Price Sealed Bid

VCG : Vickrey Clarke Groves (Generalized Vickrey Auction)

WDP : Winner Determination Problem

WSS : Weighted Super Solutions

WWWTPP : Waste Water Treatment Plant Problem

CHAPTER 1
Introduction

This chapter gives a brief introduction to the topics discussed in this thesis. First, an introduction of the weaknesses in resource allocation problems is presented in order to motivate the development of this research. After that, the concrete topic of this dissertation is accurately delimitated. Then, a set of objectives are established in order to evaluate the quality of this work. This chapter ends with the list of publications that have come out during the progress of this research and a description of the structure of this report.

1.1 MOTIVATIONS

Researchers in mathematical optimization have been historically concerned in developing and improving algorithms for particularly hard problems where the inputs (the data that defines the problem) are known and static. In real world situations, however, such data is not so precisely known; instead, there is usually some degree of uncertainty that may change the values of some variables in some circumstances. These changes in the data may cause difficulties in the applicability of the solutions found with the initial values.

In resource allocation problems, where a set of tasks with different requirements have to be assigned a set of resources, uncertainty appears as unpredictable events that may change both the characteristics of the resources to allocate and the requirements of the tasks to be performed. In this situation it is possible that even a very small change in the inputs of the problem causes the optimal solution to become completely unfeasible. For instance, a task taking a little more time than expected to be performed, could cause the following task to not being performed at its scheduled time, which could in turn cause a collapse of the whole schedule. This weakness, originated by the inherent uncertainty that typically accompanies real world problems, motivates the research in approaches that consider data uncertainty while looking for a solution, so that the obtained solution is able to overcome (up to a given degree) possible changes that may occur, i.e. it is a *robust solution*.

Clearly, the price of robustness is optimality [5], since generally the optimal solution to a problem is not robust, and consequently most of the robust solutions are sub-optimal. This fact turns the problem into a multi-objective one, where the two objectives (robustness and optimality) are conflicting. In some systems that require providing functionality in adverse situations, robust solutions are usually preferred since the loss of optimality can be overcome by the increase in applicability of those robust solutions. However, in the domains where the benefit depends directly on the optimality, robustness is usually sacrificed. In general, the desired degree of robustness needs to be identified for each case after a thorough analysis.

Although we all agree that robustness is a desired quality in many practical applications, it is cer-

tainly a quite vague term and it is therefore not easy to typify such robustness on the whole for any possible optimization problem involving uncertainty. Instead, we first need a proper and concise definition of the following concepts:

- the resource allocation problem itself,
- the implications of the uncertainty in its data, and
- the exact meaning of a robust solution in this setting

In order to deal with the first point, in this thesis we adopt auctions [49] as the means to define the resource allocation problem. Auctions are an ancient economical mechanism to sell a set of items to a group of buyers, where the buyers express their interest in the items by sending bids, and the auction is used both to decide the winners of the items and the price of the items. In the last years auctions have been increasingly used to deal with resource allocation problems, since they enable an efficient distribution of resources amongst the buyers. An auction is simply, but also strictly, defined with a set of items to sell (resources), and a set of bids requesting them at a given price. This strict definition gives a concise and proper modeling of resource allocation problems.

In this context, the implications of data uncertainty in an auction can be straightforwardly defined as either changes in the resources or in the bids. A change in a resource would imply that either the resource disappears or its capacity decreases[1] (in the case of multi-unit resources). On the other hand, as the bids are composed by the (set of) requested resources and its price, a change in a bid could be an alteration of any of these values. Some researchers have also considered another kind of change concerning the bids: the bid withdrawal [36], where a winning bid reneges on it and does not neither want the item/s nor paying the agreed quantity. This fact could cause an unacceptable loss in revenue for the auctioneer if some "critical" winning bid is withdrawn, and therefore a robust solution guaranteeing that possible bid withdrawals can be easily repaired with a restricted loss in revenue is desired.

Regarding the exact meaning of a robust solution, it is usually defined as its ability to absorb the changes of the environment caused by uncertainties. This ability can be mainly understood from two points of view:

- if the solution remains valid when breakages happen
- if the solution is easily *reparable* when a breakage occurs

In terms of an auction, a robust solution is one that the set of winning bids does not cause problems (loss of revenue, spare items, etc.) to the auctioneer when unexpected changes happen.

1.2 The Topic of this Research

In this research we focus mainly in a concrete class of auctions that are known as combinatorial auctions [16], which are a generalization of auctions where the bidders are able to place bids on

[1] Increases in the resources availability are naturally not considered as they do not negatively affect the applicability of the solution.

bundles of items rather than just on individual items. This generalization is needed in order to be able to represent the kind of resource allocation problems that we are interested in.

As we will see in the following chapter, there has already been some work in robustness for combinatorial auctions, mostly focused in changes in the bids, and concretely in the problem of bid withdrawal [36]. Therefore, in our work we will deal with another kind of uncertainty that may affect an auction, that is, changes in the resources. More concretely we will consider the resource unavailability problem, which happens when a resource becomes unavailable once the auction has been solved (the winning bids have been selected and announced) but before the resources have been given to the winning bidders. Regarding the robustness of the solutions, we give a definition of a robust solution that is based on the maximum number of changes in the environment (caused by inherent uncertainties) and the maximum size of the repair. This definition allows us to cover the two previous definitions of robust solutions, given that setting a maximum size of the repair of zero actually means that the solution must remain valid when changes in the environment happen, while any size greater than zero enables a repair.

As far as we know, this concept of robustness considering resource unavailability based on repairable solutions has not been studied before. However, it is not because it is an insignificant problem, but because typical applications of auctions do not have a long transaction phase (the time in which the items are delivered to the agents) and, therefore, unavailable resources are simply not auctioned. Nowadays, auctions are being applied to a wide range of real world domains, some of them having a long transaction phase, where some of the resources may turn unavailable after an allocation has been found. This thesis will be focused on the analysis and development of techniques for finding robust solutions for such resource allocation problems.

As an example, consider a (combinatorial) auction designed for assigning rooms for simultaneous conferences. In the building where the conferences are allocated, there are several available rooms with different characteristics, such as total capacity, availability of projector, air conditioning, chair style, etc. Each conference organizer would bid for (possibly multiple) combinations of rooms, according to the conference's needs (number of participants, number of parallel sessions, conference requirements, etc.). The auction would be executed a few weeks before the conferences start because some preparatives would be made in the rooms, and also the conference programs should be designed and printed with the rooms' location information. Nevertheless, right before the actual use of the rooms, or even worse, during the conferences, unexpected events could happen (as for example a projector in a room that is broken, a room that needs some maintenance or repair works, not to mention fires, floods or any other disaster), causing the unavailability of some rooms. In such a case, it would be reasonable to make as few changes as possible in the rooms assignments, instead of finding a new (maybe completely different) allocation, in order to minimize the inconveniences caused by other room reassignments. Therefore if a robust solution is searched in the first assignment (instead of an optimal), such solution would be ready to minimize the effects of unforseen events, maybe only requiring few changes to the organizers and thus avoiding possible conflicts.

There are many other domains where the problem of resources that become unavailable after the auction has been solved comes out, as for example the assignment of rooms in a hotel to client demands, the allocation of sportive events in sport complexes, the placing of vacation days for the employees in a company, etc. Therefore, robustness for resource unavailability is indeed applicable to many domains today, and in the future will probably be a deciding point in many critical real

world applications.

In summary, in this thesis we will analyze the up till now unexplored problem of resource unavailability in combinatorial auctions with repairable solutions. We will first quantify the effects that resources becoming unavailable cause in the revenue of the auction, in a set of real world scenarios. We will mathematically formalize this concept of robustness and propose an approach for finding such robust solutions in combinatorial auctions. Our approach is flexible in the sense that it allows to easily set the desired degree between optimality and robustness. Particularly, our approach is sustained in propositional logic [9] and that in turn allows the model to be able to be applied in any Boolean formula, being auctions a particular case.

1.3 OBJECTIVES

The objectives that we set to this thesis can be grouped in five main groups:

- the quantification of the negative effects that resources that become unavailable in combinatorial auctions produce to the auctioneer
- the design of a mechanism to incorporate robustness based on repair solutions in order to avoid this problem
- the adaptation of that mechanism to add flexibility in the solutions so that the desired degree between optimality and robustness can be easily set
- the study of the strategy proofness of the proposed mechanism
- the evaluation of the results obtained using the proposed methods, varying the different available parameters; and an analysis of the trade-off between optimality and robustness of the solutions

The problem of resource unavailability in combinatorial auctions has not been studied before. The reason for that is because current applications domains of combinatorial auctions did not have to deal with this problem, but now that combinatorial auctions are becoming more popular and are being applied to domains with long transaction phases, this problem cannot be neglected any more. The first objective of this research is to measure the consequences of such problem which will raise the necessity of incorporating robustness. Extensive analysis are performed in different auction distributions modeling various real world scenarios, in order to quantify the implications of resources that become unavailable after a solution to the combinatorial auction has been found in a wide range of applications.

In the related work chapter, we will see that supermodels [25] and super solutions [30] have been successfully applied to add robustness in propositional satisfiability (SAT) and constraint programming (CP) respectively. We also use them for our approach concerning resource unavailability. Thus, the second objective of this thesis consists in the adaptation of the existing supermodels (for SAT) to our notion of robustness. For this purpose we will extend (generalize) the existing encoding of a combinatorial auction to a SAT formula, so that it will be able to model the two definitions of robustness (solutions that either remain valid or are easily reparable).

Although the generalized encoding for finding super solutions will allow us to find robust solutions to combinatorial auctions concerning resource unavailability, we will show that such encoding is quite strict as it may not produce any solution in some particularly restrictive instances where no completely robust solution exists. Therefore, the third objective of this work is to incorporate flexibility to the robustness mechanism with the aim of being able to produce solutions in adverse situations, furthermore allowing to define the desired balance point between optimality and robustness and thus, converting it into a more reliable tool.

An auction mechanism is said to be strategy-proof if it never rewards the participants that do not inform their true preferences. This is the reason why in many real applications only strategy-proof mechanisms are used. The fourth objective of this thesis will be to analyze the strategy-proofness of the proposed robustness mechanism.

Finally, our approaches will be evaluated. We will perform experiments in a wide range of instances of combinatorial auctions generated with a tool that lets us test with real-world-like instances. With the obtained results, we will analyze in detail the solutions to see if the problem of resource unavailability is effectively solved with the proposed robustness mechanism and how it affects the goodness of the solutions. Furthermore, we will study the trade-off between optimality and robustness of the solutions and discuss the reasoning that could be made by a real user in order to choose the desired balance point.

1.4 Statement of the Thesis

The thesis can be stated as follows:

In combinatorial auctions resources that become unavailable after a solution has been found can cause large losses in revenue for the auctioneer. Robust solutions can be defined based on the maximum number of breaks (resources that become unavailable) allowed, the maximum size of the repair, and the minimum acceptable revenue for the auctioneer in any case. It is also possible for this robustness to be flexible, allowing to easily set the desired balance point between optimality and robustness.

1.5 Publications

The work developed in the last four years within the eXiT group[2] 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 related areas such as resource allocation problems, constraint satisfaction and optimization problems, combinatorial auctions, fairness, trust & reputation and propositional satisfiability.

The list of publications related to this research that have been published as either articles in journals or in conference proceedings are the following (ordered by year):

[2] Control Engineering and Intelligent Systems Group, http://exit.udg.edu

1.5.1 JOURNALS

- Javier Murillo, Beatriz López, **Víctor Muñoz**, Dídac Busquets. "Fairness in Recurrent Auctions With Competing Markets and Supply Fluctuations". *Computational Intelligence, 2010*. In press.
- Javier Murillo, **Víctor Muñoz**, Dídac Busquets, Beatriz López. "Schedule Coordination through Egalitarian Recurrent Multi-unit Combinatorial Auctions". *Applied Intelligence, Springer, 2009*. Online version available.
- Beatriz López, **Víctor Muñoz**, Javier Murillo, Federico Barber, Miguel ngel Salido, Montserrat Abril, Mariamar Cervantes, Luis F. Caro, Mateu Villaret. "Experimental Analysis of Optimization Techniques on the Road Passenger Transportation Problem". *Engineering Application of Artificial Intelligence 22, pp. 374-388, (Ed. Elsevier Science), ISSN: 0952-1976, 2009*.
- **Víctor Muñoz**, Javier Murillo. "Agent UNO: Winner in the 2nd Spanish ART competition". *Inteligencia Artificial, Revista Iberoamerica de Inteligencia Artificial, ISSN 1137-3601, N. 39, pp. 19-27, 2008*.

1.5.2 ARTICLES IN CONFERENCES

- **Víctor Muñoz**, Dídac Busquets. "Balancing Optimality and Robustness in Resource Allocation Problems". *Nineteenth European conference on Artificial Intelligence ECAI 2010. Lisbon, Portugal. August 16-20, 2010*.
- **Víctor Muñoz**, Javier Murillo, Beatriz López, Dídac Busquets. "Strategies for Exploiting Trust Models in Competitive Multiagent Systems". *Seventh German conference on Multi-Agent System Technologies MATES 2009. Hamburg, Germany. September 9-11, 2009*.
- Javier Murillo, **Víctor Muñoz**, Beatriz López, Dídac Busquets. "Developing Strategies for the ART Domain". *Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA). Sevilla, Spain, November 9-13, 2009*.
- Javier Murillo, **Víctor Muñoz**, Beatriz López, Dídac Busquets. "A Fair Mechanism for Recurrent Multi-unit Auctions". *Sixth German conference on Multi-Agent System Technologies MATES 2008, pp 147-158. Lecture Notes in Computer Science. Volume 5244/2008. September 2008. Kaiserslautern, Germany. September 23-26, 2008*.
- **Víctor Muñoz**, Dídac Busquets. "Robustness in Recurrent Auctions for Resource Allocation". *Artificial Intelligence Research and Development. (Proceedings of 11th Catalan Congress on Artificial Intelligence CCIA 08, pp. 70-79). IOS Press. 2008*.
- **Víctor Muñoz**, Javier Murillo. "CABRO: Winner Determination Algorithm for Single-unit Combinatorial Auctions". *Artificial Intelligence Research and Development. (Proceedings of 11th Catalan Congress on Artificial Intelligence CCIA 08, pp. 303-312). IOS Press. 2008*.
- **Víctor Muñoz**, Dídac Busquets. "Managing Risk in Recurrent Auctions for Robust Resource Allocation". *Proceedings of the 4th European Starting AI Researcher Symposium STAIRS 2008 (in ECAI), pp 140-150. Patras, Greece. July 21-25, 2008*.

- Javier Murillo, Dídac Busquets, Jordi Dalmau, Beatriz López, **Víctor Muñoz**. "Improving Waste Water Treatment Quality Through an Auction-based Management of Discharges". *Proceedings of the 4th International Congress on Environmental Modelling and Software (iEMSs 2008), pp 1370-1377. Barcelona, Catalonia, Spain. July 7-10, 2008. ISBN: 978-84-7653-074-0.*
- **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.*
- 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.*

1.6 AWARDS

While doing this research, we have participated in some national and international competitions and presented parts of this work to various events with successful results. The following is the list of the received awards:

- Best Paper Award in iEMSs 2008 for the paper "Improving Waste Water Treatment Quality Through an Auction-based Management of Discharges".
- Best Student Paper Award in MATES 2008 for the paper "A Fair Mechanism for Recurrent Multi-unit Auctions".
- Catalan Association for Artificial Intelligence (ACIA) second prize for the Master Thesis work 2009. Project title: Robust Combinatorial Auctions for Resource Allocation.
- Winner in the 2008 International ART (Agent Reputation and Trust) Testbed Competition. Estoril, Portugal. May 12-16, 2008.

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

1.7 Outline of the Thesis

This document is structured in 8 chapters, followed by two appendices, and a bibliography section at the end:

- **Chapter 1, Introduction.** This first chapter has introduced the concept of robustness for resource allocation problems, established the topics that are going to be studied in this research, and delimited the objectives of this thesis.
- **Chapter 2, Background.** In the second chapter, related work on robustness in general and concretely for combinatorial auctions is reviewed and a background for the areas that shall be used in the rest of the document is given, namely, resource allocation problems, auctions, propositional satisfiability (SAT), SAT modulo theories (SMT), supermodels, super solutions and weighted super solutions.
- **Chapter 3, Sensitivity Analysis.** The third chapter performs a "sensitivity analysis" of the solutions in several combinatorial auctions instances in order to see the effects of resources becoming unavailable in the revenue of the auctioneer. This analysis provides a strong motivation for our work.
- **Chapter 4, Robustness against Resource Availability.** This chapter formalizes the concept of robustness based in resource unavailability and provides a modeling for obtaining such robust solutions to SAT formulas. Some experiments applied to combinatorial auctions are performed using the presented model and the obtained results are analyzed and discussed.
- **Chapter 5, Flexible Robustness.** In this chapter the previous modeling is adapted in order to incorporate flexibility so that a trade-off between optimality and robustness can be easily set up. Again, some more experiments are executed to analyze the outcome of this model, and a more extensive analysis of the trade-off between optimality and robustness is performed.
- **Chapter 6, Incentive Compatibility.** This chapter discusses the strategy-proofness of the proposed mechanism. Two methods for proving the non-monotonicity (a necessary condition for being strategy-proof) of our approach are described and tested.
- **Chapter 7, Robustness for Recurrent Auctions.** Here we deal with the problem of recurrency in auctions, discuss the challenges that it poses and propose an approach for achieving robust solutions based in a trust model.
- **Chapter 8, Conclusions and future work.** The last chapter presents the conclusions of this work and provides some ideas for future work.

- **Appendix 1, The WWTP Problem.** The first appendix analyzes the Waste Water Treatment Plant Problem (WWTPP), a case study that is used in some chapters for the experimentation. It performs a benchmark of the different tools available to solve it.
- **Appendix 2, Winner Determination Algorithm for Single-unit Combinatorial Auctions.** The second appendix describes an algorithm for solving optimally combinatorial auctions that was developed during this research with promising results.

CHAPTER 2
Background

In this chapter, we present a background of the topics used in the rest of the document. First of all, we introduce resource allocation problems (RAP) and auctions (specially combinatorial auctions) as the mechanism used for dealing with them. Then, we review some generic approaches for robustness, later focusing on the mechanisms designed for propositional satisfiability and constraint programming, namely supermodels and super solutions respectively. This background is needed for our approach of robustness, which is based in a combination of supermodels and weighted super solutions over satisfiability formulas. Finally, we present the current state-of-the-art on robustness specifically for combinatorial auctions, while comparing the different techniques with the approach we are proposing.

2.1 RESOURCE ALLOCATION

Resource Allocation problems are an important topic in computer science as well as in economics [12]. They consist in the assignment (allocation) of a set of *resources* (probably with different characteristics), to a set of *agents* that want to use them, given a set of *restrictions*.

The resources can be classified according to different features:

- **Durable vs perishable**: *Durable* resources are those that maintain their value as time goes by. That is, no matter when they are used, their properties are not changed. On the other hand, *perishable* resources are those that lose value when held over an extended period of time. In this case, these resources cannot be stored for a later use, but have to be immediately used. An example of a perishable resource is communication bandwidth, since it only has value if it is used, but it cannot be stored for future increases in the bandwidth of a network connection. On the other hand, an example of durable resources are artistic paintings, since they do not never lose their value.

- **Static vs renewable**: A *static* resource is one that is assigned once, and thereafter its assignment does not change. On the other hand, *renewable* resources are only assigned for a given period of time, after which they must be reassigned again. This allows to model temporary access to limited and shared resources such as for example CPU time.

- **Divisible vs indivisible**: This feature defines whether the resource can be indefinitely divided into as many units as desired (*divisible* resource) or it cannot be divided (*indivisible* resource). In this latter case, however, there may be multiple units of the resource (each of them being indivisible). An example of a divisible resource is the fuel.

- **Controlled vs non-controlled**: Although usually the resources belong to an owner that controls the access to them, in some domains the resources are somehow "public", meaning that anyone can use them, even without having been authorized to do so. This *uncontrolled access* makes the resource allocation problem harder. Examples of non-controlled resources can be found in natural resources, which are usually accessible to anyone, and for which there are no physical means of controlling the access of the agents to them.

Regarding the agents, they are the *entities* that receive the resources after they sent a request for getting them. In some domains, as for example scheduling, we will not refer to them as agents but as *tasks*, which are the ones that make use of the resources. An assignment of resources to agents (or tasks) is called an *allocation*. The set of resources allocated to an agent is also called in some domains a *bundle*.

There are many variations of resource allocation problems, for example assigning also the time at which the resource has to be used by the agent (or when a task has to be performed), minimizing the cost of the allocation (given that each allocation of a resource to an agent has an associated cost), leveling the usage of the resources, etc. Regarding the restrictions, they usually include the capacities of the resources, which limits the number of agents (or tasks) that can be simultaneously using a single resource, precedence relations that force some tasks to be performed before or after other tasks, and deadlines establishing a maximum time at which a task must be performed.

The two most known classes of resource allocation problems are *scheduling* and *auctions*. Scheduling is the problem of assigning a set of tasks to a set of resources, usually minimizing the total time required to perform all the tasks (makespan). It has many industrial applications such as optimizing production processes, transportation timetables, employees schedules, CPU utilization, video broadcasting, etc. On the other hand, in the case of auctions the agents are assigned the available resources in an economic way, trying to maximize the auctioneer's revenue.

The objective of resource allocation problems is either to find a feasible solution (e.g. an allocation of tasks to resources that guarantees that all the restrictions are satisfied) or to find the optimal solution given an objective function. In the latter case the objective function can be focused either on the entity that is performing the allocation (e.g. the auctioneer in a combinatorial auction) or in the entities that are requesting the resources, given an aggregation function of their individual preferences (e.g. an allocation of resources maximizing the average utility obtained by the entities). This aggregation of individual preferences can be modeled using the concept of *social welfare* as studied in Welfare Economics and Social Choice Theory. The two most used social welfare aggregations functions are, *utilitarian*, where the aim is to maximize the sum of individual utilities, and *egalitarian*, where the goal is to maximize the welfare of the entity that is getting less [12]. In the case of scheduling the main objective is usually to find the optimal solution such that the total makespan is minimized. In auctions, the objective is usually to maximize the revenue of the auctioneer. However, in recurrent versions of auctions and scheduling problems social welfare measures can be taken into account.

There are two main approaches for solving resource allocation problems (both scheduling and auctions). The first one is by using a *totally centralized* approach based on classical artificial intelligence techniques. The main drawback of this approach is that the central entity has all the power. That is, it makes all the decisions, and therefore does not let the agents take part in the obtention of the

allocations [1].

To allow for a more participative mechanism, a distributed approach can be taken. In this case, the agents can interact with the central entity, or between them, in order to reach an agreement on the allocations. This approach can be divided into two main classes, depending on who is the responsible of decision making:

- *Centralized approach*: In this approach, as in the classical one, there is a single entity that solves the problem. However, there is some interaction between the agents and the decision maker, so that the decisions are not totally taken by the latter. An example of such approach are auctions [73, 14], where agents bid for using the resources and the auctioneer decides which agents can do so.
- *Decentralized approach*: In this approach there is no central entity, but the agents themselves are the ones solving the scheduling problem. They communicate with each other in order to reach a solution. This approach includes negotiation protocols, in which agents trade resources until they are all satisfied with the allocation [40], and also distributed constraint optimization problems (DCOP) [50, 64].

There is also a lot of literature regarding preference representation. In the context of resource allocation, preferences express the satisfaction of an agent when deciding between different potential allocations of the resources, i.e. the different bundles of resources received by the agent. As the set of alternatives in resource allocation problems is exponential in the number of resources (one for each combination of resources), it is not reasonable to ask the agents explicitly for the entire preference list. For this reason, languages for preference representation are used [12].

Resource allocation problems are closely related (and equivalent in some cases) to matching, knapsack and set packing/covering problems, consequently falling in the class of *NP-Complete* problems (in its decision version), and therefore when developing methods for resource allocation problems their complexity needs also to be taken into account.

In our work we deal with resource allocation problems where a set of agents compete for a set of resources. However, as we focus on the robustness of the allocation mechanism, time constraints are not the crucial point, and therefore we choose auctions as the framework to deal with RAPs instead of scheduling. For our purposes, the resources in the auction are durable, static, indivisible and controlled. We will use a centralized approach where the central unit is the auctioneer which wants to maximize his revenue while producing robust allocations, and the agents inform their preferences through the bids that are composed by the items (or combinations of items) that they want.

In the following section we give a background on auctions (entering in more detail on combinatorial auctions) and its challenges.

2.2 AUCTIONS

Auctions are an ancient economic mechanism designed to trade items between individuals where the values of the items are not precisely known. In an auction, a seller (the *auctioneer*) offers the

[1] In appendix A a concrete resource allocation problem is analyzed, modeled and solved using totally centralized techniques, comparing their performance

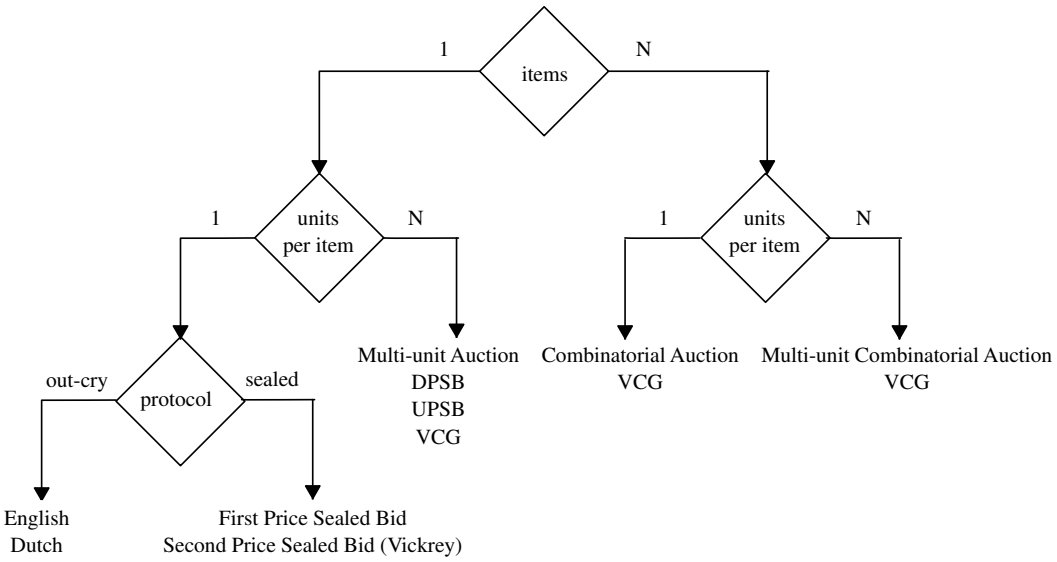

Figure 2.1: Auctions classification

items, and a set of buyers (the *bidders*) notify their interest on them by submitting *bids*, composed by the desired item/s and the price that the bidder is willing to pay for it/them. Once all the bids are received, the auctioneer selects the (set of) winning bids, which in turn determines the prices of the items that have been sold.

Auctions 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, industrial scheduling, and memory allocation in operating systems. Auctions are currently being used in several industrial scenarios [6], such as the electricity market, in which different kinds of energies are auctioned in order to favour the use of non-pollutant sources of energy [57]. One of the reasons for the increased popularity of auctions is the perceived improvements in efficiency and revenue for the seller.

2.2.1 Auction Mechanisms Classification

Several types of auctions have been defined, a partial diagram of them can be seen in Figure 2.1. Based on the number of items being offered, auctions can be classified as either single-item or multi-item auctions. The former are the most common, where bidders compete for a single good. There exist quite a few protocols for them, being the most common types: English, Dutch, First Price Sealed Bid and Vickrey.

In an *English auction*, also called an open-outcry ascending-price auction, the auctioneer begins the auction at the reserve price (the lowest acceptable price); then the bidders are free to raise their bid,

which must be higher than the last bid price. When no more bids are risen the winner is the last (highest) bidder which pays the price he declared. This is the typical auction used, for instance, to sell artistic works. In a *Dutch auction*, also known as clock auction, or open-outcry descending-price auction, the auctioneer lowers the price until a bidder takes it (or a minimum price is reached). The first bidder to speak wins, paying the last announced price. This type of auction was first used to sell tulips in the Netherlands, and has been extensively used for many years also in fish markets. In *First Price Sealed Bid Auction (FPSB)*, each bidder submits a bid without knowing the other bidders' bids. The highest bid wins, paying the price he submitted. This differs from English auction because as bids are not open or called, bidders must submit valuations based on an estimation of the market value of the item and their own willingness to pay, as opposed to competing through relative prices with other bidders. *Vickrey auction* [71], also known as *Second Price Sealed Bid Auction (SPSB)*, is quite similar to first price sealed bid, but here the winner bidder pays the second highest price submitted. This small alteration, however, has important theoretical implications, as it gives bidders an incentive to bid honestly, meaning that the best strategy for the bidders is to reveal their true values for the items, which does not happen in FPSB.

When the quantity of the items being sold is greater than one (multiple copies of each item), auctions are called multi-unit. Single-item auctions with multi-unit items are differently classified based on the pricing rules. For example, in a *Discriminatory Price Sealed Bid* (DPSB) auction, all the winners pay their bid price. Alternatively, in a *Uniform Price Sealed Bid* (UPSB) auction, all winners pay the same price which is the highest bidding price of the losers.

Multi-item auctions are known as Combinatorial Auctions (CA), which are the kind of auctions that we will basically use in our work. In this kind of auctions, bidders can place bids on more than one item at the same time. We can also have multi-unit items in a combinatorial auction, turning it into a Multi-unit Combinatorial Auction (MUCA). We will describe combinatorial auctions in detail in section 2.3, however an even more extensive study on combinatorial and multi-unit combinatorial auctions can be found in the book by Peter Cramton, Yoav Shoham and Richard Steinberg [16].

2.2.2 Auction Phases

In an auction we can distinguish three phases: the *bidding phase*, the *auction clearing phase* and the *transaction phase*. In the bidding phase, the bidders submit their bids according to the interest they have on the objects that are being auctioned. After that, in the auction clearing phase, the auctioneer determines the set of winning bids that maximizes his revenue. Finally, in the *transaction phase* the items are delivered to the corresponding bidders.

Figure 2.2 shows a diagram of the phases of an auction, where the transaction phase contains a long period of time where unexpected changes can occur. In standard auctions the transaction phase is not considered as it is instantaneously made after the auction clearing, but there exist other domains in which this phase can last for weeks or even months. It is in this situations where finding a robust solution is particularly important.

Coming back to the example presented in the introductory chapter about assigning rooms for simultaneous conferences, in the first phase the organizers would send their bids for (possibly multiple) combinations of rooms according to their requirements. The organizers would have a deadline to

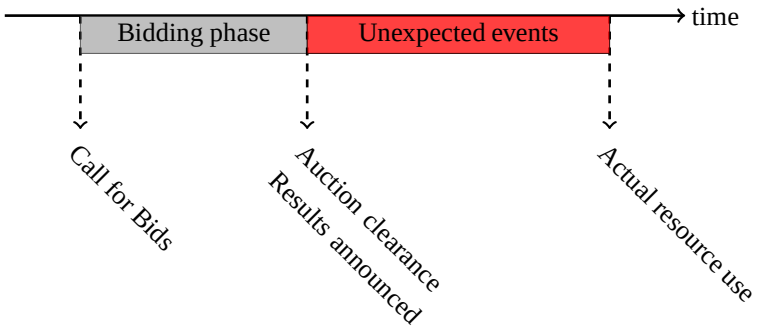

Figure 2.2: Auction phases.

send the bids, which would be typically some weeks before the conference starts. After this deadline, with all the received bids, the second phase would begin and the auction would be solved and the results announced to the participants. But after that, there would be a period of time before the rooms are actually used where some unpredictable events could happen. At the end of this second phase, the conference would begin and only the available rooms would be used by the participants (transaction phase). However, as some events could happen between the auction clearing phase and the transaction phase, it is possible for the solution previously found to be not valid. Therefore, solution robustness would be useful in this case, since during the transaction phase some breaks can make the solution previously found invalid.

2.2.3 Incentive Compatibility

In this section, we consider the *truthfulness* of mechanisms for combinatorial auctions. This is an interesting feature that is usually considered when designing mechanisms for auctions (and combinatorial auctions). This means whether it is possible or not to design a mechanism for the auction such that it is in the best interest of the bidders to send bids that truthfully reveal their preferences [43]. Auction mechanisms that have the feature of incentivizing bidders to bid truthfully are called *incentive-compatible*, or *strategy-proof*. On the other hand, non-incentive-compatible mechanisms allow the possibility that the bidders strategically manipulate the auction in order to decrease the final price of the items, with the consequent gain for them (and loss for the auctioneer). Incentive-compatible mechanisms are important in auctions because it is known that no untruthful mechanism achieves better outcome than any truthful (non-manipulable) mechanism. For that reason, developing auction mechanisms fulfilling this requirement is currently a major concern for auction designers, and a lot of research has already been taken by the computer science community in this direction [55, 54, 43]. In many applications of auctions, only mechanisms assuring that the bidders will bid truthfully are considered.

The first auction mechanism that was proved to be incentive compatible was the Vickrey auction [71] which, despite its simplicity, accomplished the theoretical definition. The Vickrey auction was then extended in order to deal with multi-unit and combinatorial auctions; resulting in the *Generalized*

Vickrey Auction, also known as VCG[2]. The drawback of VCG auctions for both CA and MUCA is that they are much more computationally expensive, as the price paid by each winner *k* is computed by deducting the sum of payments of all the other bidders in the current solution 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. This requires to solve $q + 1$ optimization problems (where the optimal solutions are composed by q bids). Therefore, and given that relaxations of the model are not a good option since they compromise truthfulness [55], this method is not generally used in practice in large problems due to its intractable computation time.

Incentive compatibility is hard to prove in complex auction mechanisms. For a mechanism to be truthful it has to satisfy the conditions of *Exactness*, *Participation*, *Critical*, *A-Monotonicity* and *P-Monotonicity* [43]. Informally, exactness means that each bidder either gets exactly the set of goods he requests or nothing, critical means that each winning bidder pays the lowest value he could have declared and still be allocated the goods he requested, participation means that bidders getting nothing pay zero. Finally, regarding monotonicity conditions, a method is said to be monotone if and only if each bid from the solution that increases its price still continues in the solution, provided that all the other bids remain fixed. This condition can be used conversely for proving non-monotonicity [37].

The strategy-proofness of the approach that we will present later in this work for dealing with robustness will be analyzed in Chapter 6 in the same way, using this necessary condition of monotonicity that incentive compatible mechanisms must hold.

2.3 Combinatorial Auctions

In this thesis, we focus on Combinatorial Auctions, as they are able to encode resource allocation problems. These kind of auctions were first proposed by Rassenti, Smith, and Bulfin in 1982 [60], for the allocation of airport landing slots. In a combinatorial auction, bidders can bid on bundles (combinations) of multiple distinguishable items instead of just individual items. This allows the bidders to be more expressive in the valuations of the items. The information contained in the bids is composed by the desired subset of objects together with the price that the bidder is willing to pay for it.

In recent years combinatorial auctions have emerged and grown rapidly as a popular mechanism for the sale of a set of items among which bidders perceive dependencies between the goods. The most important dependencies are *complementarities* and *substitutabilities* [65, 24]:

- **Substitutability**: A bidder's value of getting various goods is less than the sum of the 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 bidder's value of getting various goods is greater than the sum of the values for each individually (e.g., they are at least partially co-dependent). For example, in the case of a suit, a bidder can still think of buying the jacket and the pant separately but it is certainly more valuable to buy the jacket and the pant together.

[2]Where "V" stands for Vickrey [71], "C" for Clarke [13], and "G" for Groves [27], the three researchers that created the generalized versions of the Vickrey auction.

This increase in expressiveness that combinatorial auctions provide allows more economical allocations of the items, since the bidders do not obtain undesired partial bundles of low value.

To solve the auction (allocate the items to the bidders), the auctioneer gets the set of price offers for various combinations of goods coming from the bidders, and his aim is to allocate the goods in a way that maximizes his revenue or, in other words, the auctioneer selects a set of these bids that provides him the highest revenue without assigning any item to more than one bidder. The problem of selecting the optimal set of bidders to allocate the goods, known as the Winner Determination Problem, has a high computational complexity compared with single-item auctions, as we will see in section 2.3.2.

2.3.1 EXAMPLE

In order to understand the importance of the superior expressivity that combinatorial auctions provide, consider a very simple example of a combinatorial auction with three bidders (b_1, b_2, b_3), and two items for sale, A and B. The first bidder (b_1) is only interested in the item A, the second bidder (b_2) is only interested in the item B, and the third bidder (b_3) is interested in both items A and B, however it is not interested in receiving only one of those items. The first two bidders are willing to pay 10 each for the respective items, and the third bidder is willing to pay 18 for both, as seen in Table 2.1. Therefore, the optimal solution for the auctioneer is to sell the item A to the bidder b_1 and the item B to the bidder b_2, achieving a total revenue of 20, instead of selling both items to the third bidder which would achieve a total revenue of only 18.

Bidder	A	B	AB
b_1	10		
b_2		10	
b_3			18

Table 2.1: Combinatorial auction example

The problem of simple (non-combinatorial) auctions is that the auctioneer cannot know in advance whether it is better to offer the items separately or as a pack. For example, if the auctioneer had sold the items A and B together as a pack (in a single-item auction), then the revenue would have been only 18 (from the third bidder) because the first two bidders would have not bid for the pack. With these prices it would be better for the auctioneer to offer the items separately. On the other hand, if the third bidder offer was 22 instead of 18, then it would be better for the auctioneer to offer them as a pack instead of separately, since he could get the offer of 22 which is higher than the sum of the other two bids (20).

Therefore, given that it is not possible for the auctioneer to know in advance whether it is better to sell the items together or separately, a combinatorial auction where bidders can submit bids on any possible combination of items is the best option for him.

2.3.2 The Winner Determination Problem

The *Winner Determination Problem (WDP)* of a combinatorial auction, also called the *auction clearing algorithm*, is roughly defined as: given a set of bids in a combinatorial auction, select the winning bids that maximize the seller's revenue, subject to the constraint that each good cannot be allocated more than once. The formal definition is as follows.

Let $\mathcal{G} = \{g_1, g_2, ..., g_m\}$ be a set of goods, and let $\mathcal{B} = \{b_1, b_2, ..., b_n\}$ be a set of bids. Each bid b_i is a pair (p_i, G_i) where $p_i \in \mathbb{R}^+$ is the price offer of bid b_i and $G_i \subseteq \mathcal{G}$ is the set of goods requested by b_i. For each bid b_i a binary indicator variable x_i is defined to encode the inclusion or exclusion of bid b_i from the allocation, i.e. whether b_i is winner (1) or loser (0). Then, the single-unit WDP is the following constraint optimization problem (COP):

$$\max \sum_{i=1}^{n} x_i \cdot p_i$$

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

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

$$\max \sum_{i=1}^{n} x_i \cdot p_i$$

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

The WDP is equivalent to the weighted set-packing problem, the knapsack problem and the maximum weighted clique problem[3], and its decision version is therefore *NP-Complete* even in its single-unit variant (see e.g., [62]). 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* [65].

The above problem formulations assume the notion of *free disposal*. This means that in the optimal solution not all of the items have to be mandatorily sold. Otherwise, if it is required for all the items to be sold, the inequalities ($\leq$) should be changed by equalities ($=$); then the problem becomes equivalent to the Set Partition Problem [20], which is *NP-Complete* as well.

[3] 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.

2.3.3 Experiments with Combinatorial Auctions

Most of the literature on combinatorial auctions performs the respective tests using the benchmark for combinatorial auctions developed by Kevin Leyton-Brown *et al.* [44] called "Combinatorial Auctions Test Suite" (CATS). Since its first release in 2000, CATS has become the standard tool for evaluating and comparing WDP algorithms [66, 16]. It generates realistic combinatorial auction instances, following a set of real-world economically motivated scenarios as well as many previously published distributions (called legacy). Setting a given number of goods and bids, the program generates the set of bids by selecting which goods to include in each bid following the chosen distribution.

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 these distributions 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 [44]:

- **Paths**. This distribution models auctions regarding shipping, rail and bandwidth problems. Goods are represented as edges in a nearly planar graph, with agents submitting a set of bids for paths connecting two nodes.
- **Arbitrary**. 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**. 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.
- **Scheduling**. This distribution generates bids for a distributed job-shop scheduling domain, and also its application to power generation auctions.
- **Regions**. This distribution models an auction of real state, or more generally of any goods over which two-dimensional adjacency is the basis of complementarity, e.g. spectrum rights or property. Again, the relationship between goods is represented by a graph, in this case strictly planar.

The "legacy" distributions are the following [44]:

- **L1**, the *Random* distribution from [65], chooses a number of items uniformly from [1,m], and assigns the bid a price drawn uniformly from [0, 1].
- **L2**, the *Weighted Random* distribution from [65], chooses a number of items g uniformly from [1, m] and assigns a price drawn uniformly from [0, g].
- **L3**, the *Uniform* distribution from [65], sets the number of items to some constant c and draws the price offer from [0, 1].
- **L4**, the *Decay* distribution from [65] starts with a bundle size of 1, and increments the bundle size until a uniform random value drawn from [0, 1] exceeds a parameter α.

- **L5**, the *Normal* distribution from [38], draws both the number of items and the price offer from normal distributions.
- **L6**, the *Exponential* distribution from [24], requests g items with probability $C{\cdot}e^{-g/q}$, and assigns a price offer drawn uniformly from [0.5g, 1.5g].
- **L7**, the *Binomial* distribution from [24], gives each item an independent probability of p of being included in a bundle, and assigns a price offer drawn uniformly from [0.5g, 1.5g] where g is the number of items selected.
- **L8**, the *Constant* distribution with 3 goods per bid, with a quadratic calculation for the prices.

In our work we will mostly use the set of realistic instances and only one of the legacy distributions (L7), to examine the effects of robustness in various possible applications of combinatorial auctions.

2.3.4 Optimal Algorithms for Combinatorial Auctions

Since the problem of finding the optimal solution to a combinatorial auction is *NP-Hard*[4], any optimal algorithm for the problem will be slow on some problem instances. However, in the last years considerable research has been done in the combinatorial auction winner determination problem. For a more extended survey, see [20] and [16]. We will briefly describe only some of the most known specific algorithms for solving combinatorial auctions that find the exact optimal solution, which are actually able to solve quite large instances in practice. We will also give details on how to model combinatorial auctions with Integer Linear Programming to be run with a generic commercial LP solver as CPLEX [15], which has nowadays become the generally used solving method for CAs.

CASS

One of the first specific solvers for combinatorial auctions was CASS (Combinatorial Auction Structured Search) [24], developed in the Stanford University by Yuzo Fujishima, Kevin Leyton-Brown and Yoav Shoham. It used a clever branch and bound search algorithm with dynamic programming and caching techniques that allowed to solve quite large problems in practice.

The crucial detail about CASS is that it structures the search space using *bins* (see Figure 2.3). A bin is created for each good, and every bid is placed into the bin corresponding to its lowest-order good. Instead of 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. To treat the possibility that the auctioneer's revenue can increase by keeping items, *dummy* bids of price zero are placed on those items that received no 1-item bids. 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.

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

[4] *NP-Complete* in its decision version, i.e. deciding if a solution exists.

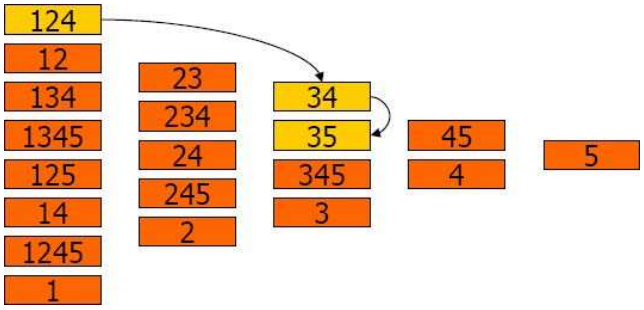

Figure 2.3: Partition into bins.

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. 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[5].

BIDTREE

Bidtree [65] is the other special-purpose WDP algorithm that has been most 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

Researchers soon realized that combinatorial auctions could be easily converted into an integer programming problem, taking advantage of the astonishing improvements that such solvers, and spe-

[5] http://www.cs.ubc.ca/∼kevinlb/downloads.html

cially CPLEX, was including in the last versions of its mixed integer programming module. Nowadays, CPLEX is the default (and fastest) approach for solving the WDP.

The Winner Determination Problem can be easily modeled as an Integer Programming Problem. To do 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). The constraint optimization problem is the following:

$$\text{maximize f} = \sum_{i=1}^{n} p_i \cdot X(i) \tag{2.1}$$

$$\forall g \in G \sum_{i \in C_g} X(i) \leq 1 \tag{2.2}$$

where n is the number of bids, and C_g is the set of bids containing item g. 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. Version 10 (2006) improved the time to optimality by an average of 30% and improvements average 70% for particularly difficult models. CPLEX 11 introduced a new search algorithm, dynamic search, while retaining its conventional branch-and-cut algorithm, but with advances in branching, cuts and heuristics. By selecting the more efficient of the two search strategies, CPLEX 11 improved the time to optimality by 15% on average for models solved in less than one minute, three times faster on average for models in the range of one minute to one hour, and for hard models requiring more than one hour to solve, the speed up was a factor of ten on average. CPLEX 12 (released in 2009) is a 10% faster in large problems, but the best good new was the for the first time, it was available freely for academic purposes.

Another possibility is to use the free open-source solver GLPK (GNU Linear Programming Kit) [26]. Although ILOG claims that its CPLEX solver is 100 times faster than GLPK, it is lighter and enough for solving not-hard medium-size instances.

CABOB

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 [66]. The CABOB (Combinatorial Auction Branch on Bids) algorithm is a depth first branch and bound search with linear relaxations 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 2.4. 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 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).

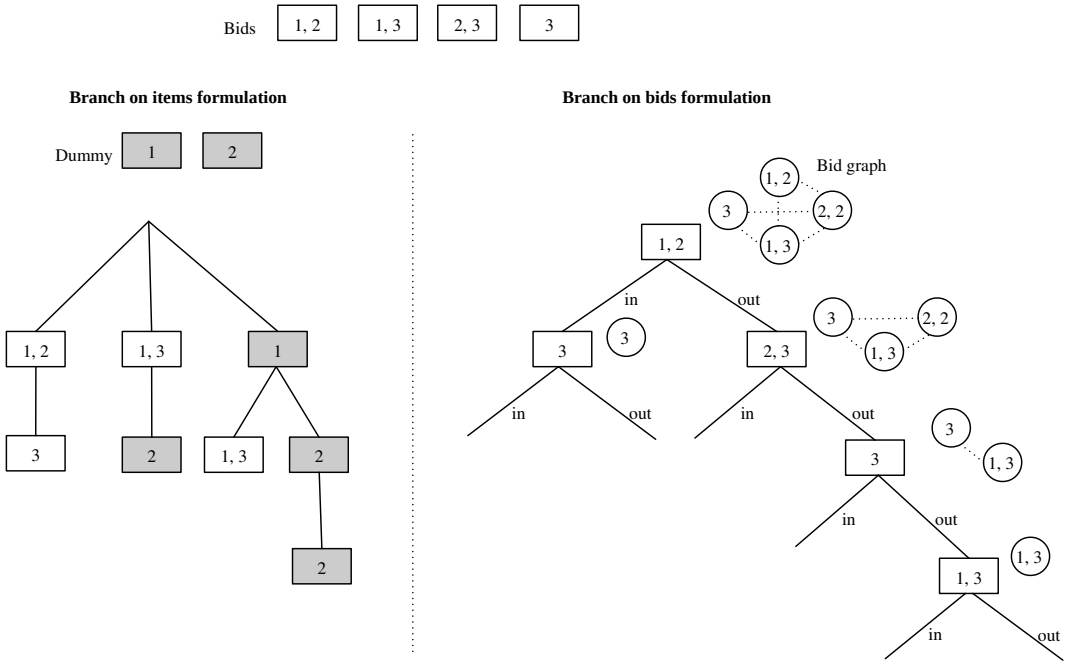

Figure 2.4: Branch on items (left) versus branch on bids (right) formulation. Figure extracted from [16].

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 linear programming as a subroutine and uses a similar search strategy.

CABRO

We should also mention the algorithm CABRO, that was developed during this research, and published in 2008 [51]. CABRO (Combinatorial Auction BRanch and bound Optimizer) is mainly a branch and bound depth-first search algorithm with a specially significative polynomial-time procedure to reduce the size of the input problem. The algorithm is divided in three main phases:

- The first phase performs a fast preprocessing (polynomial time) with the aim of removing as many bids as possible. Bids removed in this phase may be either bids that *cannot* be in the optimal solution, or bids that *have to* be in the optimal solution.
- The second phase consists in calculating upper and lower bounds for each bid. The upper bound of a bid is computed by formulating a relaxed linear programming problem (LP), while the lower bound is computed generating a solution quickly. 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. In this phase the two previous phases are used also as heuristics and for pruning.

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. 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.

The algorithm was compared against CASS, GLPK and CPLEX, beating clearly CASS and GLPK in average, and being competitive with CPLEX. More details about the algorithm are given in Appendix B.

2.3.5 Summary of combinatorial auctions solvers

We have described five different methods to solve a combinatorial auction. Table 2.2 shows a comparison of these methods, focusing on the following characteristics:

- *Performance*. How fast the algorithm ends giving the optimal solution.

- *Anytime performance*. How fast the algorithm produces a valid solution.
- *Input & output*. This 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*. The price of the software.

Regarding the overall performance, CPLEX is clearly the best product followed by CABOB, CABRO and GLPK, with CASS at some distance, and finally BidTree. However, 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	Free**
GLPK	Fast	Bad	Transformation	Fast	Free
CABOB	Fast	Bad*	Direct*	Slow*	Unavailable
CABRO	Fast	Good	Direct	Fast	Avail. under demand

*Unknown (presumed values).
**For academic purposes.

Table 2.2: Auction solvers comparative.

For academic purposes there is no doubt that CPLEX is the best option. However, for other industrial applications CPLEX could be quite expensive. Therefore, CASS and CABRO should be the first options to try, as they are easy to use (receiving as input directly the list of bids). If they were not able to solve the problems because of its large size, then a transformation to LP should be considered to test whether GLPK is able to solve it or not. Otherwise, CPLEX would be considered if its cost could be afforded.

2.4 Robustness

Previous sections have introduced resource allocation problems and auctions, since they are the kind of problems that we will deal with. In this section, we will give some background on robustness

in general, and the following section will talk more concretely about robustness in combinatorial auctions.

There are two general approaches for dealing with robustness: reactive and proactive. Whereas *reactive* techniques address the problem of how to recover from a disruption once it has occurred, *proactive* methods construct solutions that are inherently robust (up to a given degree) to uncertainty in the data.

Kentaro Tsuchida [70] presented a reactive 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.

On the proactive field, Andrew J. Davenport proposed a slack-based technique for robust scheduling [18]. The key idea of slack-based techniques is to provide each activity with extra time to execute so that some level of uncertainty can be absorbed without rescheduling.

Another usual way of achieving proactivity is by using supermodels [25] and super solutions [30], which are defined by two parameters, *a* and *b*. The parameter *a* specifies the maximum size of the break that the robust solution is able to absorb, and the parameter *b* sets the maximum size of the repair needed to fix the solution whenever a break occurs. Whereas slack-based techniques are widely used in scheduling problems, supermodels and super solutions are mostly used in Boolean satisfiability and Constraint Programming problems, respectively.

In our work we will focus mainly in proactive approaches. Supermodels [25] were defined to find robust solutions based on repairs for propositional satisfiability (SAT) formulas. Later, they were extended to super solutions [30] in order to find robust solutions for constraint programming problems. After that, weighted super solutions [35] were introduced in order to handle more easily failure probabilities and costs of the repairs.

In our work we will extend mainly the work on supermodels for SAT formulas, adding some concepts from weighted super solutions. The resulting formulation will need to move from SAT to a much richer framework, known as SAT modulo theories (SMT). Therefore, we will first give a brief introduction on SAT and SMT; later we will describe the supermodels for SAT, super solutions for CP and weighted super solutions. This will give us a good background on the techniques that we will use in the central part of this thesis.

2.4.1 PROPOSITIONAL SATISFIABILITY

A *propositional variable* is a variable whose value can be either $true$ or $false$. A *propositional formula* (or *Boolean formula*) over a set of propositional variables P is any variable $p \in P$ or a negation $(\neg F_0)$, a disjunction $(F_0 \vee F_1)$ or a conjunction $(F_0 \wedge F_1)$ of smaller formulas F_0 and F_1 (note the parentheses). The number of parentheses can be reduced by introducing precedence rules, commonly giving highest priority to $\neg$ and lowest priority to $\vee$. Also, other connectives can be used as abbreviations, e.g., $p \rightarrow q$ for $\neg p \vee q$, and $p \leftrightarrow q$ for $(p \rightarrow q) \wedge (q \rightarrow p)$.

A variable p is an *atom*, and a variable p or its negation $\neg p$ is a *literal*. A *clause* is a disjunction of literals $l_1 \vee \cdots \vee l_n$. A formula is in *conjunctive normal form* (CNF) if it is written as a conjunction of clauses $C_1 \wedge \cdots \wedge C_m$. CNF formulas are sometimes denoted as a set of clauses $\{C_1, \ldots, C_m\}$.

An *interpretation* (or *truth assignment*) I for a formula F is a function mapping the variables of F to $\{true, false\}$. An interpretation I *satisfies* a formula F, denoted $I \models F$, if under this interpretation of variables and the usual truth table interpretation of the logical connectives, the formula F evaluates to $true$. An interpretation I satisfying a formula F is called a *model* of F. A formula F having some model is called *satisfiable*, and *unsatisfiable* otherwise.

Satisfiability (SAT) is the problem of determining the satisfiability of a propositional formula F [7]. The formal definition of SAT actually requires the formula to be expressed in CNF. Besides the standard SAT problem, some variants have been defined. Max-SAT is the problem of finding the maximum number of clauses that can be satisfied by any truth assignment. Weighted Max-SAT is a variant of Max-SAT where every clause has a weight, i.e., a weighted Max-SAT formula is a conjunction of weighted clauses of the form (C, w), where C is a clause and w is a natural number indicating the cost (weight) of the falsification of C. The cost of a truth assignment for the formula is the sum of the costs of the clauses falsified by this assignment. Given a weighted formula, weighted Max-SAT is the problem of finding a truth assignment with minimal cost. A particular case of weighted Max-SAT is *partial weighted Max-SAT* where some clauses are mandatory (have an infinite cost of falsification) and the other clauses are weighted. Table 2.3 shows schematically the different variants of propositional satisfiability problems.

Type	Formula
SAT	$(p \vee q) \wedge (\neg p \vee q) \wedge (p \vee \neg q) \wedge (\neg p \vee \neg q)$
	Result: **unsat**
Max-SAT	$\mathbf{(p \vee q)}\wedge\mathbf{(\neg p \vee q)}\wedge\mathbf{(p \vee \neg q)}\wedge(\neg p \vee \neg q)$
	Result: Maximum number of satisfied clauses (**3**)
Weighted Max-SAT	$\mathbf{(p \vee q, 10)}\wedge\mathbf{(\neg p \vee q, 10)}\wedge\mathbf{(p \vee \neg q, 10)}\wedge(\neg p \vee \neg q, 10)$
	Result: Max sum of weights of satisfied clauses (**10**)
Partial Weighted Max-SAT	$\mathbf{(p \vee q)}\wedge\mathbf{(\neg p \vee q)}\wedge\mathbf{(p \vee \neg q, 10)}\wedge(\neg p \vee \neg q, 10)$
	Result: sat/Maximum sum of weights of unsat clauses (**10**)

Table 2.3: Propositional satisfiability variants. Satisfied clauses are shown in boldface.

If F and F' are two formulas such that F' is true in all models of F, then we say that F' is a *logical consequence* of F, or *logically follows* from F, or that F *logically implies* F', and we denote it by $F \models F'$. Two formulas F and F' are said to be *(logically) equivalent*, written $F \eqiv F'$, if, and only if, they have the same truth value in each interpretation (i.e., are either both $true$ or both $false$), in other words, if $F \models F'$ and $F' \models F$.

2.4.2 Satisfiability Modulo Theories

Satisfiability Modulo Theories (SMT) is a generalization of SAT in which some propositional variables have been replaced by predicates with predefined interpretations from background theories. For example, a formula can contain clauses like, e.g., $p \vee q \vee (x + 2 \leq y) \vee (x > y + z)$, where p and q are Boolean variables and x, y and z integer ones. Predicates over non-Boolean variables, such as

linear integer inequalities, are evaluated according to the rules of a background theory. Examples of theories include equality, linear (integer or real) arithmetic, arrays, bit vectors, etc., or combinations of them.

Formally speaking, a *theory* is a set of first-order formulas closed under logical consequence. A theory T is said to be *decidable* if there is an effective method for determining whether arbitrary formulas are included in T.

The SMT problem for a theory T is: given a first-order formula F, determine whether there is a model of $T \cup \{F\}$. Usually, T is restricted to be decidable and F is restricted to be quantifier-free so that, while providing a much richer modeling language than it is possible with propositional formulas, the problem is still decidable. The dominating approach to SMT is based on the integration of a SAT solver and a solver for the given theory T, being in charge respectively of the Boolean and the theory-specific components of reasoning. A survey on this so-called lazy approach can be found in [68].

It is also remarkable that state-of-the art SMT solvers have a rich input language, and it is not necessary (neither convenient) to translate any formula into a set of clauses (CNF format) in order they can read it.

SAT solvers are used for Bounded model checking, and AI planning among other things. Some of the most used SMT solvers are Z3 [19], Yices [21], Barcelogic [8] and MathSAT [10].

2.4.3 PSEUDO-BOOLEAN

A closely related problem to that of weighted Max-SAT is pseudo-Boolean optimization. *Pseudo-Boolean constraints* (PB-constraints) are linear constraints over Boolean variables, that is, constraints of the form $C_0 l_0 + ... + C_{n-1} ln - 1 \geq C_n$ where, for all i, l_i is a literal and C_i is an integer constant. A true literal is interpreted as 1 and a false literal is interpreted as 0, so that a truth assignment satisfies a PB-constraint if the sum of the C_i whose corresponding l_i is assigned to true exceeds or is equal to the right-hand constant C_n. Hence, PB-constraints can be seen as a generalization of clauses, that coincide with clauses in the case that all the C_i are 1.

The *pseudo-Boolean optimization* (PBO) problem consists in finding a satisfying assignment to a set of clauses that minimizes a given objective function of the form $\sum_{j=1}^{m} C_j x_j$, where C_j is a non-negative integer cost associated to the variable x_j. PBO is indeed a particular case of integer linear programming (ILP) which is known as 0-1 integer programming. Moreover, every PBO instance can be translated into a partial weighted Max-SAT formula, where each PB-constraint is translated into a set of mandatory clauses [22] and the objective function is translated into a set of non-mandatory clauses, each summand $C_j x_j$ becoming a unit clause ($\overline{x}_j$, C_j).

Pseudo-Boolean problems can be straightforwardly formulated as an integer program, in which the non-linear constraints are linearized. This idea is used by the solver glpPB, which applies GLPK [26] for solving the IPs. The solver BSOLO [46] combines integer programming techniques with SAT-solving. Another approach is from the point of view of constraint integer programming (CIP), the SCIP solver [4] is a combination of integer and constraint programming (CP) methods.

2.4.4 SUPERMODELS

Robustness for SAT formulas can be achieved by means of supermodels. The seminal work on robust solutions for propositional logic formulas is the one of [25], where the notion of *supermodel* was introduced. The complexity for finding such supermodels in several propositional logic fragments has been studied in [63].

The definition of a supermodel is (from [25]):

> *An* (S_1^a, S_2^b)-supermodel *of a Boolean formula* F *is a model of* F *such that if we modify the values taken by the variables in a subset of* S_1 *of size at most* ***a*** *(breakage), then another model can be obtained by modifying the values of the variables in a disjoint subset of* S_2 *of size at most* ***b*** *(repair).*

An (S_1^a, S_2^b)-supermodel in which the breakage (S_1) and the repair set (S_2) are unrestricted (but still disjoint) is denoted as an (a, b)-supermodel. The task of finding (a, b)-supermodels is NP-complete. The approach followed by [25] is to encode the supermodel requirements of a formula F as a new formula F_{SM} whose size is polynomially bounded by the size of F. This new formula F_{SM} has a model if and only if F has an (a, b)-supermodel.

For instance, the formula $F = p \vee q$ has three models, $\{p, q\}$, $\{\neg p, q\}$ and $\{p, \neg q\}$, which are all $(1, 1)$-supermodels given that if any variable changes its value (from $true$ to $false$ or viceversa) the formula is either still satisfied or can be satisfied by changing the value of the other variable. The encoding F_{SM} for a $(1, 1)$-supermodel of F, according to [25], would be:

$$\overbrace{\underbrace{(p \vee q)}_{\text{original } F} \wedge \Big(\underbrace{(\neg p \vee q)}_{\text{no repair}} \vee \underbrace{(\neg p \vee \neg q)}_{\text{repair } q} \Big)}^{\text{break in } p} \wedge \underbrace{\Big(\underbrace{(p \vee \neg q)}_{\text{no repair}} \vee \underbrace{(\neg p \vee \neg q)}_{\text{repair } p} \Big)}_{\text{break in } q}$$

Note that, for instance, if the satisfying interpretation (model) chosen for F_{SM} is $\{\neg p, q\}$, i.e., $\{p = false, q = true\}$, then $p \vee q$ is satisfied and, moreover, if q switches to $false$, then a new model for $p \vee q$ can be obtained by switching p to $true$. That is, a break in q has a repair on p. The key idea is that the value of the subformula $\neg p \vee \neg q$ under the initial interpretation coincides with the value of $p \vee q$ under the repaired interpretation and, hence, F_{SM} has a model if and only if F has a supermodel. Note also that only the first model $\{p, q\}$ is a $(1, 0)$-supermodel.

2.4.5 SUPER SOLUTIONS

The concept of (a, b)-supermodel for propositional logic was generalized to that of (a, b)-*super solution* in the context of Constraint Programming (CP) in [30]. An (a, b)-*super solution* is one

in which if at most a variables lose their values, the solution can be repaired by assigning these variables with new values and also changing the values of at most b other variables. The new values taken by the variables can be any other in its respective domains.

The paper [30] focused mainly on (1,0)-super solutions, since finding super solutions for values of a higher than 1 highly increases the complexity if the problem, and provided two alternative approaches for finding such robust solutions: either via reformulation or via search. The reformulation approach, called $P + P$, duplicates the variables. The duplicated variables have the same domain as the original ones, and have the same constraints. Additional constraints are added between each original variable and its duplicate so that they cannot have the same value. Figure 2.5 shows an example of reformulation with 3 variables (a, b and c) and 2 restrictions. An assignment to the original variables is a super solution, where the repair is given by the duplicated variables.

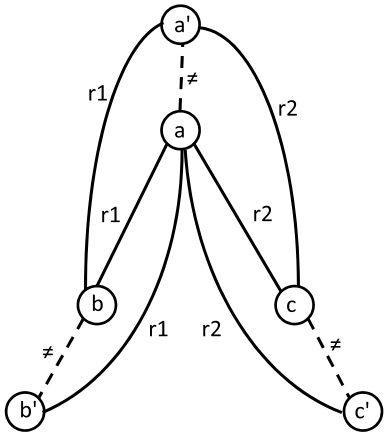

Figure 2.5: Example of (1,0)-super solution reformulation for CP.

This work also addressed the problem of finding the "most robust solution" for the cases that a robust solution does not exist. The most robust solution is a solution that maximizes the number of repairable variables. In a later paper [29] the same authors also addressed the problem of finding the "most robust optimal solution" and the "optimal robust solution".

2.4.6 Weighted Super Solutions

In order to deal with application problems such as real world combinatorial auctions the (α, β)-*weighted super solutions* [35] were developed, since super solutions were not expressive enough because they only considered the number of changed variables needed to repair a break. Instead, weighted super solutions introduce the breakage probability (α) and the cost of repair (β) to replace the parameters a and b of super solutions.

Weighted super solutions (WSS) were mainly applied to combinatorial auctions, and more con-

cretely to the problem of bid withdrawal. When a winning bid is withdrawn [6], there is a loss in revenue for the auctioneer, who is left with unallocated items that could be wanted by other losing bidders. There is therefore an opportunity for the auctioneer to reduce the resulting loss in revenue by reallocating these items to some of such losing bidders in a "repair" solution.

2.5 Robustness in Auctions

There are several works that deal with robustness with respect to potential manipulations of the auction mechanism, such as false-name bids [75, 48]. However, this is not the concept of robustness we are interested in.

As we have described before, we focus our research on robustness of the solution to the auction. Some works, such as [59, 58] add the concept of robustness (*fault tolerance*) to *mechanism design* in order to deal with potential failures in the execution of tasks by the agents. However, these works handle robustness using a probabilistic approach, and for them a robust solution is one that, on average, will perform well. Thus, since they rely on expected values, there could still be situations where a robust solution would perform badly. Nevertheless, their approach is appropriate in scenarios where there is no possibility of performing repairs in the solution. In contrast, we do consider the possibility of repairing solutions (e.g. reassigning goods) and hence in our approach we provide robust solutions that can be repaired in case any potential failure would arise.

Another closely related problem is that of robust knapsack [76] (i.e. a knapsack problem where the weights and/or values of the objects are imprecise). Given that many auction mechanisms can be modeled as a knapsack problem [39], it is reasonable to think that some of the robust approaches to this problem may yield robust solutions to auctions. However, the robustness concept used in the field of knapsack is somehow different to ours, since it does not consider the possibility of repairing a solution. Instead, a robust solution of a knapsack problem with imprecision is such that, on average, performs well regardless of what the actual weights or values of the objects are, in a similar way to the robustness presented in [59, 58].

As far as we know, the only previous work that has dealt with solution repair in combinatorial auctions is that of [36]. This work addresses the problem of bid withdrawal (i.e. a bidder that withdraws a winning bid), and, in order to find robust solutions, uses (α, β)-*weighted super solutions*. The concrete definition of a robust solution for a CA with WSS is (from [34]):

> *A robust solution for a combinatorial auction is one where any subset of successful bids whose probability of withdrawal is at least α can be repaired by reassigning items at a cost of at most β to other previously losing bids, in order to form a repair solution whose revenue is at least a fraction, γ, of optimal revenue.*

Our work is quite similar, since our approach is also based on supermodels and we look for solutions with a bounded cost. However, we consider the problem of resource unavailability, which is not considered in [36]. Moreover, we are also interested in keeping the number of repairs low, which is only done indirectly (through the cost function) in [36]. In addition, our techniques are completely different because we use the logic framework of weighted Max-SAT and Satisfiability Modulo Theories,

[6] The withdrawal of losing bids is not considered because there is no need for the auctioneer to change the solution since all items are already allocated.

while [36] presents an ad-hoc search algorithm to find robust solutions.

2.6 SUMMARY

In this chapter we have provided the necessary background on the topics that we shall use in the rest of this dissertation. We have identified combinatorial auctions as a natural and effective way to deal with resource allocation problems. The problem of resource unavailability appears in the application domains where the transaction phase is not instantaneous, and therefore some way of finding robust solutions is desirable. We have also identified that incentive compatibility issues are a major concern for auction designers and need to be taken into account.

Our approach for robustness is based on the concept of supermodel, complemented with some ideas from weighted super solutions. However, the model that we will propose is not plain SAT as in supermodels, neither CP as in super solutions and weighted super solutions. Instead, we will formulate it in SAT Modulo Theories (SMT), in order to take advantage of the recent advances that its solvers have achieved and will improve for sure in the next few years. SMT is closely related to Pseudo-Boolean, therefore, we have also introduced the necessary background on both areas.

The following chapter performs a sensitivity analysis, which will make clear the importance of incorporating robustness in combinatorial auctions that deal with uncertainty regarding resources becoming unavailable.

CHAPTER 3
Sensitivity Analysis

In this chapter we perform an analysis of the sensitivity of the optimal solutions in combinatorial auctions against resource unavailability. This analysis provides a strong motivation for our research, as it proves that breaks in resources may have a hard negative effect in the revenue of the optimal solution and, therefore, robust solutions instead would be more useful as they would be less affected.

3.1 INTRODUCTION

Resources becoming unavailable after a solution to an auction is found may produce negative effects, since the winning bids that contain such items must turn to losers and therefore their price is lost (deducted from the revenue). In this chapter we study how affected is the revenue against resource unavailability in a set of different scenarios (distributions) in what is called the *sensitivity analysis*. This analysis will make clear the importance of robustness in such kind of problems.

A similar analysis was performed in [34] for the case of bid withdrawal. The conclusion of that analysis was that robust solutions are needed in all the distributions (in ones more than in others). We perform an analogous analysis regarding resource unavailability, incorporating some additional experiments and results.

The sensitivity analysis is performed by running a large set of randomly generated instances using a combinatorial auction generator. We first solve the winner determination problem of each instance in order to find its optimal solution. Then we simulate resource unavailability by removing all the items in the auction one by one, in order to examine the effects of those breakages in the revenue. Whenever a resource is removed, it means that all the bids containing it are also removed, and in case that some bid was part of the optimal solution (it was a winning bid), then it is removed from the auction and consequently its price is deducted from the revenue. After that, we try to find a repair to that breakage in order to get as much revenue back as possible. The intuition is that the more bids participating in the auction, the greater revenue will be possible to be recovered. The objective of this analysis is to analyze the curve of percentage of optimality regarding the size of the instances, in a set of different distributions, in order to discover in which situations solution robustness would be more useful.

We have used the Combinatorial Auction Test Suite (CATS) [44] for generating all the auction instances. We have used the 4 “real-world” distributions: paths (representing transportation problems), arbitrary (for modeling electronic parts), matching (representing allocations for airline take-off and landing slots) and regions (for property and spectrum rights), plus one of the “legacy” distributions: L7 (the binomial distribution). For each distribution we have created 100 different instances with the number of items fixed to 20, and the number of bids ranging from 100 to 2000 (at intervals of

100). The other CATS flags used for the generation are “int_prices” and “bid_alpha = 1”.

We have generated instances with and without dominated bids. A bid is dominated (by another bid) 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 the set of items of b_i, $g(b_i)$ is included in the set of bids of b_j, i.e. $g(b_i) \subseteq g(b_j)$, and its price p is higher, i.e. $p(b_i) \geq p(b_j)$, b_j is dominated by b_i. Figure 3.1 shows an example of a dominated bid (b_1 dominates b_2). It actually means that dominated bids cannot appear in optimal solutions (that do not consider robustness) as they are never preferable to the bid that dominates them. This is the reason why CATS instances are generally created without dominated bids, and this is the case also in [34]. However, dominated bids could take part of an optimal robust solution and therefore we have made experiments also with them.

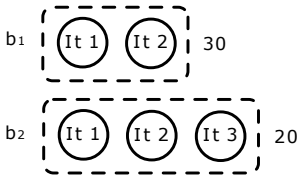

Figure 3.1: Example of dominated bids.

3.2 Reallocation and Full-reparability

Regarding the repairs we have examined two different schemes. On the one hand, we take the “reallocation” repairing mode of [34], which assumes that the auctioneer is unable to withdraw winning bids, and therefore the items becoming available after the breakage, which are the items of the bids that have been broken (in [34] because they have been withdrawn, and in our case because they contain items that have become unavailable), can only be assigned to (losing) bids requesting subsets of them. On the other hand, we consider the alternative of “full-repairability”, where the auctioneer has the full control and could withdraw winning bids if required, in order to construct a new solution as close to the optimal as possible.

The procedure for the sensitivity analysis that we perform is the following. First, each instance of a combinatorial auction (created with CATS) is converted to an integer linear program (CATS can be used as well to generate the ILP instance). After that, the optimal solution to the auction is found by using CPLEX ILP solver (version 12). Then we simulate resource unavailability by removing every single item of the auction and solving the resulting modified auction. For the “reallocation” setting the resulting modified auction is only composed by the bids containing subsets of the items that had been left free by the winning bid that included the removed item (causing the bid to be now loser). Hence, the revenue of the repair solution would be computed as the revenue of the original (with all the items available) optimal solution, deducting the price of the removed winning bid, and adding the revenue of the optimal solution of the modified auction. For the “full-repairability” setting, we simply delete all the bids containing the removed item and re-solve the modified auction. The revenue of the repair solution in this case is given directly by the revenue of the optimal solution of the modified auction.

Once the process is finished and all the revenues from optimal and repair solutions have been found, the percentage of optimality of the repair solution is computed as the revenue of the repair solution divided by the revenue of the original (with all the items available) optimal solution. The figures

below show two curves for each distribution, one displaying the average over all the instances and the other plotting the worst case. The X axis represents the number of bids of the instances and the Y axis is the revenue of the repair solution divided by the revenue of the optimal solution, i.e. the percentage of optimality.

In Figures 3.2 and 3.3 we observe the sensitivity analysis results for the "reallocation" scheme over all the distributions. We can observe how the curve of optimality of the repairs increases as the number of bids is higher, which was excepted given that the higher the number of bids, the easier is to repair any breakage, since there are a lot of bids to choose from. We also see that the differences between instances with dominated bids and without them, as we pointed out before, give slightly better optimality to instances that contain dominated bids, nevertheless the differences seem to be very low. Regarding the differences amongst the distributions, we notice that there are three distributions (arbitrary, regions and L7) that are clearly affected by resource unavailability while the other two (paths and matching) are not that affected.

Therefore, solution robustness seems to be specially useful for arbitrary, regions and L7-like instances on not very large problems, since arbitrary only achieves between 80% and 94% of optimality in average and between 65% and 88% for the worst case; and for regions, although it gives slightly better optimality, the values are between 82% and 94% for the average case and between 65% and 89% for the worst case. Conversely the L7 distribution gets the worst results with values between 5% and 13% in average and between 0% and 1% in the worst case. On the other hand, paths-like and matchings-like instances do not seem to require that much looking for robust solutions since the optimal solutions seem to be quite inherently robust, specially for large instances. We observe values between 91% and 92% for the average case and between 79% and 82% for the worst case in the matching distribution, and even better for paths with an optimality between 94% and 95% on average and between 92% and 93% in the worst case.

However, even in those distributions that seem to be not much affected by resource unavailability, there is a loss of revenue between 5% and 9% in average (and between 7% and 21% in the worst case), which is also enough to motivate the development of robustness techniques.

Figures 3.4 and 3.5 show the same results for the "full-repairability" scheme. We can appreciate the differences on the percentages of optimality that here are much higher than in the "reallocation" scheme, which was expected since the repair in this scheme has a wider range of action. In these cases, the percentage of optimality of the repair solutions in the arbitrary distribution goes between 91% and 95% on average and between 81% and 91% in the worst case, which is considerably higher than in the previous case. In the regions distribution the values go from 89% to 95% on average and between 76% and 91% in the worst case, which is again notably higher than in the reallocation scheme. For the L7 distribution the optimality goes from 92% to 94% in average and from 87% to 94% in the worst case, which are extremely better results than in the previous setting. For the instances that were not much affected in the previous case, here the results are quite similar. For the matching distribution we get values from 92% and 93% in the average cases and between 81% and 83% in the worst cases, which is only 1% or 2% higher than in the previous case. For the paths distributions the obtained values are 95% on average and between 92% and 93% in the worst case, which is pretty much the same results as before. Again, we can conclude that even in the most robust distributions, there is a loss of optimality of about 5% on average and 8% in the worst case, which could be sufficient in some domains to consider incorporating robustness. Obviously,

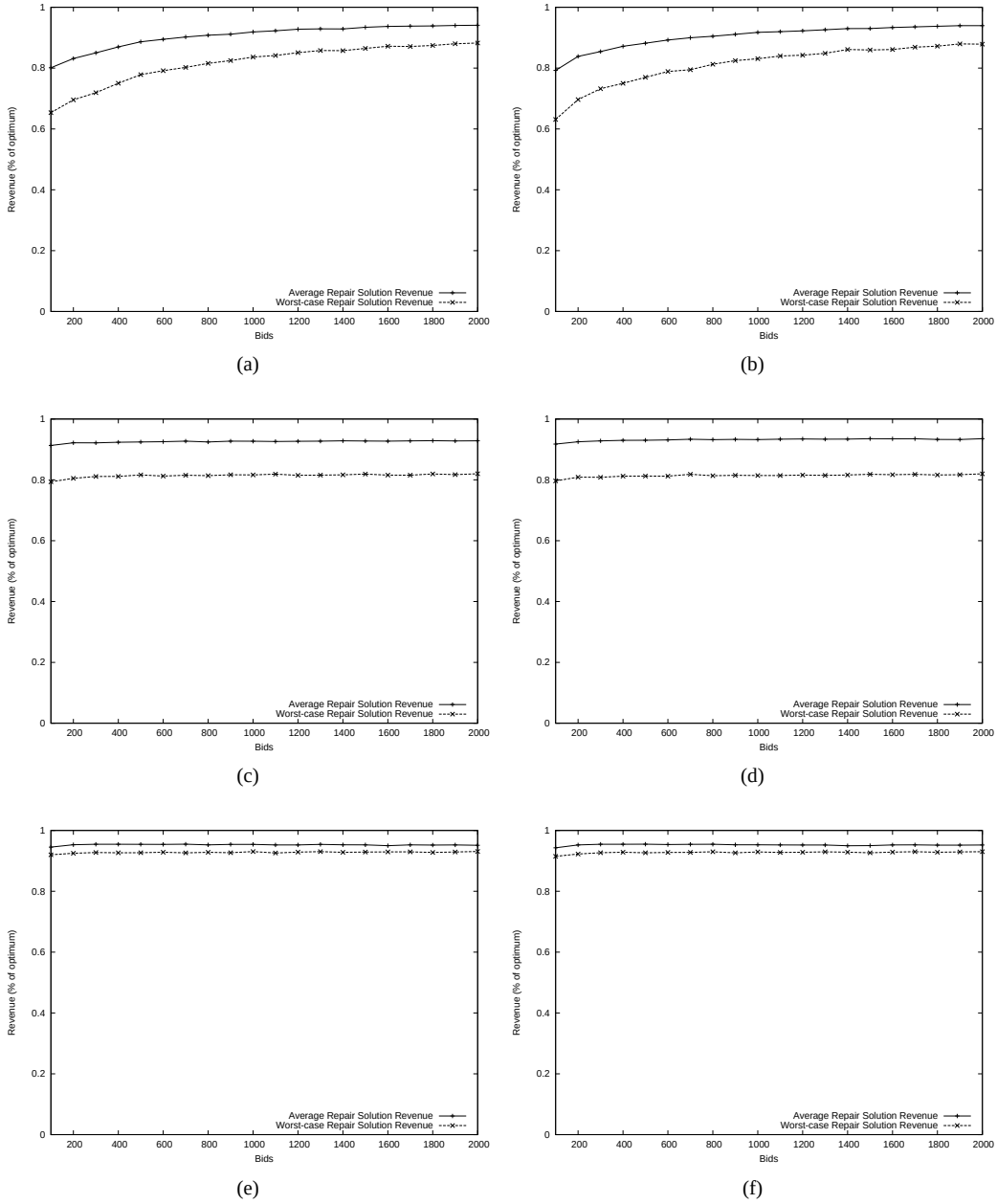

Figure 3.2: Reallocation results. (a) Arbitrary, (b) Arbitrary without dominated bids, (c) Matching, (d) Matching without dominated bids, (e) Paths, (f) Paths without dominated bids.

in the distributions that are highly affected by resource unavailability such as arbitrary or regions, robustness is without any doubt a necessity.

3.3 Repair Size Analysis

When repairing a solution, in the previous analysis (as well as in the sensitivity analysis of [34]) we have not cared about the size of the repair, since the objective of this analysis was to see the potential effects of resources that become unavailable on the revenue. Therefore, the size of the repair could be as large as required with the aim of finding the best possible repair solution. In practice, however,

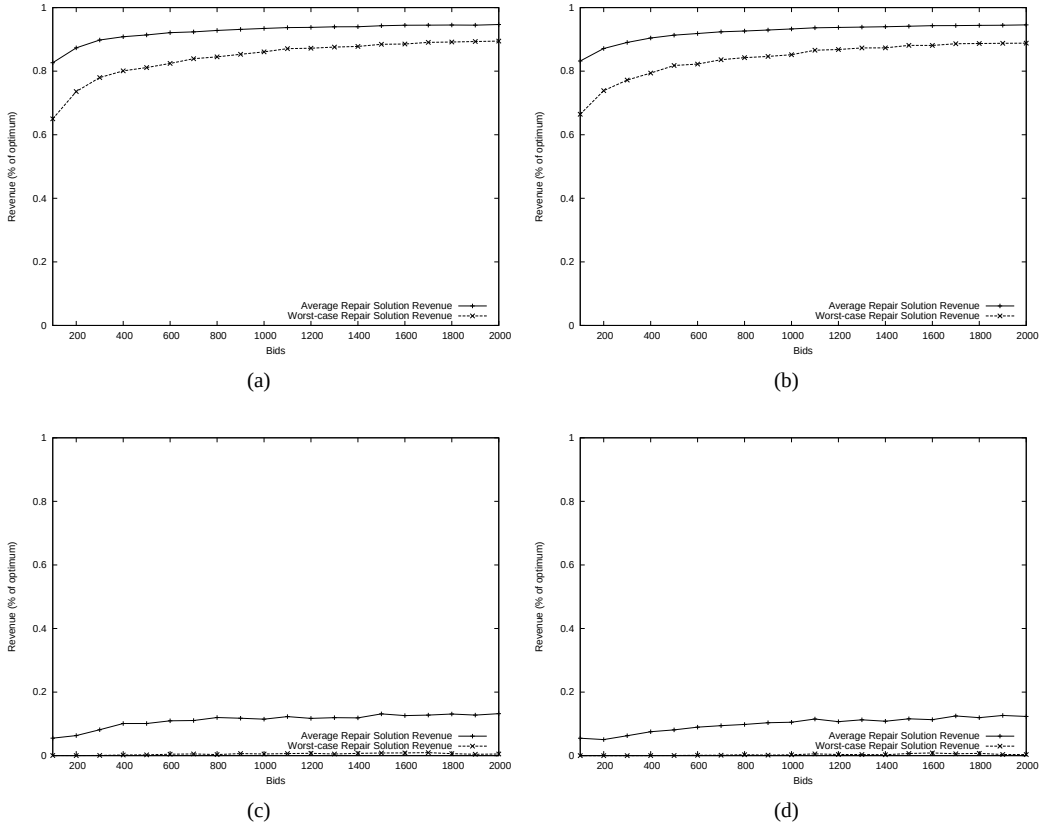

Figure 3.3: Reallocation results. (a) Regions, (b) Regions without dominated bids, (c) L7, (d) L7 without dominated bids.

the size of the repair may not be unlimited, and repairs of small size would be preferred, since the participants do not need to be bothered much whenever a break is produced, which could give a bad impression to the participants if the solution was changed completely at every break, moreover given that it is possible that some of them had began some actions based on the previous solution.

In this section we analyze the repairs' size in order to see whether the previous (optimistic) results were realistic enough or otherwise required too many changes in the solution that they were not actually practicable in real world situations. Therefore we study the number of changes required for the repair solution. If this value was low, it would mean that the size of the repair is not a crucial factor to consider. However, we will see that this is not the case indeed.

For this analysis we will not experiment with all the distributions, since the results would be quite similar, instead we will choose only one of them. The chosen one is the regions distribution, since in the results of the previous section we saw that it is not the most affected distribution when resources become unavailable (the most affected is L7) nor the least (the least affected are matching and paths) and therefore the results should be representative enough.

In Figure 3.6 we see the results of the regions distribution and the reallocation scheme. The graph shows the average and worst case (largest) size of the repair and also the size of the solution, for both dominated and non-dominated instances. We observe that although the average repair size is always between 1 and 2, the worst case can be much larger, up to 9 in non-dominated instances, and

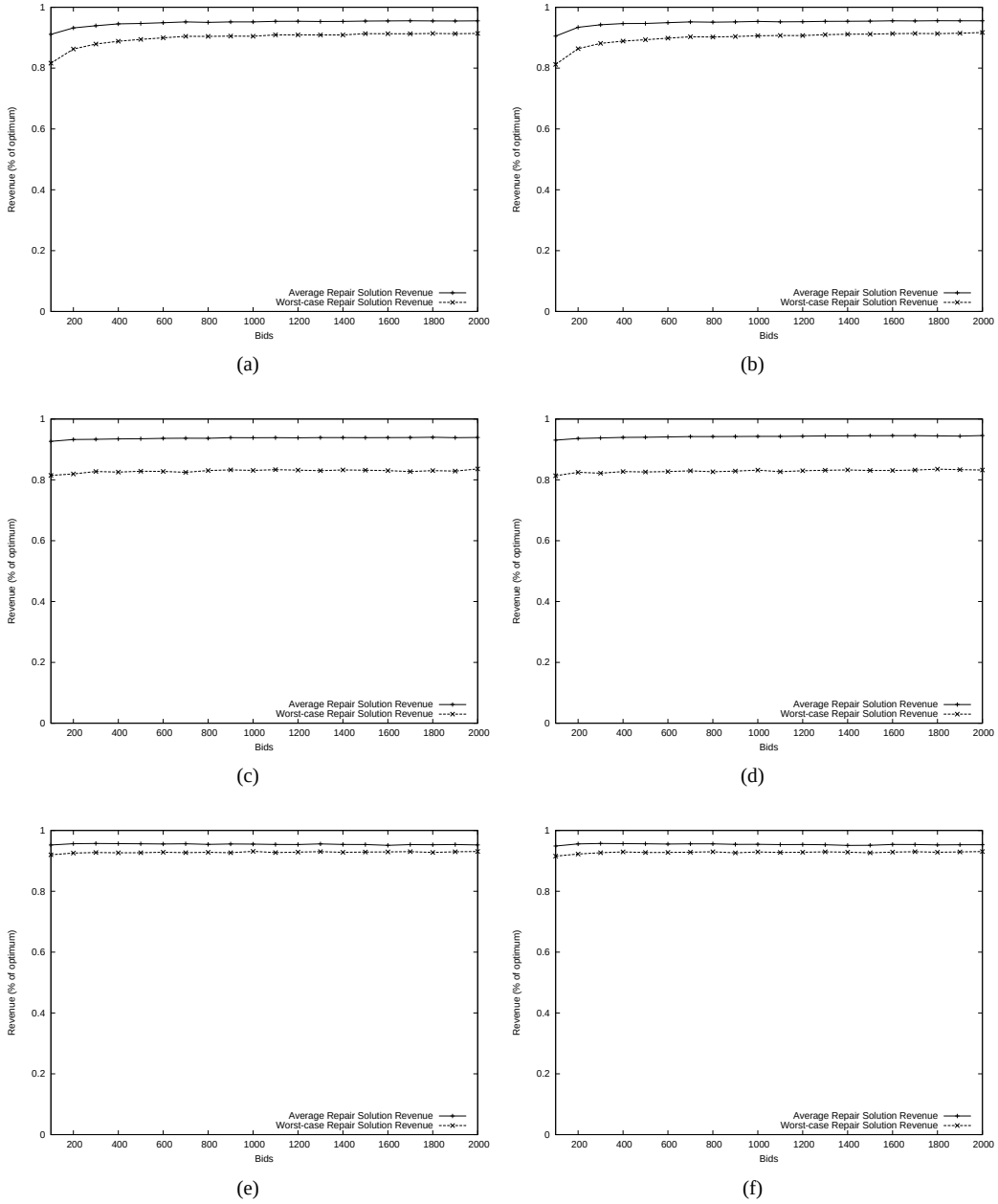

Figure 3.4: Full-reparability results. (a) Arbitrary, (b) Arbitrary without dominated bids, (c) Matching, (d) Matching without dominated bids, (e) Paths, (f) Paths without dominated bids.

up to 12 for dominated ones. In this case, the size of the problems does not seem to affect too much. However, we see that the worst case is always around 7. Although the sizes of the repairs are small, they represent about 10% of the solution size in average, which could be enough in some domains for requiring robustness, and in the worst cases it goes up to 50% (exceeding 100% in one case).

These results point out the necessity of establishing an upper bound for the repair size, since we do not see any tendency in the graphs and therefore it does not seem that the repair size can be actually predicted or controlled at all. Thus, imposing a limit in the repair size would avoid such large size of the repairs that otherwise could be needed in some instances.

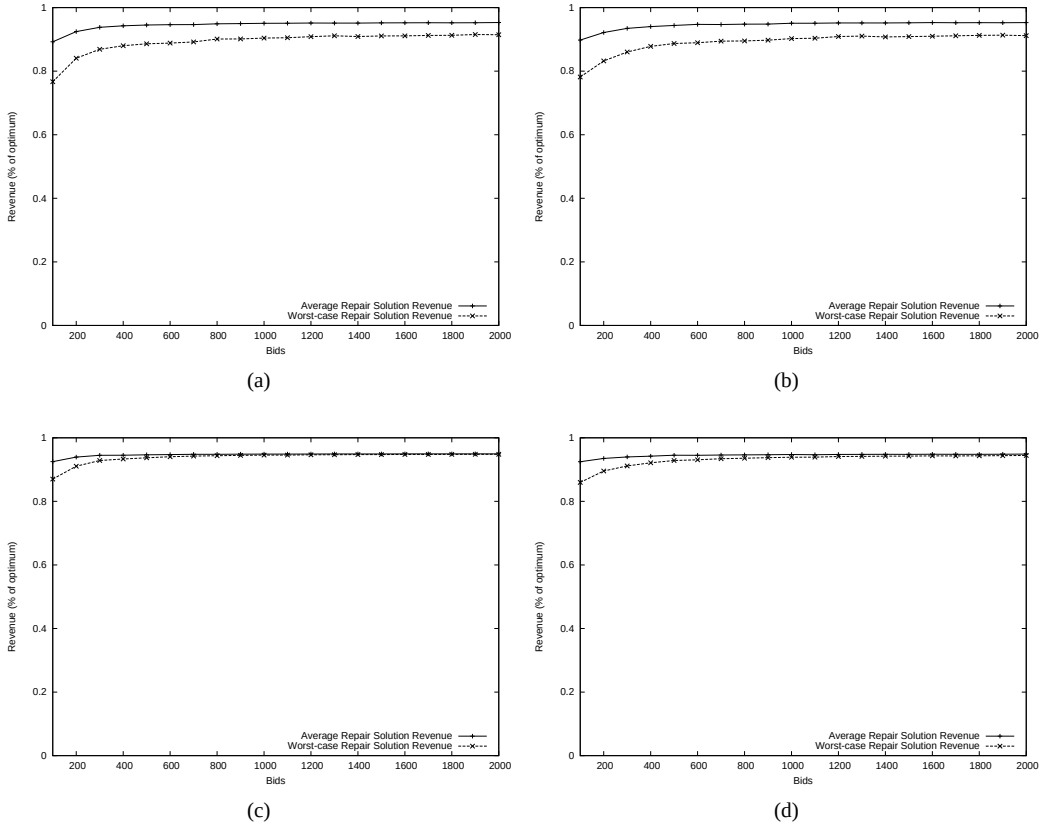

Figure 3.5: Full-Reparability results. (a) Regions, (b) Regions without dominated bids, (c) L7, (d) L7 without dominated bids.

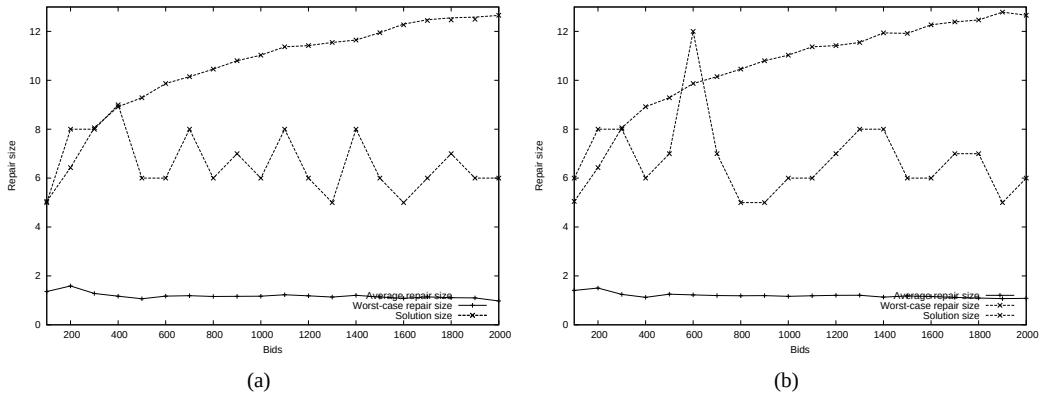

Figure 3.6: Repair size analysis. (a) Without dominated bids, (b) With dominated bids.

3.4 SUMMARY

In this chapter we have performed an extensive sensitivity analysis that has proved the importance of incorporating robustness mechanisms for combinatorial auctions, since the optimal solution in many distributions is highly affected when some of the resources become unavailable, both in terms of revenue for the auctioneer and number of changes required to repair the solution.

Concretely, we have seen that the most affected distributions are arbitrary, regions and L7. Additionally, we have observed that when the number of bids is very high, robustness is not required so much, since a repair can be easily found anyway. Therefore, in the following chapters we will restrict our experiments with the distribution L7 and with a restricted number of bids.

For the experiments in this chapter we have only considered what happens when one of the resources turns unavailable. If the number of resources that become unavailable was higher it is expected that the results should be even worse in terms of a greater loss in revenue for the auctioneer and, therefore, robustness mechanisms would be even more necessary.

The following two chapters show how to find robust solutions for combinatorial auctions using an encoding based on an extension of propositional logic. The next chapter shows a strict modeling that finds robust solutions with hard constraints, while the following chapter presents a more flexible modeling, based on soft constraints, which could be more reliable in some applications.

CHAPTER 4
Robustness of Resource Availability

In this chapter we formalize the concept of robustness for resource allocation problems formulated as combinatorial auctions. We begin by representing the resource allocation problem as a combinatorial auction, and encoding that auction as a weighted Max-SAT formula. Then, we extend that encoding in order to incorporate robustness. The extension is similar as for supermodels, but we also consider resource availability and add a parameter that guarantees a minimum revenue of the solution, including all the possible repairs. The new encoding is proved mathematically and the resulting model is tested varying the different parameters, and with several solvers. In the next chapter we will add a modification to the encoding in order to add flexibility to it and facilitate the task of defining the desired balance point between optimality and robustness.

4.1 SCHEMATIC VIEW

Our approach of robustness for resource allocation problems is based in four steps:

1. Representing the resource allocation problem as a combinatorial auction
2. Modeling the combinatorial auction as a propositional formula (Max-SAT)
3. Extending the Max-SAT formula in order to incorporate robustness, so that the solution of the formula implies a robust solution to the auction, and to the initial resource allocation problem
4. Solving the extended formula in order to find a solution that is a robust solution of the resource allocation problem

Figure 4.1 shows an schema of our approach. We begin with a resource allocation problem P which is converted to an auction A. This auction is then transformed into a partial weighted Max-SAT formula F^A that encodes exactly the same auction (and the original resource allocation problem). Then, in order to add robustness we add some clauses to the formula, hence we get a new larger formula F^A_{SM}, which is not Max-SAT but Max-SMT (since it has some inequalities). Then we use a Max-SMT solver[1] in order to find a model of the formula F^A_{SM}. This model is actually a supermodel of the formula F^A, which turns to be a super solution to the auction A, and finally, a robust solution for the original resource allocation problem P.

The initial resource allocation problem is written as a combinatorial auction by putting the resources requirements of the different agents as the bids of the auction, and their valuation as their prices. In

[1] Apart from SMT, the model can be written as a pseudo-Boolean or Linear Programming problem as well.

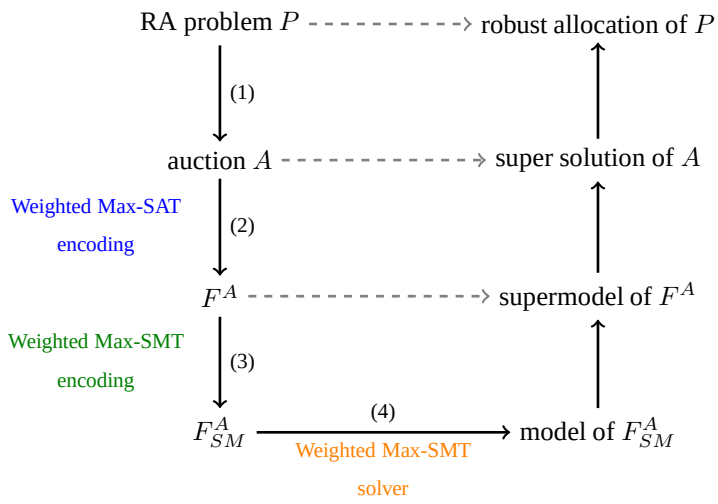

Figure 4.1: Schematic view.

the following sections, we will explain how to encode an auction into a partial weighted Max-SAT formula. Then we will define concretely what is robustness for an auction and see how to incorporate robustness through a reformulation of that formula.

4.2 Auctions as partial weighted Max-SAT problems

An auction can be easily encoded as a partial weighted Max-SAT formula [38, 32]. A partial weighted Max-SAT formula has the form $F = C \wedge W$, where C denotes the set of mandatory constraints, and W denotes the set of weighted, non-mandatory constraints. In the standard modeling (non-robust), a set of Boolean variables encode each bid $b_1, ..., b_n$ to indicate whether it is winner or loser. The mandatory clauses of the formula encode the restrictions regarding the items of the auction (the resources of the resource allocation problem) that cannot be assigned to more than one bid (to more than one agent or task in the respective resource allocation problem). Hence, the mandatory clauses (C) are the following:

- *Bid incompatibility*: For each pair of incompatible bids i,j ($i \neq j$), i.e. such that $G_i \cap G_j \neq \emptyset$, where G_i is the set of resources requested by bid i, we state:

$$\neg b_i \vee \neg b_j \tag{4.1}$$

  since it is not possible to assign both bids as winners, given that they share some items.

The non-mandatory clauses (W) are the prices (weights) of the bids:

- *Bids' values*: For each bid we add a weighted unit clause (b_i, p_i) indicating that if the i-th bid is not part of the solution, then there is a loss of revenue of p_i.

The sum of values of the satisfied weighted clauses will be maximized when solving the weighted Max-SAT problem, which means that the revenue of the auction (i.e. the sum of winning bids) will be also maximized.

This encoding allows finding an optimal solution, but does not consider robustness. Since we deal with robustness against resource unavailability, we first extend this encoding by adding another set of Boolean variables to represent resource availability $g_1, ..., g_m$. We also need to add the following mandatory clauses, which will be included in C:

- *Resource availability*: In order to indicate resource availability we state:

$$b_j \Rightarrow g_{i1} \wedge ... \wedge g_{in_j} \tag{4.2}$$

  to indicate that whenever bid j is accepted, then all of its goods must be available ($i_1, ..., i_{n_j}$ are the indices of the goods of the bid j, i.e. $G_j = \{g_{i1}, ..., g_{in_j}\}$).

Finally, we allow to introduce additional constraints in order to tailor the auction to follow specific rules. For instance, we could add the constraint that each agent wins at least one of its bids (we will use this restriction in the example of Section 4.4):

- *Minimum winning bids*
  For each set $R_k = \{i_1, \ldots, i_{n_k}\}$ corresponding to the bids of agent k, we state the following mandatory clause:

$$b_{i_1} \vee \cdots \vee b_{i_{n_k}}$$

Any additional constraint (as long as it is formulated as a Boolean formula) could be added. For example, we could easily introduce other constraints on the number of winning bids (e.g. set a minimum, maximum or exact number of winning bids per agent), impose the non-free disposal condition (i.e. every good must be allocated to some agent), or even introduce constraints over the goods being allocated (e.g. if good X is allocated to some agent, then good Y cannot be allocated to any agent), among others. The latter would allow us to express incompatibility between goods: for instance, in the conference room assignment scenario, we could have a large room (L) that can be also reconfigured into two smaller rooms (S_1, S_2), but obviously we could not allocate both the large and some small room at the same time. We could then state the clause $L \rightarrow \neg S_1 \wedge \neg S_2$.

Moreover, if we consider directly encoding the problem as a SMT formula (e.g. SAT modulo quantifier-free linear integer arithmetic) instead of plain SAT, as we will see in the following subsection, then the encoding of such constraints would be straightforward thanks to the higher expressiveness of the language.

The conjunction of the previous sets of clauses is a weighted Max-SAT problem for the auction. In the following section we show how a weighted Max-SAT formula F^A defining an auction A can be transformed into a weighted Max-SMT formula defining a robust version of the former. In particular, we describe how to obtain a weighted Max-SMT formula F^A_{SM} such that F^A_{SM} has a model if and only if A has an (a, b, β)-super solution (see definition below).

4.3 Robust auctions as partial weighted Max-SMT problems

Robustness with reparable solutions is defined based on three parameters: a, b and β, where a is the number of breakages that the solution can absorb, b is the maximum size of a repair (number

of variables changed), and β is the minimum benefit of the solution (including its possible repairs). Thus, the definition of an (a,b,β)-super solution is the following:

Definition 1. *An* (a, b, β)-super solution *of an auction is a (maximal revenue) solution for the auction such that, if* ***a*** *goods become unavailable (breakage), then another solution can be obtained by changing at most* ***b*** *bids from winner to loser or vice-versa (repair) and, moreover, the solution and all possible repaired solutions have a revenue of at least* β.

Note that asking for a (1,0,β)-super solution would make no sense, since this means handling any break (a good becoming unavailable) by not changing any bid assignment. However, we must bear in mind that if a good becomes unavailable, an immediate change must be made to the variable of the winning bid requesting that good (resource availability constraint), which should become loser. Given that a super solution must handle any possible break (i.e. in the case of $a = 1$, any single resource break), the only (1,0,β)-super solution would be to assign all bids as losers, so a break in a resource would not affect any bid. But obviously this solution would be totally useless. A more reasonable demand would be that of (1,1,β)-super solution, where one break should be repairable with one change in bid assignment, and so this single change would be on the bid requesting the affected resource, leaving the rest of bids unaltered.

4.3.1 Robust auctions as robust partial weighted Max-SAT problems

Now that we have defined the concept of robustness in auctions, we need to map it to the SAT framework. The following definition generalizes the one of [25] to weighted Max-SAT.

Definition 2. *An* (S_1^a, S_2^b, β)*-supermodel of a partial weighted Max-SAT formula* F *is a (minimal cost) model of* F *such that if we modify the values taken by the variables in a subset of* S_1 *of size at most* ***a*** *(breakage), another model can be obtained by modifying the values of the variables in a disjoint subset of* S_2 *of size at most* ***b*** *(repair) and, moreover, the solution and all possible repaired solutions have a cost of at most* β.

Note that while in auctions the objective is to maximize the revenue of the auctioneer, in weighted Max-SAT the objective is to minimize the cost of the unsatisfied clauses. Therefore the previous definition updates the one in the previous section taking that into account.

Next lemma shows that finding a super solution for an auction A is equivalent to finding a supermodel for the formula F^A that encodes the auction. Nevertheless there is a subtle asymmetry concerning initial availability of goods. Namely, notice that in auctions, all goods are assumed to be available at the starting point, while in F^A, values for variables denoting availability of goods are not established. Fortunately, next fact shows that for any optimal supermodel of F^A, there exists an equivalent supermodel (i.e., a model with exactly the same winning bids) with all good availability variables set to $true$.

Fact 1. *For any auction* A *having* N *goods and* M *bids, let* F^A *be the formula encoding it. If* I *is an* (S_1^a, S_2^b, β)*-supermodel of* F^A*, where* $S_1 = \{g_1, ..., g_N\}$ *and* $S_2 = \{b_1, ..., b_M\}$*, then there exists a supermodel* I' *where all availability variables are set to true and,* $I'(b_i) = I(b_i)$ *for all* $1 \leq i \leq M$.

Proof. Let I be an (S_1^a, S_2^b, β)-supermodel of F^A such that for some i; $I(g_i) = false$. Let I' be an interpretation for F^A such that $I'(b_i) = I(b_i)$ for all $1 \leq i \leq M$ and $I'(g_i) = true$ for all $1 \leq i \leq N$. The cost of I' is the same as the one of I because it falsifies the same set of weighted clauses that I since $I'(b_i) = I(b_i)$ for all $1 \leq i \leq M$, and *bid's value* clauses are the only weighted ones, therefore I' is optimal too. Concerning repairability, it is not hard to see that for any breakage s' (with $s' \subseteq S_1$ and $|s'| \leq a$) in I', there exists some breakage s (with $s \subseteq S_1$ and $|s| \leq |s'|$) in interpretation I such that for every g_i that is $false$ in I' under breakage s', g_i is also $false$ in I under breakage s. Therefore, all bids allowed (according to *resource availability* formula) in I under breakage s are also allowed in I' under breakage s'. Finally, since I is an (S_1^a, S_2^b, β)-supermodel and $|s| \leq a$, s must be repairable and hence, having I and I' the same values for all b_i, s' must also be repairable for I'. □

Lemma 2. *An auction A with N goods and M bids has an (a, b, β)-super solution if and only if the partial weighted Max-SAT formula F^A has an (S_1^a, S_2^b, β')-supermodel where $S_1 = \{g_1, \ldots, g_N\}$, $S_2 = \{b_1, \ldots, b_M\}$, and cost (i.e. loss of revenue) $\beta' = \left(\sum_{i=1}^{M} p_i\right) - \beta$, and p_i is the value of the i-th bid.*

Proof. ($\Rightarrow$) According to the existing robust solution for A and without loss of generality. according to the previous fact, we define an interpretation I^A for F^A that sets to $true$ all good availability variables and all winning bid variables, and sets to $false$ all loser bid variables. Now we prove that I^A is effectively an (S_1^a, S_2^b, β')-supermodel of F^A. From the construction of F^A and I^A it is easy to see that $I^A \models F^A$ because F^A encodes the auction semantics of bid incompatibility, of resource availability and of any additional restriction the auction may have. Moreover, the cost of F^A under I^A is the sum of the prices of the loser bids, in other words, the total amount of the prices minus the winning ones, i.e. $\left(\sum_{i=1}^{M} p_i\right) - \left(\sum_{i=1}^{M} p_i | I^A(b_i) = true\right)$. Since our auction solution is an (a, b, β)-super solution, we have that $\left(\sum_{i=1}^{M} p_i | I^A(b_i) = true\right) \geq \beta$, and hence the cost of F^A under I^A is at most $\left(\sum_{i=1}^{M} p_i\right) - \beta$ as required.

Next, we need to prove for I^A that any breakage of size at most a on S_1 can be repaired with at most b changes in S_2 variable values with a cost of at most β'. Since our auction solution is an (a, b, β)-super solution, when any (at most) a goods become unavailable, there exists a repair of size at most b on the winning bids, that has a revenue of at least β. Notice that having a breakages means changing the value of a variables encoding good availability from $true$ to $false$. Therefore, for any a changes in S_1 variable values, we can build an interpretation $I^A_{a,b}$ consisting on the same I^A but where the a variables of the breakage have been set to $false$, and the b variables corresponding to the winning bids of the repair of the auction have been flipped. A similar reasoning as for I^A serves to prove that $I^A_{a,b} \models F^A$ and that F^A has a cost of at most $\left(\sum_{i=1}^{M} p_i\right) - \beta$ under interpretation $I^A_{a,b}$.

Finally, since our auction super solution is optimal, so it is I^A. Suppose the opposite. Then there would exist an (S_1^a, S_2^b, β')-supermodel I' of F^A with cost smaller than the one of I^A. From I' we could build an (a, b, β)-super solution for A, following the reasoning of the ($\Leftarrow$) part of the proof, and this would have a greater revenue than the original super solution, contradicting its optimality.

($\Leftarrow$) Let I^A be an (S_1^a, S_2^b, β')-supermodel of F^A. We can build a bid assignment S_{I^A} for the auction A, setting as winning bids all the bids i such that $I^A(b_i) = true$. Since $I^A \models F^A$ and F^A satisfies all the semantics of the auction A, S_{I^A} is effectively a solution of A. Moreover, F^A has a cost equal to $\left(\sum_{i=1}^{M} p_i\right) - \left(\sum_{i=1}^{M} p_i | I^A(b_i) = true\right)$ under I^A, and we know that this cost is smaller

than or equal to β', which amounts to $\left(\sum_{i=1}^{M} p_i\right) - \beta$. Therefore, the revenue of S_{I^A}, namely $\left(\sum_{i=1}^{M} p_i | I^A(b_i) = true\right)$, is at least β.

Now we need to prove that for any (at most) a goods becoming unavailable, we can change (at most) b bid values of S_{I^A} from losers to winners or vice-versa, still obtaining a solution for A with a minimum revenue of β. Since I^A is an (S_1^a, S_2^b, β')-supermodel of F^A we know that for any (at most) a value changes (w.r.t. I^A) of variables in S_1, we can build another model for F^A by changing (at most) b values of variables in S_2. Let's call this interpretation $I^A_{a,b}$. Then $S_{I^A_{a,b}}$ is a bid assignment for auction A when at most a goods become unavailable, involving at most b changes in bid assignment w.r.t. S_{I^A}. With a similar reasoning as before, $S_{I^A_{a,b}}$ is a solution of auction A with a minimum revenue of β.

Finally, since our F^A supermodel I^A is optimal, so it is S_{I^A}. Suppose the opposite. Then there exists an (a, b, β)-super solution S' of A with profit greater than the one of S_{I^A}. From S' we could build another (S_1^a, S_2^b, β')-supermodel for F^A following the reasoning of the ($\Rightarrow$) part of the proof, and this would have a smaller cost than I^A, contradicting its optimality. □

Now we show how to construct a partial weighted Max-SMT formula F_{SM} from a partial weighted Max-SAT formula F, such that F has an (S_1^a, S_2^b, β)-supermodel if and only if F_{SM} has a model. Weighted Max-SMT formulas are a generalization of the weighted Max-SAT formulas where propositional variables can be replaced by predicates of the underlying theory (linear integer arithmetic for our purposes). Given a weighted Max-SMT formula, the weighted Max-SMT problem is the problem of finding an assignment with minimal cost.

4.3.2 Robust partial weighted Max-SAT as partial weighted Max-SMT

In this section we show how robustness can be added to any partial weighted Max-SAT formula. We proceed by means of a transformation, generalizing the result of [25]. Interestingly, after our transformation we get a partial weighted Max-SMT formula.

Let

$$F = C \wedge W$$

be a weighted Max-SAT formula, where C denotes the set of mandatory constraints and W denotes the set of weighted, non-mandatory constraints [2]. For the sake of simplicity, we will assume that W consists only of unary clauses of the form (b, w), where b is a Boolean variable and w is a weight. Note that any Max-SAT formula can be transformed into an equisatisfiable one fulfilling this requirement by reification, i.e., by replacing any weighted constraint (G, w) such that G is not unary by $(G \leftrightarrow b) \wedge (b, w)$.

Now, assuming that

$$W = (b_1, w_1) \wedge \cdots \wedge (b_k, w_k)$$

[2] In the following, we will sometimes talk about constraints instead of clauses since our transformation into the SMT setting does not require the formula to be in clausal form.

we introduce a set of integer variables $i_1, \ldots, i_k$ and define

$$L = \bigwedge_{j \in 1..k} (b_j \rightarrow i_j = 1) \wedge (\neg b_j \rightarrow i_j = 0)$$

Let S^n denote the set of all (possibly empty) subsets of a set S whose size is at most n, and let S^{n+} denote the set of all non-empty subsets of a set S whose size is at most n. Moreover, let $F_{\overline{S}}$ denote a Boolean formula F where all occurrences of variables in the set S have been flipped (i.e., negated).

We define

$$B_{\overline{S}} = \sum_{j \in 1..k} \begin{cases} i_j \cdot w_j & \text{if } b_j \in S \\ (1 - i_j) \cdot w_j & \text{if } b_j \notin S \end{cases}$$

where S is the set of Boolean variables occurring in W. We denote by B the particular case $B_{\overline{\emptyset}} = \sum_{j \in 1..k} (1 - i_j) \cdot w_j$, corresponding to the cost of the unsatisfied clauses in W. If the Boolean values $false$ and $true$ were interpreted as the integer values 0 and 1, respectively, in arithmetic expressions, we could forget about L and directly write $B = \sum_{j \in 1..k} (\neg b_j) \cdot w_j$ and analogously for $B_{\overline{S}}$.

Finally, we define

$$\begin{aligned} F_{SM} \quad = \quad & C \wedge W \wedge L \wedge (B \leq \beta) \wedge \\ & \bigwedge_{S \in S_1^{a+}} \left(\bigvee_{T \in (S_2 \setminus S)^b} (C_{\overline{S \cup T}} \wedge (B_{\overline{S \cup T}} \leq \beta)) \right) \end{aligned} \tag{4.3}$$

Roughly speaking, the meaning of the instance F_{SM} is the following: we have C, that is the mandatory clauses of the original formula, and we add $B \leq \beta$ to bound the cost. Next, we need to be able to repair all possible breakages. The big *and* accounts for all possible (*breakage*) sets S of size smaller than or equal to a, and the big *or* ensures the existence of a repair for each S. This is done by means of considering all possible (*repair*) sets T of variables from S_2 (excluding the ones in S), and limiting the number of variables from S_2 to b. By flipping the breakages and the repairs, each subformula $(C_{\overline{S \cup T}} \wedge (B_{\overline{S \cup T}} \leq \beta))$ enforces (see Lemma 3) the existence of a possible repair T for the breakage S, with a cost which is bounded by β.

Observe that, since a and b are constants, the size of F_{SM} is polynomially bounded by the size of F. Namely, F_{SM} is $\mathcal{O}(n^{a+b})$ larger than F, where n is the number of variables of F. In Section 4.3.3 we show how, by encoding the disjunctions of F_{SM} into cardinality constraints, an equisatisfiable formula which is only $\mathcal{O}(n^a)$ larger than F can be obtained.

Note that, due to L and the constraints of the form $B \leq \beta$, this formula is not plain SAT; it falls into SAT modulo the quantifier-free fragment of the (first-order) linear arithmetic theory[3].

Now we proceed to prove the correctness of the F_{SM} reformulation.

Definition 3. (Flipped interpretation). *Let S be a set of variables and I be an interpretation. Then $I_{\overline{S}}$ denotes the interpretation such that $I_{\overline{S}}(x) = \neg I(x)$ if $x \in S$ and $I_{\overline{S}}(x) = I(x)$ for all other variables x.*

[3]It will be integer or real arithmetic depending on the type of the weights w_j and β.

Lemma 3. *Let F be a Boolean formula and S be a set of propositional variables occurring in F. Then I is a model of F if and only if $I_{\overline{S}}$ is a model of $F_{\overline{S}}$.*

Proof. We proceed by induction on the size of S. If S is empty, the lemma holds trivially. To conclude we need to show that if $I \models F \Leftrightarrow I_{\overline{S}} \models F_{\overline{S}}$ then $I \models F \Leftrightarrow I_{\overline{S\cup\{p\}}} \models F_{\overline{S\cup\{p\}}}$ for every set of propositional variables S and every propositional variable p not included in S. For this observe that, for every literal l with the variable p (that is p or $\neg p$), we have $I_{\overline{S}}(l) = I_{\overline{S\cup\{p\}}}(\neg l)$. Therefore, $I_{\overline{S}} \models F_{\overline{S}} \Leftrightarrow I_{\overline{S\cup\{p\}}} \models F_{\overline{S\cup\{p\}}}$ and hence if $I \models F \Leftrightarrow I_{\overline{S}} \models F_{\overline{S}}$ then $I \models F \Leftrightarrow I_{\overline{S\cup\{p\}}} \models F_{\overline{S\cup\{p\}}}$. □

The next theorem follows the spirit of the result of [25], but adding costs. Here, the key idea is that $B_{\overline{S\cup T}}$, gives us the cost of the unsatisfied clauses in W if the variables in $S\cup T$ were to change their value with respect to the initial solution. Note that the variables in S represent the breakage variables and the ones in T represent the repair variables and, hence, $S \cup T$ denotes the set of variables that are going to change their value. Note also that the sets S and T are disjoint, since it makes no sense to repair a broken variable.

Theorem 4. *Let F be a partial weighted Max-SAT formula, let S_1 and S_2 be sets of variables occurring in F, and let a, b and β be non-negative integer constants. Then F has an (S_1^a, S_2^b, β)-supermodel if and only if the instance F_{SM} has a solution.*

Proof. Let us recall that a (S_1^a, S_2^b, β)-supermodel of F is a minimal cost model of F such that if we modify the values taken by the variables in a subset of S_1 of size at most a, then another model can be obtained by modifying the values of the variables in a disjoint subset of S_2 of size at most b and, moreover, the solution and all possible repaired solutions have a cost of at most β.

By assumption, $F = C \wedge W$ where C is a set of mandatory clauses and W is a set of weighted (non-mandatory) unit clauses. For the right-to-left implication, let I be a solution (i.e., an optimal truth assignment) to F_{SM}. From the definition of B, we clearly have that the cost of the unsatisfied clauses in W is at most β. Notice also that I is a model of F, since F is included in the PB-constraints of F_{SM}. To conclude we will show that I is a model of F such that if we modify the values taken by the variables in a subset S of S_1 with $|S| \leq a$, then another model can be obtained by modifying the values of the variables in a disjoint subset T of S_2 with $|T \cap S_2| \leq b$ and, moreover, the repaired solution has a cost of at most β. If $S = \emptyset$ then this trivially holds taking $T = \emptyset$. Otherwise $a > 0$ and $1 \leq |S| \leq a$ and, since I is a solution of F_{SM}, then $I \models \vee_{T\subseteq(S_2\setminus S),|T\cap S_2|\leq b}(C_{\overline{S\cup T}} \wedge (B_{\overline{S\cup T}} \leq \beta))$, that is, $I \models C_{\overline{S\cup T}} \wedge (B_{\overline{S\cup T}} \leq \beta)$ for some T in $(S_2 \setminus S)$ with $|T \cap S_2| \leq b$. Now, since $I \models C_{\overline{S\cup T}}$, by Lemma 3 we have $I_{\overline{S\cup T}} \models (C_{\overline{S\cup T}})_{\overline{S\cup T}}$ and, since $(C_{\overline{S\cup T}})_{\overline{S\cup T}} = C$, we have $I_{\overline{S\cup T}} \models C$ as desired. Finally, we show that $I \models B_{\overline{S\cup T}} \leq \beta$ implies that the cost of the unsatisfied clauses in W under interpretation $I_{\overline{S\cup T}}$ is at most β. Notice that $I(B)$ is the cost of the unsatisfied clauses in W. Now, considering the interpretation $I_{\overline{S\cup T}}$ we have that $I_{\overline{S\cup T}}(B_{\overline{S\cup T}}) = I(B)$. Therefore, $I_{\overline{S\cup T}}((B_{\overline{S\cup T}})_{\overline{S\cup T}}) = I(B_{\overline{S\cup T}})$, that is, $I_{\overline{S\cup T}}(B) = I(B_{\overline{S\cup T}})$. From this and the fact that $I \models B_{\overline{S\cup T}} \leq \beta$, we finally get that $I_{\overline{S\cup T}}(B)$ (which amounts to the cost of the unsatisfied clauses in W under interpretation $I_{\overline{S\cup T}}$) is at most β.

For the left-to-right implication, let I be a model of F such that if we modify the values taken by the variables in a subset S of S_1 of size at most a, then another model can be obtained by modifying the values of the variables in a disjoint subset T of S_2 such that $|T \cap S_2| \leq b$ (that

is, $I_{\overline{S \cup T}} \models F$) and, moreover, both the solution and the repaired solution have a cost of at most β. We will show that every such an interpretation I is a solution of F_{SM}. Since the solution has cost at most β, we have that $I \models B \leq \beta$. Now it remains to be proved that it holds $I \models \wedge_{S \subseteq S_1, 1 \leq |S| \leq a}(\vee_{T \subseteq (S_2 \setminus S), |T \cap S_2| \leq b}(C_{\overline{S \cup T}} \wedge (B_{\overline{S \cup T}} \leq \beta)))$. For this recall that, by assumption, for every subset S of S_1 with $|S| \leq a$ there exists a subset T of $(S_2 \setminus S)$ with $|T \cap S_2| \leq b$ such that $I_{\overline{S \cup T}} \models C$. Then, by Lemma 3, $(I_{\overline{S \cup T}})_{\overline{S \cup T}} \models C_{\overline{S \cup T}}$ and, since $(I_{\overline{S \cup T}})_{\overline{S \cup T}} = I$, we have $I \models C_{\overline{S \cup T}}$. In order to show that $I \models B_{\overline{S \cup T}} \leq \beta$, recall that the cost of the repaired solution is at most β, which means that $I_{\overline{S \cup T}} \models B \leq \beta$. Then since, as seen before, $I_{\overline{S \cup T}}(B) = I(B_{\overline{S \cup T}})$, we have $I \models B_{\overline{S \cup T}} \leq \beta$. □

Observe that, in the previous proof, we have established a bijection between supermodels of F and solutions of F_{SM}. Moreover, since B denotes the cost of the unsatisfied clauses in W, it is not difficult to see that optimality is preserved. Hence, it follows that a solution of F_{SM} is precisely a (S_1^a, S_2^b, β)-supermodel of F.

Corollary 5. *Let F be a partial weighted Max-SAT formula, let S_1 and S_2 be sets of variables occurring in F, and let a, b and β be non-negative integer constants. Then every optimal solution of F_{SM} is a (S_1^a, S_2^b, β)-supermodel of F.*

Theorem 6. *An auction A has an (a, b, β)-super solution if and only if the weighted Max-SMT formula F_{SM}^A has a model.*

Proof. Let $F^A = C \wedge W$ denote the weighted Max-SAT formula obtained from A as explained in Subsection 4.2, where C and W denote respectively the set of mandatory and non-mandatory constraints introduced in section 4.2. Let $S_1 = \{g_1, \ldots, g_N\}$, $S_2 = \{b_1, \ldots, b_M\}$ and $\beta' = \left(\sum_{i=1}^{M} p_i\right) - \beta$. By Theorem 4, F^A has an (S_1^a, S_2^b, β')-supermodel if and only if F_{SM}^A has a model. Therefore, by Lemma 2, A has an (a, b, β)-super solution if and only if F_{SM}^A has a model. □

Given an (a, b, β)-super solution and a breakages, finding a repair with at most b changes in variable values has polynomial time complexity: simply generate the $\mathcal{O}(n^b)$ repairs on the broken solution, check whether they are solutions or not, and get the one with the maximum revenue, greater than β.

Corollary 7. *Given an auction A, the decision problem of the existence of an (a, b, β)-super solution for A is NP-hard.*

4.3.3 Robustness with cardinality constraints

The Formula 4.3 is the most straightforward to understand but it grows exponentially with both a and b. In this section we show how we can get rid of the disjunctions of the previous encoding by means of cardinality constraints. A formula which is only $\mathcal{O}(n^a)$ larger than F is obtained. This is especially important, since it means that the complexity of our approach does not depend on the number of repairs, but only on the number of breakages, which is usually assumed to be low. In fact, in most of the previous works on robustness the number of breakages has always been set to one.

The main idea is to define, for each possible break S, a new set of variables b_{i_S}. These are duplicate variables from the original instance that will encode the repair solution. These variables have to

satisfy the same constraints as the original ones taking care that the variables of the repair solution do not use the unavailable resources, therefore the restrictions are duplicated with some modifications (the restriction $C \wedge B \leq \beta$ is duplicated to $C_{\overline{S}} \wedge B_{\overline{S}} \leq \beta$ with the variables of the breakage S negated). Finally, the changes made on the repair solution with respect to the initial solution should not exceed the maximum size of the repair (b). In order to do so, we also define new integer variables (d_i) to indicate whether the repair variable is different to the corresponding initial variable, which are calculated as shown in Equation 4.4.

$$(d_{i_S} = 0 \Leftrightarrow b_i = b_{i_S}) \wedge (d_{i_S} = 1 \Leftrightarrow b_i \neq b_{i_S}) \tag{4.4}$$

Therefore, given that each breakage cannot have a repair size bigger than b, the sum of differences of each breakage is limited to b. At the end, the improved partial weighted Max-SMT formula encoding a robust CA with cardinality constraints is the following formula:

$$F^{\nabla}_{SM} \quad = \quad C \wedge L \wedge W \wedge (B \leq \beta) \wedge \bigwedge_{S \in S_1^{a+}} \left(C_{\overline{S}} \wedge (B_{\overline{S}} \leq \beta) \wedge \sum_i d_{i_S} \leq b \right) \tag{4.5}$$

Another advantage of this modification, apart from the reduction of the complexity, is that once the model is solved, all the repairs are also determined (by the duplicated variables), while the previous encoding only assured that a repair could be found but it was not given.

Now we proceed to prove the correctness of the F^{∇}_{SM} reformulation.

Definition 4. (Variable renaming). *Let R be a set of variables. The function $\delta_R : \mathcal{X} \to \mathcal{X}$ is defined as $\delta_R(x) = x^S$ for every variable $x \in R$, where x^S is a new atom, and $\delta_R(x) = x$ if $x \notin R$.*

Definition 5. (Difference cardinality). *Let R and S be sets of variables, and $\delta_{R,S}$ be a variable renaming function. Then we define the* difference cardinality formula *as $\nabla^{\delta_{R,S}} = \sum_{x \in R} (x \neq \delta_{R,s}(x))$*

Lemma 8. *Let F be a Boolean formula, R be a subset of the variables of F and I be an interpretation. Then $I \models F^{\delta_R}$ if and only if $I \models F_{\overline{D}}$, where $D = \{x \in Var(F) \mid I(x) \neq I(\delta_R(x))\}$.*

Proof. First of all notice that from the definition of D it follows that the domain of I is $Var(F) \cup Var(F^{\delta_R})$. Moreover, since F^{δ_R} is a variable renamed version of F and $F_{\overline{D}}$ is a variable flipped version of F, we have that both formulas are the same except for some variable renamings (for each variable in R) and negations (for each variable in D). Hence, for every literal l in $F_{\overline{D}}$ we have a corresponding literal l' in F^{δ_R}.

Now, let l be any literal occurring in $F_{\overline{D}}$, and l' its corresponding literal in F^{δ_R}. Without loss of generality, we assume that l is either of the form x or $\neg x$, where x is a variable in $Var(F)$. We distinguish between two cases. If $I(x) = I(\delta_R(x))$ then $x \notin D$ and, hence, l occurs in $F_{\overline{D}}$ with the same polarity with which l' occurs in F^{δ_R}. Then, since $I(x) = I(\delta_R(x))$, we have that $I(l) = I(l')$. If, otherwise, $I(x) \neq I(\delta_R(x))$ then $x \in D$ and, hence, l occurs in $F_{\overline{D}}$ with the opposite polarity with which l' occurs in F^{δ_R}. Then, since $I(x) \neq I(\delta_R(x))$, and literals l and l' have opposite polarity, we have that $I(l) = I(l')$.

Finally, since $I(l) = I(l')$ for all literals, we have that $I \models F^{\delta_R} \Leftrightarrow I \models F_{\overline{D}}$. □

Next lemma shows the equivalence of the cardinality constraints formulation and the previous exhaustive encoding presented in section 4.3.2.

Lemma 9. *Let F be a Boolean formula, S_1 and S_2 be sets of variables occurring in F, a and b be non-negative integer numbers, and I be an interpretation. Then*

$$I \models \bigwedge_{S \subseteq S_1, 1 \leq |S| \leq a} \left(\bigvee_{T \subseteq (S_2 \setminus S), |T \cap S_2| \leq b} F_{\overline{S \cup T}} \right)$$

if and only if there exists an interpretation I' such that $I'(x) = I(x)$ for all $x \in Var(F)$ and

$$I' \models \bigwedge_{S \subseteq S_1, 1 \leq |S| \leq a} \left(F_{\overline{S}}^{\delta_{((S_2 \setminus S)),S}} \wedge (\nabla^{\delta_{(S_2 \setminus S),S}} \leq b) \right)$$

Proof. For the left-to-right direction, we have that for every set $S \subseteq S_1$ with $1 \leq |S| \leq a$, there exists a set $T \subseteq (S_2 \setminus S)$ with $|T \cap S_2| \leq b$ such that $I \models F_{\overline{S \cup T}}$. We can define an interpretation I^S whose domain is $Var(F) \cup Var(F^{\delta_{((S_2 \setminus S)),S}})$ and such that $I^S(x) = I(x)$ for all $x \in Var(F)$ and $I^S(x^S) \neq I(x)$ if and only if $x \in T$ for all $x^S \in Var(F^{\delta_{((S_2 \setminus S)),S}})$. Now observe that $F_{\overline{S \cup T}} = (F_{\overline{S}})_{\overline{T}}$ since S and T are disjoint. Moreover, since $I \models (F_{\overline{S}})_{\overline{T}}$ and $I^S(x) = I(x)$ for all $x \in Var(F)$, then also $I^S \models (F_{\overline{S}})_{\overline{T}}$. Then, by Lemma 8, taking $F_{\overline{S}}$ for F, $((S_2 \setminus S))$ for R, I^S for I and T for D, we have $I^S \models F_{\overline{S}}^{\delta_{((S_2 \setminus S)),S}}$. Concerning $(\nabla^{\delta_{(S_2 \setminus S),S}} \leq b)$, by construction of I^S we have $I^S(\nabla^{\delta_{(S_2 \setminus S),S}}) = |T \cap (S_2 \setminus S)| \leq |T \cap S_2| \leq b$. Finally, since all considered sets S are different, the domains of every pair of such interpretations I^S can only have non-renamed variables of F in common. Then, since all interpretations I^S give the same value to non-renamed variables of F, we conclude that there exists an interpretation I' that is compatible with all I^S, that is, an interpretation with the same truth assignment as every I^S.

For the right-to-left implication, we have the following: $I' \models F_{\overline{S}}^{\delta_{((S_2 \setminus S)),S}} \wedge (\nabla^{\delta_{(S_2 \setminus S),S}} \leq b)$ for every set $S \subseteq S_1$ with $1 \leq |S| \leq a$. We define T as the set of variables x in $(S_2 \setminus S)$ such that $I'(x^S) \neq I'(x)$. We trivially have that $T \subseteq (S_2 \setminus S)$. Now we show that $|T \cap S_2| \leq b$. For this, first of all note that, we have that $T \cap (S_2 \setminus S) = T \cap S_2$. Then, by definition of T, we have that $\sum_{x \in (S_2 \setminus S)} (I'(x^S) \neq I'(x)) = |T \cap (S_2 \setminus S)| = |T \cap S_2|$. Moreover, by assumption, we have that $I' \models (\nabla^{\delta_{(S_2 \setminus S),S}} \leq b)$, i.e., $\sum_{x \in (S_2 \setminus S)} (I'(x^S) \neq I'(x)) \leq b$. Therefore, $|T \cap S_2| \leq b$. Next, since $I' \models F_{\overline{S}}^{\delta_{((S_2 \setminus S)),S}}$, we also have that $I' \models F_{\overline{S \cup T}}$ by Lemma 8, taking $F_{\overline{S}}$ for F, $((S_2 \setminus S))$ for R, and I' for I. Finally since $I'(x) = I(x)$ for all $x \in Var(F)$, we have that $I \models F_{\overline{S \cup T}}$, which lets us conclude. □

Theorem 10. *Let F be a partial weighted Max-SAT formula, let S_1 and S_2 be sets of variables occurring in F, and let a, b and β be non-negative integer constants. Then F has a (S_1^a, S_2^b, β)-supermodel if and only if the instance F_{SM}^{∇} has a solution.*

Proof. By assumption, $F = C \wedge W$ where C is a set of mandatory clauses and $W = (l_1, w_1) \wedge \cdots \wedge (l_k, w_k)$ is a set of weighted, non-mandatory unit clauses. Moreover, we define $B = \sum_{j \in 1..k} \neg l_j \cdot w_j$, for all j in $1..k$. Hence, we conclude by Theorem 4 and Lemma 9, taking $F = C \wedge (B \leq \beta)$. □

Corollary 11. *Let F be a partial weighted Max-SAT formula, let S_1 and S_2 be sets of variables occurring in F, and let a, b and β be non-negative integer constants. Then every optimal solution of F_{SM}^{∇} is a (S_1^a, S_2^b, β)-supermodel of F.*

In this section we have proved that finding supermodels for partial weighted Max-SAT formulae F amounts to solving the corresponding instances F_{SM}^{∇}. It is worth noting that solutions of F_{SM}^{∇} allow

us to immediately find an appropriate repair for every permitted breakage: given a solution I, and a breakage S, a repair (set of variables that must flip their value) for S is the set $\{x \in Var(F) \mid I(x) \neq I(x^S)\}$.

This approach for finding supermodels using difference cardinality constraint resembles those presented in [72, 30, 31, 28] for the CP setting. The reformulation approach presented in [72] for finding (1,0)-super solutions, referred to as $P + P$ in [30], and explained in Section 2.4.5 consists in duplicating the breakable variables and adding a not equals constraint between each original variable and its duplicate as shown in Figure 2.5. In [28] it is mentioned that the same duplication approach could be generalized for finding (a, b)-super solutions. However, the authors argue that the size of the new problem would be prohibitive and opt for using a backtracking algorithm [31] where the original problem is only duplicated to check that an assignment can be repaired. Thus, although the algorithm finds a super solution, it does not provide the repairs, since it "forgets" them once it has checked the repairability of a solution. The main problem of using the reformulation approach in the CP setting is the space needed to store the variables' domains and the restrictions among them, which are replicated many times. In our approach we also duplicate the problem for each possible break through the renaming function, and limit the number of changes to be lower than b with the difference cardinality constraint. Fortunately, in the Boolean settings such reformulation is not that expensive, since the domains are restricted to $\{false, true\}$ and the only restrictions are the Boolean clauses.

4.4 EXAMPLE

In order to illustrate our approach, we present an example to explain each of the steps needed to find robust solutions to auctions. The example is deliberately simple so that the notion of robustness and its codification should be clear to the reader.

For this example, we use single-item bundles, that is, each bid requests only one item (not combinatorial bids), although the reader should note that the proposed approach is also valid for combinatorial auctions. Moreover, in order to show the expressiveness of our approach, we impose the constraint that each agent must win at least one of the bids it sends (which could be changed according to the auction's rules and requirements).

Assume we have 3 agents and 4 goods (or resources). In the following table we indicate the price of each single-item bid of each agent, where each row represents a bidder and each column an item.

		Goods			
		1	2	3	4
Agents	1	10	15	-	-
	2	-	5	10	20
	3	15	-	-	10

Thus we have the following list of bids:

$$[\underbrace{(1,10),(2,15)}_{\text{first agent's bids}},\underbrace{(2,5),(3,10),(4,20)}_{\text{second agent's bids}},\underbrace{(1,15),(4,10)}_{\text{third agent's bids}}]$$

We define the Boolean variables g_1, g_2, g_3 and g_4 to represent the availability of the corresponding goods, and the Boolean variables $b_i, i \in \{1..7\}$ to indicate whether bid i is winner or loser. Then, assuming that the only possible source of breakages is resource availability, we define the breakage set as being $S_1 = \{g_1, g_2, g_3, g_4\}$. Then, the repair set is $S_2 = \{b_1, b_2, b_3, b_4, b_5, b_6, b_7\}$, since a break in a resource may imply that a winning bid becomes loser and, eventually, other assignments can be reconsidered in order to improve the auctioneer's revenue under the new circumstances.

Assume that we look for *robust solutions* where one break may occur and each possible break must be repairable with at most four changes. Assume, moreover, that we want that whatever the break is, the revenue of the initial solution and of the repaired solution is at least 30. This will correspond to a $(1, 4, 30)$-super solution.

The optimal solution to this auction without considering robustness shown in Figure 4.2, would be to set as winning bids the second, the fourth, the fifth and the sixth bids, i.e.,

$$b_1 = 0, b_2 = 1, b_3 = 0, b_4 = 1, b_5 = 1, b_6 = 1, b_7 = 0,$$

which means assigning good 2 to the first agent, goods 3 and 4 to the second agent and good 1 to the third agent. For the sake of readability we use the notation such as 2456 to indicate which are the winning bids of a solution. With this solution, the auctioneer would have a revenue of 60.

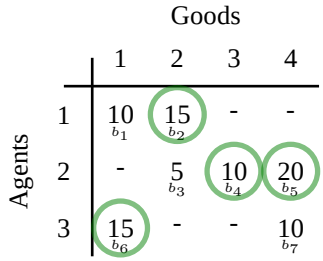

Figure 4.2: Optimal solution. Winning bids are those encircled.

However, this optimal solution is not a $(1, 4, 30)$-super solution as we can see in Figure 4.3: if good 2 became unavailable (break), the only alternative for the first agent would be good 1, but this is already allocated to the third agent; this would imply finding also an alternative for the third agent, which would be good 4, but this good is allocated to the second agent. Thus, repairing the breakage of good 2 would imply modifying two winning bids (b_2 to b_1 and b_6 to b_7) and unassigning one winning bid (b_5), meaning five repairs (as shown in boldface):

$$\mathbf{b_1 = 1}, \mathbf{b_2 = 0}, b_3 = 0, b_4 = 1, \mathbf{b_5 = 0}, \mathbf{b_6 = 0}, \mathbf{b_7 = 1},$$

which is more than the four allowed repairs. Note that for each bidder, choosing a new winning bid may imply two repairs (one to set the initially winning bid to 0, and in case he had no other winning bids, another one to set one of its losing bids to 1).

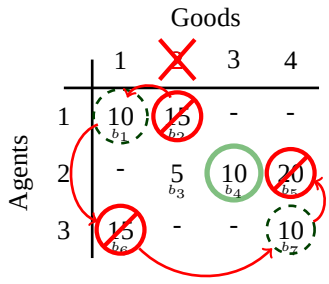

Figure 4.3: Repair solution when good 2 turns unavailable. Prohibition signs denote initially winning bids which are changed to losers. Dashed circles denote new winning bids.

This auction has 9 feasible solutions (i.e., solutions satisfying the constraints on bid incompatibility and minimum number of winning bids per agent), which are the following: 1247, 137, 1347, 147, 246, 2456, 247, 2467 and 256. Within these solutions, only three of them are $(1, 4, 30)$-super solutions: 246, 2467 and 247. Next we go through the details of solution 2467 shown in Figure 4.4, i.e.,

$$b_1 = 0, b_2 = 1, b_3 = 0, b_4 = 1, b_5 = 0, b_6 = 1, b_7 = 1$$

which has a revenue of 50 units for the auctioneer. For this solution, the four possible breakages can be repaired as follows:

1. $g_1 = 0$. The repair is $b_6 = 0$, being the new solution 247, and the revenue 35.
2. $g_2 = 0$. The repair is $b_1 = 1, b_2 = 0, b_6 = 0$, being the new solution 147 and the revenue 30.
3. $g_3 = 0$. The repair is $b_4 = 0, b_5 = 1, b_7 = 0$, being the new solution 256 and the revenue 50.
4. $g_4 = 0$. The repair is $b_7 = 0$, being the new solution 246 and the revenue 40.

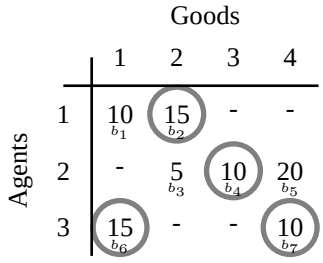

Figure 4.4: Optimal robust solution.

It can be seen that all repairs have a revenue of at least 30 and the number of repairs is not greater than 4. Moreover, in the third case there is not even loss in the revenue. We let the reader check that solutions 246 and 247 are also $(1, 4, 30)$-super solutions, with a revenue of 40 and 35, respectively. However, since the revenue of solution 2467 is higher, this would be the optimal $(1, 4, 30)$-super solution to this auction.

As for the rest of feasible solutions, some of them (147 and 1247) are $(1, 4, \_)$-super solutions, meaning that they can be repaired with at most 4 changes, but not $(1, 4, 30)$-super solutions, since

they do not satisfy that the solution and its repairs have a revenue of at least 30. In particular, solution 147 has a revenue of 30, but one of its repairs has a revenue of only 25 (when good 3 becomes unavailable, the second agent must be assigned good 2, which corresponds to a low value bid). Similarly, solution 1247 has a revenue of 45, but it also fails in the revenue of repairs, since one of them has again a revenue of 25.

Finally, some solutions (137, 1347, 2456 and 256) are not even $(1, 4, \_)$-super solutions, since they do need more than 4 changes in order to repair some of the breakages. This is the case of the optimal solution without robustness (2456), as we have seen a few paragraphs above.

Now we describe how to model this example as a robust auction. We begin by modeling the auction as a Max-SAT problem as explained in Subsection 4.2, which gives us $F^A = C \wedge W$ with

$$\begin{aligned} C = {} & (b_1 \rightarrow g_1) \wedge (b_2 \rightarrow g_2) \wedge (b_3 \rightarrow g_2) \wedge (b_4 \rightarrow g_3) \wedge \\ & (b_5 \rightarrow g_4) \wedge (b_6 \rightarrow g_1) \wedge (b_7 \rightarrow g_4) \wedge \\ & (\neg b_1 \vee \neg b_6) \wedge (\neg b_2 \vee \neg b_3) \wedge (\neg b_5 \vee \neg b_7) \wedge \\ & (b_1 \vee b_2) \wedge (b_3 \vee b_4 \vee b_5) \wedge (b_6 \vee b_7) \end{aligned}$$

and

$$\begin{aligned} W = {} & (b_1, 10) \wedge (b_2, 15) \wedge (b_3, 5) \wedge (b_4, 10) \wedge \\ & (b_5, 20) \wedge (b_6, 15) \wedge (b_7, 10) \end{aligned}$$

Note that the sum of the costs of the non-mandatory weighted clauses is $10 + 15 + 5 + 10 + 20 + 15 + 10 = 85$. Then, as stated by Lemma 2, in order to A have a $(1, 4, 30)$-super solution, we must look for a $(S_1^1, S_2^4, 85 - 30)$-supermodel of F^A, i.e., a $(S_1^1, S_2^4, 55)$-supermodel of F^A, where $S_1 = \{g_1, g_2, g_3, g_4\}$ and $S_2 = \{b_1, b_2, b_3, b_4, b_5, b_6, b_7\}$. Finally, according to Theorem 4, this amounts to find a model of the weighted Max-SMT formula

$$\begin{aligned} F_{SM}^A \quad = \quad & C \wedge W \wedge L \wedge (B \leq 55) \wedge \\ & \bigwedge_{S \in S_1^{1+}} \left(\bigvee_{T \in (S_2 \setminus S)^4} (C_{\overline{S \cup T}} \wedge (B_{\overline{S \cup T}} \leq 55)) \right) \end{aligned}$$

as described in Subsection 4.3.2, where

$$L = \bigwedge_{j \in 1..7} (b_j \rightarrow i_j = 1) \wedge (\neg b_j \rightarrow i_j = 0)$$

and

$$\begin{aligned} B = {} & (1 - i_1) \cdot 10 + (1 - i_2) \cdot 15 + (1 - i_3) \cdot 5 + (1 - i_4) \cdot 10 \\ & + (1 - i_5) \cdot 20 + (1 - i_6) \cdot 15 + (1 - i_7) \cdot 10 \end{aligned}$$

Note that S_1^{1+} denotes the non-empty subsets of S_1 with at most one element, i.e., the singletons $\{g_1\}, \{g_2\}, \{g_3\}$ and $\{g_4\}$. And, since S_1 and S_2 are disjoint, we have that $(S_2 \setminus S)^4 = S_2^4$, i.e., the (possibly empty) subsets of S_2 of size at most 4.

Due to their extension we only develop the first terms of $C_{\overline{S \cup T}}$, i.e. when the breakage set is g_1 and the repair set is empty:

$$\begin{aligned} C_{\overline{\{g_1\}}} = & (b_1 \rightarrow \neg g_1) \wedge (b_2 \rightarrow g_2) \wedge (b_3 \rightarrow g_2) \wedge (b_4 \rightarrow g_3) \wedge \\ & (b_5 \rightarrow g_4) \wedge (b_6 \rightarrow \neg g_1) \wedge (b_7 \rightarrow g_4) \wedge \\ & (\neg b_1 \vee \neg b_6) \wedge (\neg b_2 \vee \neg b_3) \wedge (\neg b_5 \vee \neg b_7) \wedge \\ & (b_1 \vee b_2) \wedge (b_3 \vee b_4 \vee b_5) \wedge (b_6 \vee b_7) \end{aligned}$$

Finally, according to Theorem 10, the formula can be written using the cardinality constraints reformulation as follows:

$$\begin{aligned} F^A_{SM} \quad = \quad & C \wedge W \wedge L \wedge (B \leq 55) \wedge \\ & \bigwedge_{S \in S_1^{1+}} \left(C_{\overline{S}}^{\delta_{(S_2 \setminus S), S}} \wedge B_{\overline{S}}^{\delta_{(S_2 \setminus S), S}} \leq 55 \wedge \sum_i d_{iS} \leq b \right) \end{aligned}$$

4.4.1 Pseudo-Boolean Formulation

Pseudo-Boolean optimization problems are closely related to weighted Max-SAT problems. The constraints are linear equations over those Boolean variables, where a variable with the value $true$ is interpreted as 1 and a $false$ is interpreted as 0. The *pseudo-Boolean optimization* (PBO) problem consists in finding a satisfying assignment to the set of Boolean variables that minimizes a given objective function.

Now we show how to model the previous weighted Max-SAT formulation as a pseudo-Boolean (PB) instance. In this case we do not have weighted clauses but an objective function to be maximized. The original formulation (without cardinality constraints) is the following:

$$\begin{aligned} F_{SM} = & \\ & \textbf{Minimize} \quad B \\ & \textbf{Subject To} \quad C \wedge (B \leq \beta) \wedge \\ & \bigwedge_{S \subseteq S_1, 1 \leq |S| \leq a} \left(\bigvee_{T \subseteq (S_2 \setminus S) \cup S_3, |T \cap S_2| \leq b} C_{\overline{S \cup T}} \wedge (B_{\overline{S \cup T}} \leq \beta) \right) \end{aligned}$$

where

$$B = \sum_{j \in 1..k} \neg l_j \cdot w_j$$

which amounts to the cost of the unsatisfied clauses in W.

Notice that F_{SM} is not a set of PB-constraints, but a conjunction of disjunctions of PB-constraints. As noted in [45], some problems are most directly specified in this latter form. Moreover, notice that we use literals in the objective function B rather than only variables, but this can be avoided by introducing new variables if necessary. Also, $\leq$-constraints can be changed into $\geq$-constraints by negating all constants. Observe that minimizing B is indeed optional, i.e., it is only necessary if we want the robust solution to be optimal in turn. However, since we are adding robustness to Max-SAT, we will assume that optimality is desired for robust solutions.

Similarly, the formula with cardinality constraints would be encoded as the following PB instance:

$$
\begin{aligned}
F^{\nabla}_{SM} = \\
&\textbf{Minimize} && B \\
&\textbf{Subject To} && C \wedge (B \leq \beta) \wedge \\
& && \textstyle\bigwedge_{S \subseteq S_1, 1 \leq |S| \leq a} \left(C_{\overline{S}}^{\delta_{(S_2 \setminus S), S}} \wedge (B_{\overline{S}}^{\delta_{(S_2 \setminus S), S}} \leq \beta) \wedge \ (\nabla^{\delta_{(S_2 \setminus S), S}} \leq b) \right)
\end{aligned}
$$

In this case the OR inside the big AND is substituted by the clauses of the duplicated variables ($\delta_{(S_2 \setminus S), S}$), assuring that the total number of differences regarding the original variables, $\nabla^{\delta_{(S_2 \setminus S), S}}$, is not greater than b.

The formulation can be written similarly as an Integer Programming instance by using $Big - M$ transformations.

4.5 Other robustness notions

Although throughout the report we focus on robustness with respect to good unavailability, we could deal with distinct robustness variants. For instance, for having robustness with respect to bid withdrawal, we simply need to set both S_1 and S_2 to the variables that encode whether bids are winners or losers. Another straightforward variant would be to directly designate the potential breaks to handle: instead of using S_1^{a+}, we could decide what (combinations of) breaks deserve being repaired, which would be a subset of 2^{S_1}. This would be useful if we only want to consider those breaks having a non negligible probability of occurring. We could also think of a robustness notion where each breakable variable has a corresponding set of associated repairable variables. This could serve, for instance, in the presence of scheduling, where one should only look for repairs on the forthcoming assigned resources, or in an auction scenario where it is not permitted to make repairs by switching a winning bid to a loser one. Actually, this last robustness variants are not compatible with the definition of (S_1^a, S_2^b, β)-supermodels for weighted Max-SAT, since the breakage and the repair sets specification requires more information. However, thanks to the high expressiveness of SMT we could easily directly encode such notion of robustness without moving out from the SMT setting, and therefore, there would be no need of changing the underlying solving method.

4.6 Experimentation

In this section we show results of robust solutions using the encoding with the cardinality constraints, and varying the different parameters of the robustness model.

a	b	%opt	%SAT	Yices			Size
				%Sol	SAT Time	UNSAT Time	
1	1	60	90	100	8.84 (2.64)	1.36 (0.37)	3246 vars
		80	2		3.7 (0.00)	1.15 (0.7)	
	2	60	98		24.66 (15.74)	4.89 (0.00)	
		80	2		8.93 (0.00)	1.57 (1.17)	
	4	60	100		106.6 (54.4)	-	
		80	18		18.69 (6.32)	3.4 (3.65)	14714 constr
	8	60	100		37.09 (10.36)	-	
		80	54		20.94 (7.29)	1.48 (1.21)	
2	1	60	0	100	-	10.74 (7.11)	33160 vars
		80	0	100	-	4 (1.47)	
	2	60	-	90	*time out*	38.77 (58.9)	
		80	0	100	-	4.21 (1.59)	
	4	60	-	24	*time out*	95.18 (69.51)	
		80	0	100	-	4.75 (3.71)	149297 constr
	8	60	-	14	*time out*	78.85 (70.61)	
		80	0	100	-	11.3 (35.12)	

Table 4.1: Experimentation results with Yices.

For the experimentation we have used the popular benchmark for combinatorial auctions CATS (Combinatorial Auction Test Suite) [44]. CATS generates instances following different distributions. Given a required number of goods and bids, the distributions randomly select which goods to include in each bid. We have chosen the L7 distribution (the binomial distribution, also described in [24]) of CATS since it generates bids with bundles that are not too large (i.e. with just a few of the resources being auctioned). We wanted to avoid having large bundles because they do not help that much in repairing a broken solution. Moreover, as we saw in the sensitivity analysis chapter, this distribution is particularly affected by resources becoming unavailable.

The number of goods offered in the auction and the number of participating bids were fixed to 20 and 40, respectively. For the rest of parameters we have varied them as follows: number of allowed breakages a in $\{1,2\}$, number of allowed repairs b in $\{1,2,4,8\}$, and the value β has been set according to several percentages of the optimal revenue when robustness is not considered, concretely: $\{80\%, 60\%\}$.

For each configuration of the parameters, we have generated 50 auction instances of the L7 distribution of CATS. Then, for each instance we first find the optimal revenue without considering robustness, and then we solve each of the (a, b, $\%opt$) robust versions of the problem. To solve each instance, we have used several solvers. For SMT we have chosen Yices [21] since it was the winner in the last SMT competition. For the pseudo-Boolean framework we have chosen two solvers: BSOLO v3.1 [46] and SCIP v1.2 (with SoPLEX 1.4.2)[4] since they were the winners in the pseudo boolean Competition of 2009 and 2010 respectively. Finally, for linear programming we have chosen the best commercial solver, CPLEX 12[5] and the best free solver, GLPK 4.35[6]. We have set a solving timeout of 300 seconds. The experiments have been performed on an Intel Core i5 CPU at

[4] http://scip.zib.de/
[5] http://www.ilog.com/products/cplex
[6] http://www.gnu.org/software/glpk

a	b	%opt	%SAT	BSOLO %Sol	BSOLO SAT Time	BSOLO UNSAT Time	Size
1	1	60	90	100	0.25 (0.04)	0.25 (0.02)	3246 vars
		80	2		0.25 (0.00)	0.21 (0.03)	
	2	60	98		0.30 (0.08)	0.50 (0.00)	
		80	2		0.24 (0.00)	0.27 (0.07)	
	4	60	100		0.38 (0.12)	-	
		80	18		0.52 (0.26)	0.30 (0.11)	14714 constr
	8	60	100		0.25 (0.03)	-	
		80	54		0.29 (0.04)	0.21 (0.02)	
2	1	60	0	100	-	29.25 (3.39)	33160 vars
		80	0		-	26.37 (5.40)	
	2	60	4		27.23 (1.79)	30.07 (3.87)	
		80	0		-	28.71 (3.28)	
	4	60	28		41.43 (7.11)	35.77 (8.02)	
		80	0		-	28.86 (3.50)	149297 constr
	8	60	68		31.84 (3.31)	32.07 (4.12)	
		80	0		-	28.96 (3.74)	

a	b	%opt	%SAT	SCIP %Sol	SCIP SAT Time	SCIP UNSAT Time	Size
1	1	60	90	100	68.12 (24.62)	64.9 (4.71)	3246 vars
		80	2	100	77.5 (0.00)	57.01 (19.52)	
	2	60	98	86	165.53 (51.94)	*time out*	
		80	2	90	*time out*	136.33 (50.5)	
	4	60	100	92	191.63 (58.18)	-	
		80	18	12	*time out*	188.82 (55.15)	14714 constr
	8	60	100	84	176.82 (65.1)	-	
		80	54	32	173.66 (40.15)	116.00 (0.00)	
2	*	*	*	0	*time out*		

Table 4.2: Experimentation results with BSOLO and SCIP.

2.66 GHz, with 4GB of RAM, running openSUSE 11.2 (kernel 2.6.31).

Tables 4.1, 4.2 and 4.3 show the average percentage of satisfiability (deviaton in brackets) of the 50 instances (with 20 goods and 40 bids) of each parameter configuration (%SAT), i.e. a robust solution exists, and for each solver, the percentage of instances solved before the timeout (%Sol) and the average time and deviation (in seconds) for solving the instances[7], differentiating the satisfiable and unsatisfiable cases. We also show the average size of the generated instances (number of variables and constraints). A '-' indicates that there were either no SAT or UNSAT instances in a given configuration, and time out indicates that the solver reached the time out of 300 seconds before finding a solution.

The first thing to notice is that the pseudo-Boolean solver BSOLO is the only solver capable of solving all the instances before the timeout. CPLEX does solve all the (1,*) instances but then

[7] Computed only for those solved before the timeout.

its solving performance falls down to about 79% for the (2,*) instances. As for SCIP and GLPK, they could only solve about 74% and 51% of the (1,*) instances, respectively, and none of the (2,*) instances. Moreover, the time taken by SCIP and GLPK is two and three orders of magnitude more than the time taken by BSOLO. Comparing the solving times of BSOLO and CPLEX, we can observe that when solving the (1,*) instances, BSOLO is faster than CPLEX. However, with the (2,*) instances, we can see an interesting effect. When the percentage of optimality is set to 80%, CPLEX is an order of magnitude faster than BSOLO, while with the percentage set to 60%, CPLEX encounters hard problems and can only solve about 58% of the instances. A possible explanation to this behavior is that asking for a revenue of at least 80% of the optimal is a very strict restriction, and so CPLEX rapidly finds out that there is no solution to the problem (note that in all cases where CPLEX solves the instances quickly, the percentage of satisfiability is null). However, when relaxing the revenue constraint, the search space is enlarged and so it takes more time to solve the problem. On the other hand, BSOLO is not affected at all by this parameter, and all its solving times are almost the same.

a	b	%opt	%SAT	CPLEX			Size
				%Sol	SAT Time	UNSAT Time	
1	1	60	90	100	0.54 (0.51)	3.27 (1.15)	3246 vars
		80	2		0.44 (0)	0.74 (0.61)	
	2	60	98		0.76 (0.94)	5.62 (0)	
		80	2		0.38 (0)	2.87 (2.43)	
	4	60	100		0.7 (0.72)	- (0)	
		80	18		15.02 (16.1)	6.24 (14.44)	14714 constr
	8	60	100		0.31 (0.17)	-	
		80	54		1.61 (1.71)	0.4 (0.22)	
2	1	60	0	100	-	30.99 (11.28)	33160 vars
		80	0	100	-	5.44 (1.71)	
	2	60	4	68	19.3 (7.86)	146.2 (83.96)	
		80	0	100	-	5.61 (3.55)	
	4	60	28	10	25.9 (0)	113.9 (96.76)	
		80	0	100	-	5.39 (7.27)	
	8	60	68	54	56.4 (64.48)	185.2 (90.39)	149297 constr
		80	0	100	-	4.46 (4.89)	

a	b	%opt	%SAT	GLPK			Size
				%Sol	SAT Time	UNSAT Time	
1	1	60	90	90	83.58 (74.57)	171.21 (77.63)	3246 vars
		80	2	8	147.49 (0.00)	61.21 (44.01)	
	2	60	98	54	194.98 (77.67)	*time out*	
		80	2	48	107.73 (0.00)	198.34 (61.06)	
	4	60	100	52	230.55 (60.37)	-	
		80	18	28	201.98 (0.00)	213.24 (44.66)	14714 constr
	8	60	100	78	230.14 (43.57)	-	
		80	54	54	250.52 (23.68)	216.24 (50.50)	
2	*	*	*	0	*time out*		

Table 4.3: Experimentation results with CPLEX and GLPK.

We can also observe the effect of allowing more breakages to occur: when increasing a to 2, the solving time increases two orders of magnitude; by contrast, the value of b has not a considerable effect on the solving time. Regarding satisfiability, as mentioned before, we can observe that asking for a robust solution being 80%-optimal is way too strict, and so only a few instances have such a robust solution (percentage that increases as b increases).

We have to point out that the main goal of the experimentation was not to perform a comparison of the solvers, but to show that our reformulation approach can be effectively implemented and solved. Actually, given the declarative nature of our approach, we are not bound to any specific solver, and we can profit from the advances in the state-of-the-art solvers, be them pure pseudo-Boolean solvers, SMT solvers, or more generic integer programming solvers.

Finally we should remark that, although not shown in the results, we have also solved (3,8) instances ($\sim$ 200000 variables and 1 million constraints) using BSOLO, with an average solving time of 180 seconds.

4.7 SUMMARY

In this chapter we have shown that finding a robust allocation can be encoded as an (a, b, β)-super solution for an auction, which can be reduced to modeling the auction as a partial weighted Max-SAT formula and then looking for a supermodel of this formula. This results into the new problem of robust weighted Max-SAT. We have faced these problems following the approach of [25] for SAT. However, since SAT does not allow to easily encode formulas with arithmetic operations, needed to achieve robustness, we have moved the problem to the richer logical framework of Satisfiability Modulo Theories.

We have presented a first approach of robustness that had a very high complexity, growing exponentially with both the parameters a and b. After that, we have presented a second approach based in cardinality constraints that allows to reduce the complexity of the model by removing the parameter b from the exponent. This is especially important, because it means that the complexity of our approach does not depend on the number of repairs, but only on the number of breakages, which is usually assumed to be low. In fact, given that in most of the works $a = 1$ the increase would be linearly in the size of the problem.

In the experiments section we have observed that our formulation can be easily converted into many modelings. Concretely, we have encoded it as SMT, pseudo-Boolean and integer linear programming. We have observed that CPLEX and BSOLO are the best solvers up to now, and are able to solve efficiently our approach of robustness for values of a and b up to 2 and 8 respectively, which are much higher than previous approaches of robustness.

Nonetheless, the increase in size of our reformulation would eventually produce intractable instances, for example when the number of bids or goods are really high, or when the parameters a and b are set too high. In those cases, other approaches could be considered. Instead of solving the problem through reformulation, an alternative could be to modify the search procedure of a combinatorial auction solver in order to directly deal with robustness. Such approach should be more effective in short, but reformulation is certainly more flexible and can take advantage of the future advances in the development of competitive solvers.

CHAPTER 5
Flexible Robustness

In this chapter we add a modification to the previous encoding of robustness that allows some degree of flexibility to the solutions so that we are able to find solutions in hard scenarios and also allows easily switching the characteristics of the solutions between more optimal than robust or more robust than optimal, depending on a single parameter.

5.1 ADDING FLEXIBILITY

The encoding that we have seen in the previous chapter forces the solutions to be (a,b,β) reparable, i.e. if a single breakage is not reparable the problem is not satisfiable, and so there is no robust solution. Although having a completely robust solution would be the ideal situation, it may not be always possible to find it. This fact can be observed clearly in the experiments of the previous chapter, where the percentage of solved instances in some cases was very low (when a high percentage of optimality is required, or when the parameter a is greater or equal to 2). Therefore, in order to be able to obtain solutions in hard instances, we add the possibility of allowing some of the breakages to be left unrepaired, in what we call *flexible robustness*.

The concept of flexibility is also tackled in [30]. Although that paper was focused on defining super solutions for constraint programming, it also pointed out a first notion of flexibility by defining the "most robust solution" for the cases that a robust solution does not exist. The most robust solution is a solution that maximizes the number of repairable variables. In a later paper [29] the same authors also addressed the balance between optimality and robustness by searching for the "most robust optimal solution" and the "optimal robust solution". This work is pretty close to ours but it is rather a preliminary step into flexible robustness and, additionally, does not consider the cost of the repair but only the total number of variables changed.

Our way technique for adding flexibility is based on *soft constraints* [61]. Soft constraints provide a way to model preferences over constraints, so that some constraints can be violated for overconstrained problems, still such violation of constraints is avoided as far as possible.

5.2 FORMALIZATION OF FLEXIBILITY

In order to add flexibility we need to slightly modify the previous encoding in order to make the breakages not mandatorily reparable using a framework to model soft constraints based on weighted variables that activate constraints. To do so, we first relax the repair's constraints so that they are not mandatory to be satisfied. This is done by adding new Boolean variables r_S for each breakage S that will be $true$ if (and only if) the breakage is reparable and $false$ otherwise. These variables will allow to leave some breakages unrepaired by setting the respective r_S to $false$. Then we have to

add a mechanism to count the number of reparable breakages, in order to maximize it (i.e. maximize the repairability of the solution). Hence, Formula 4.5 becomes:

$$F_{SM} = C \wedge W \wedge L \wedge (B \leq \beta) \wedge \bigwedge_{S \in S_1^{a+}} r_S \Leftrightarrow \left(C_{\overline{S}} \wedge B_{\overline{S}} \leq \beta \wedge \sum_i d_{i_S} \leq b \right) \tag{5.1}$$

An alternative formulation would be to change the double implication ($r_S \Leftrightarrow (...)$) by an or ($r_S \vee (...)$). The difference using this alternative is that the number of r_S being $true$ should be minimized instead of maximized. However, current solvers may already perform this transformation if better.

With this new encoding, if a given breakage S is not repairable, its corresponding variable r_S is set to $false$, and this will not affect the satisfiability of the whole formula, since now the only mandatory clauses are those in C (which contains the bid incompatibility clauses and resource availability restrictions) and $B \leq \beta$.

Obviously we do not want all the r_S variables to be $false$ but to maximize the number of r_S variables set to $true$. Therefore, we associate a weight w_S to each variable r_S, and state the weighted clause (r_S, w_S) for each S. The satisfaction of these new weighted clauses will be maximized together with the weights of the bids.

The values of the weights of these variables allow us to express different robustness notions:

- *Optimal solutions as robust as possible (OR)*. In this setting we are concerned with finding, from all the solutions that have the maximum benefit possible, the one that is most robust.
- *Robust solutions as optimal as possible (RO)*. Here we look, among the solutions that have the maximum possible level of robustness, for the one with the highest benefit.
- *Trade-off solutions*. This is a hybrid setting that uses a parameter to determine the desired degree between optimality and robustness.

The difference between the three above settings is in the weights that are set for the bids' prices and for the robustness clauses. For the first setting the weights of the robustness clauses must be much lower than the bids' prices (since we are giving preference to optimality). For example, we could set a weight of 1 for all the robustness clauses, and add the total number of robust clauses, $|S_1^{a+}|$, to the bids' prices. The second setting is the inverse, we are first interested in robustness, and secondly in optimality, so the robustness clauses have to be much more valuable than the bids. Similarly, we could leave the same weights for the bids (their prices) and set the robustness clauses to be the sum of weights of the bids ($\sum_{i=1}^{n} b_i$).

Another alternative would be that the weights of the bids and the robustness clauses do not overlap, and so when summing up all weights, the most significative digits would be only affected by robustness weights, and the least significative would be affected by bids' prices for the RO setting (or vice-versa for the OR setting). For instance, if the sum of the bids ($\sum_{i=1}^{n} b_i$) is X, then the weights of the bids for RO should be from 0 to $10^{\lceil log_{10} X \rceil}$ and the robustness clauses weights beginning from $10^{\lceil log_{10} X \rceil}$, and the other way round for OR.

Alternatively, for the RO and OR cases we could apply a Boolean Multilevel Optimization approach [2], since in these two cases there is a hierarchy in the objectives (first robustness and then optimality, or inversely).

Note that this model can actually generalize the one defined in the previous chapter. We can obtain the same solutions (i.e. full repairability) by using the RO setting and adding a constraint to force the total revenue to be greater or equal to the sum of weights of the robustness clauses, that is, force all breakages to be repaired.

For the hybrid setting we introduce a new parameter α, that defines the trade-off between optimality and robustness, so that $\alpha = 0$ is equivalent to the first setting (OR) and $\alpha = 1$ is equivalent to the second setting (RO). In this hybrid setting, the weights of the robustness clauses and bids prices will be combined, and so they have to be normalized. Thus, we set all the robustness clauses weights to be $w_S = \alpha/n$, where n is the number of robustness clauses, so that the maximum revenue that the solution is able to get from the robustness clauses is α. The prices of the bids are also normalized, setting for each bid i a weight $w_i = (1-\alpha)\cdot p_i/op$ where p_i is the original price of the bid and op is the revenue of the optimal solution (without robustness), so that the maximum revenue that the solution would be able to obtain from the bids weights is $1-\alpha$, and consequently the maximum revenue from the entire formula is 1.

The flexibility that we have added to the model allows us to extend it with new interesting features such as the probability of breakage. It is likely that some breakages are more probable than others, and so they should have different weights. In order to implement this, the weights of the robustness clauses could be multiplied by the probability p_S of the breakage occurring ($w_S = p_S \cdot \alpha$). Actually, with the weights given before, $w_S = \alpha/n$, we were considering all breakages equally probable, i.e. $p_S = 1/n$, for all $S \in S_i^{a+}$.

5.3 EXPERIMENTATION

For the experimentation we have used again the Combinatorial Auction Test Suite (CATS) [44] with the L7 distribution. However, in this case we have already performed experiments in other distributions, namely, arbitrary, paths, matching and regions, obtaining similar results.

We have analyzed the effect of some of the parameters of the (a,b,β)-super solutions, as well as the number of bids participating. Some of the parameters have not been varied, such as a which has been fixed to 1 (it increases considerably the complexity of the instances), and the number of goods being sold, which has been set to 15. We have varied the number of bids in $\{15, 20, 25, 30\}$, the number of repairs b in $\{1, 2, 3, 4, 5\}$ and, finally, the value β has been set according to several percentages of the optimal revenue when robustness is not considered, concretely $\{50\%, 60\%, 70\%, 80\%, 90\%\}$.

For each combination of number of goods and number of bids we have generated 50 auction instances. Then, we have solved each instance without considering robustness to get the optimal revenue, and then we have solved each of the (a,b,β)-robust solutions of the problem, setting β to the appropriate value given the optimal revenue and percentage of optimality.

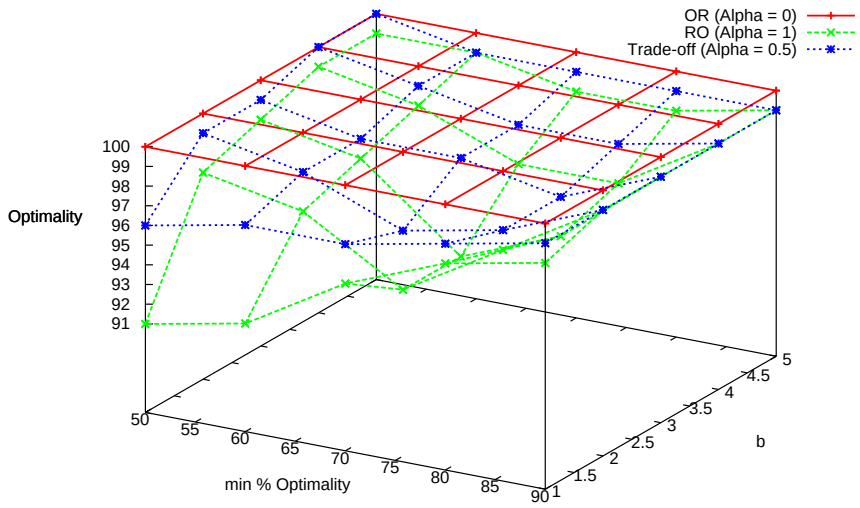

Figure 5.1: Optimality varying b and β.

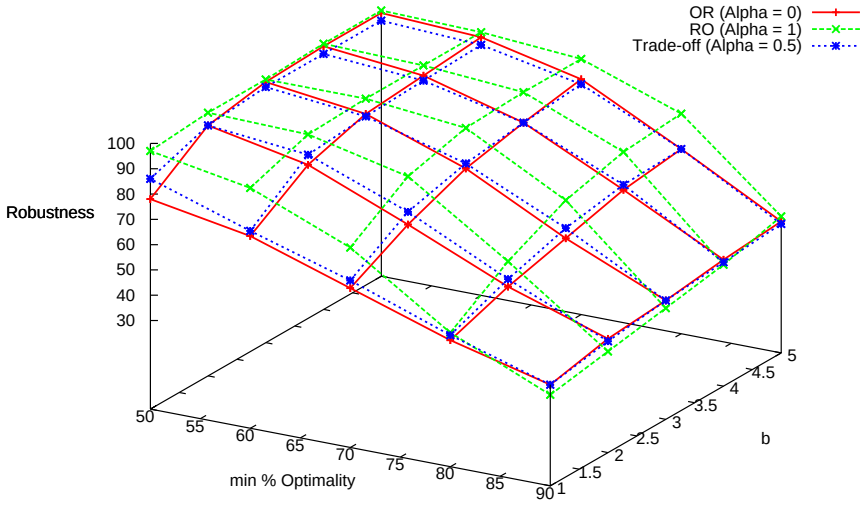

Figure 5.2: Robustness varying b and β.

				OR ($\alpha = 0$)			$\alpha = 0.25$			$\alpha = 0.5$			$\alpha = 0.75$			RO ($\alpha = 1$)		
NB	a	b	β	Solv	OptRev	Rob	Solv	OptRev	Rob	Solv	OptRev	Rob	Solv	OptRev	Rob	Solv	OptRev	Rob
15	1	1	90	100	100	35.47	100	98.89	38.93	100	99.77	34.40	100	99.07	35.87	100	98.88	36.00
			80	100	100	45.60	100	97.04	50.80	100	99.86	44.13	100	98.21	47.07	100	97.03	47.87
			70	100	100	58.67	100	95.65	66.13	100	99.61	58.53	100	97.83	61.87	100	95.65	63.20
			60	100	100	71.07	100	92.42	79.20	100	99.71	70.27	100	97.55	73.33	100	93.09	76.13
			50	100	100	78.53	100	91.37	91.73	100	99.47	79.33	100	96.13	86.13	100	91.81	88.67
		2	90	100	100	40.00	100	100	41.57	100	99.07	42.53	100	99.29	40.44	100	99.43	39.33
			80	100	100	53.87	100	98.38	55.90	100	96.44	59.87	100	96.87	57.71	100	97.55	56.27
			70	100	100	70.00	100	96.61	75.38	100	93.87	79.33	100	95.78	77.40	100	96.54	75.07
			60	100	100	86.67	100	96.80	90.03	100	96.35	94.27	100	97.77	94.27	100	98.43	90.40
			50	100	100	94.13	100	98.43	96.83	100	97.12	98.93	100	98.57	97.04	100	99.12	94.80
		3	90	100	100	42.40	100	99.07	45.33	100	99.82	40.93	100	99.06	42.40	100	99.06	42.40
			80	100	100	59.20	100	95.38	67.20	100	99.69	59.07	100	97.30	63.20	100	95.38	64.27
			70	100	100	79.20	100	93.08	86.93	100	99.78	78.67	100	98.41	81.47	100	95.03	83.47
			60	100	100	93.20	100	97.85	96.40	100	99.80	91.47	100	98.96	92.80	100	97.83	93.47
			50	100	100	98.93	100	98.70	99.73	100	100	96.00	100	99.82	96.27	100	98.68	96.80
		4	90	100	100	45.47	100	99.58	46.64	100	99.07	47.73	100	99.66	45.08	100	99.06	44.80
			80	100	100	65.73	100	98.49	68.98	100	95.38	71.20	100	96.63	69.69	100	97.30	67.20
			70	100	100	84.27	100	96.60	86.00	100	93.08	88.93	100	93.08	86.01	100	98.41	84.67
			60	100	100	95.33	100	99.15	96.11	100	97.85	96.93	100	98.22	95.31	100	98.96	93.47
			50	100	100	99.60	100	98.94	99.68	100	98.70	99.73	100	99.07	97.74	100	99.82	96.67
		5	90	100	100	47.87	100	99.62	49.07	100	99.91	45.60	100	99.75	46.00	100	99.61	46.13
			80	100	100	68.53	100	98.02	72.13	100	99.66	67.87	100	99.23	68.40	100	98.02	69.20
			70	100	100	88.00	100	98.54	89.87	100	99.94	85.87	100	99.59	86.40	100	98.54	86.93
			60	100	100	97.20	100	99.43	97.60	100	99.99	94.40	100	99.88	94.53	100	99.88	94.53
			50	100	100	99.60	100	99.82	99.73	100	100	96.67	100	100	96.67	100	99.81	96.80

Table 5.1: Experimentation results: averages of 50 auction instances of 15 bids for each configuration.

				OR ($\alpha = 0$)			$\alpha = 0.25$			$\alpha = 0.5$			$\alpha = 0.75$			RO ($\alpha = 1$)		
NB	a	b	β	Solv	OptRev	Rob	Solv	OptRev	Rob	Solv	OptRev	Rob	Solv	OptRev	Rob	Solv	OptRev	Rob
20	1	1	90	100	100	28.53	100	98.11	32.80	100	99.66	29.83	100	98.59	31.73	100	98.11	32.13
			80	100	100	44.40	100	96.71	50.27	100	99.61	47.15	100	98.51	48.67	100	96.71	49.60
			70	100	100	61.07	100	94.63	72.67	100	98.88	65.70	100	95.83	71.07	100	94.62	72.00
			60	100	100	80.13	100	93.92	89.60	100	99.34	85.09	100	98.67	86.67	100	95.82	88.40
			50	100	100	91.73	100	97.36	99.47	100	99.29	95.74	100	97.83	98.67	100	97.83	98.67
		2	90	100	100	32.93	100	99.12	34.47	100	98.45	36.67	100	99.02	35.95	100	98.78	35.73
			80	100	100	56.27	100	97.47	60.00	100	94.78	65.20	100	96.02	65.20	100	97.50	62.80
			70	100	100	79.07	100	96.49	85.56	100	93.94	89.33	100	94.97	88.88	100	96.58	87.47
			60	100	100	93.87	100	98.05	96.81	100	96.63	99.07	100	97.79	98.59	100	98.35	97.60
			50	100	100	99.07	100	100	99.01	100	98.95	100	100	99.03	99.68	100	99.67	99.07
		3	90	100	100	35.33	100	98.10	40.40	100	99.65	37.53	100	98.75	39.20	100	98.09	39.73
			80	100	100	65.60	100	94.94	74.40	100	99.84	68.25	100	97.72	72.00	100	94.94	73.73
			70	100	100	86.00	100	94.06	94.80	100	99.25	89.00	100	97.93	92.00	100	94.65	94.00
			60	100	100	97.87	100	98.56	99.73	100	99.91	98.08	100	99.53	98.67	100	99.02	98.93
			50	100	100	99.87	100	99.91	100	100	100	99.18	100	99.91	99.33	100	99.91	99.33
		4	90	100	100	38.40	100	99.33	40.50	100	98.45	43.73	100	98.61	43.17	100	98.79	42.80
			80	100	100	71.87	100	96.72	75.41	100	94.89	80.13	100	96.39	78.84	100	97.63	77.60
			70	100	100	91.87	100	98.19	94.70	100	95.95	96.53	100	97.54	95.81	100	99.43	94.40
			60	100	100	99.33	100	99.02	98.99	100	98.79	100	100	100	99.11	100	100	98.80
			50	100	100	100	100	100	100	100	100	100	100	100	100	100	99.99	99.33
		5	90	100	100	42.00	100	98.69	46.13	100	99.60	44.05	100	99.02	45.20	100	98.69	45.47
			80	100	100	77.33	100	96.18	83.60	100	99.77	79.04	100	98.13	81.60	100	96.18	82.93
			70	100	100	94.00	100	98.24	96.93	100	99.89	94.78	100	99.59	95.47	100	98.66	96.13
			60	100	100	99.60	100	99.72	100	100	99.99	98.90	100	99.71	99.33	100	99.71	99.33
			50	100	100	100	100	100	100	100	100	99.31	100	100	99.33	100	99.99	99.33

Table 5.2: Experimentation results: averages of 50 auction instances of 20 bids for each configuration.

				OR ($\alpha = 0$)			$\alpha = 0.25$			$\alpha = 0.5$			$\alpha = 0.75$			RO ($\alpha = 1$)		
NB	a	b	β	Solv	OptRev	Rob	Solv	OptRev	Rob	Solv	OptRev	Rob	Solv	OptRev	Rob	Solv	OptRev	Rob
25	1	1	90	100	100	28.53	100	99.72	29.81	100	99.33	30.53	100	98.97	30.80	100	98.97	31.20
			80	100	100	43.20	100	99.54	47.04	100	98.98	47.07	100	97.17	48.27	100	97.17	48.67
			70	100	100	62.13	100	98.57	71.70	100	97.66	73.07	100	96.18	74.13	100	95.68	74.67
			60	100	100	81.20	100	99.05	88.93	100	98.57	88.93	100	96.74	90.13	100	96.75	90.53
			50	100	100	88.53	100	98.87	95.22	100	97.92	96.53	100	96.77	97.33	100	96.34	97.87
		2	90	100	100	32.93	100	100	34.04	100	98.45	36.67	100	98.67	36.01	100	98.78	35.73
			80	100	100	56.27	100	99.66	60.12	100	94.78	65.20	100	96.98	63.39	100	97.50	62.80
			70	100	100	79.07	100	98.11	81.59	100	93.94	89.33	100	94.18	88.81	100	96.58	87.47
			60	100	100	93.87	100	97.38	95.81	100	96.63	99.07	100	97.05	97.14	100	98.35	97.60
			50	100	100	99.07	100	99.13	99.01	100	98.95	100	100	99.15	99.32	100	99.67	99.07
		3	90	100	100	34.40	100	99.80	36.48	100	98.74	38.40	100	98.21	38.80	100	98.21	39.20
			80	100	100	60.00	100	98.95	66.16	100	95.98	72.00	100	93.75	73.60	100	93.75	74.00
			70	100	100	84.67	100	99.34	91.19	100	97.82	93.60	100	96.88	94.27	100	95.41	95.07
			60	100	100	97.60	100	99.92	97.86	100	99.57	98.67	100	98.95	98.93	100	98.05	99.60
			50	100	100	99.47	100	100	97.11	100	99.99	99.07	100	99.20	99.47	100	99.21	99.87
		4	90	100	100	38.40	100	99.66	40.41	100	98.45	43.73	100	98.65	43.47	100	98.79	42.80
			80	100	100	71.87	100	96.82	75.58	100	94.89	80.13	100	96.19	78.30	100	97.63	77.60
			70	100	100	91.87	100	98.84	93.10	100	95.95	96.53	100	97.38	95.09	100	99.43	94.40
			60	100	100	99.33	100	99.00	99.67	100	98.79	100	100	100	98.94	100	100	98.80
			50	100	100	100	100	100	100	100	100	100	100	100	100	100	99.99	99.33
		5	90	100	100	43.07	100	99.36	46.29	100	98.28	48.40	100	97.59	48.93	100	97.60	49.33
			80	100	100	75.87	100	99.35	79.50	100	98.29	80.93	100	96.13	82.40	100	96.13	82.80
			70	100	100	94.40	98	99.70	95.85	100	99.36	96.00	100	99.03	96.27	100	98.45	96.80
			60	100	100	99.33	100	100	98.99	100	99.78	99.20	100	99.78	99.20	100	99.79	99.60
			50	100	100	99.87	100	100	99.50	100	99.95	99.60	100	99.95	99.60	100	99.95	100

Table 5.3: Experimentation results: averages of 50 auction instances of 25 bids for each configuration.

				OR ($\alpha = 0$)			$\alpha = 0.25$			$\alpha = 0.5$			$\alpha = 0.75$			RO ($\alpha = 1$)		
NB	a	b	β	Solv	OptRev	Rob	Solv	OptRev	Rob	Solv	OptRev	Rob	Solv	OptRev	Rob	Solv	OptRev	Rob
30	1	1	90	100	100	24.27	100	99.78	26.27	100	98.78	28.00	100	98.27	28.40	100	98.25	29.87
			80	100	100	46.27	100	99.69	49.73	100	98.37	52.40	100	96.83	53.33	100	96.84	53.33
			70	100	100	68.27	100	99.42	72.93	100	98.09	66.80	100	96.36	76.93	100	96.38	76.93
			60	100	100	87.20	100	99.56	90.80	100	98.22	90.40	98	96.68	93.07	100	96.31	94.80
			50	100	100	96.13	100	99.79	91.20	100	99.02	97.87	100	98.36	99.60	98	98.37	99.60
		2	90	100	100	32.93	100	99.33	34.72	100	98.45	36.67	100	98.61	36.20	100	98.78	35.73
			80	100	100	56.27	100	97.44	60.28	100	94.78	65.20	100	95.99	63.19	100	97.50	62.80
			70	100	100	79.07	100	95.52	82.46	100	93.94	89.33	100	95.73	88.64	100	96.58	87.47
			60	100	100	93.87	100	97.76	95.42	100	96.63	99.07	100	97.22	98.78	100	98.35	97.60
			50	100	100	99.07	100	99.69	99.54	100	98.95	100	100	99.31	100	100	99.67	99.07
		3	90	100	100	33.07	100	99.24	37.47	100	97.93	39.73	100	97.64	40.00	98	97.64	40.00
			80	100	100	67.07	96	99.30	72.67	96	96.99	74.67	100	93.76	78.80	96	93.79	78.80
			70	98	100	90.93	96	99.48	91.73	94	98.50	91.07	100	96.57	98.27	100	96.58	98.27
			80	98	100	99.73	96	99.94	93.60	98	99.94	97.87	100	99.93	100	100	99.94	100
			50	100	100	100	100	99.99	93.60	88	99.99	97.87	100	99.99	100	100	100	100
		4	90	100	100	38.40	100	99.70	40.23	100	98.45	43.73	100	98.99	43.14	100	98.79	42.80
			80	100	100	71.87	100	93.87	77.47	100	94.89	80.13	100	96.31	78.60	100	97.63	77.60
			70	100	100	91.87	100	96.25	93.48	100	95.95	96.53	100	97.35	95.55	100	99.43	94.40
			60	100	100	99.33	100	99.50	99.63	100	98.79	100	100	99.33	99.00	100	100	98.80
			50	100	100	100	100	100	100	100	100	100	100	100	100	100	99.99	99.33
		5	90	100	100	42.27	94	99.45	46.00	98	98.54	48.00	100	98.38	48.13	100	98.40	48.13
			80	100	100	80.80	98	99.28	83.07	96	97.97	85.20	100	95.40	88.93	100	95.42	88.93
			70	96	100	98.27	94	99.89	99.07	98	99.77	99.33	100	99.24	99.60	100	99.24	99.60
			60	98	100	100	86	99.99	91.47	98	99.99	97.87	100	99.99	100	100	100	100
			50	100	100	100	94	99.99	93.60	98	99.99	97.87	100	99.98	100	100	100	100

Table 5.4: Experimentation results: averages of 50 auction instances of 30 bids for each configuration.

Tables 5.1 to 5.4 show the detailed results of the experiments performed with distribution L7. The results with 15, 20, 25 and 30 goods are shown for five different values of $b = \{1, 2, 3, 4, 5\}$. Each row shows the average of the 50 instances of each case. The column $Solv$ indicates the percentage of instances where the solver finds the solution before the timeout (set to 1000 seconds). The column $OptRev$ indicates the percentage of optimality, computed only for the satisfiable instances. The column Rob (robustness) indicates the percentage of robustness clauses satisfied. Each of these values has been calculated in five different settings: optimal solutions as robust as possible OR (equivalent to $\alpha = 0$), robust solutions as optimal as possible RO (equivalent to $\alpha = 1$), and trade-off solutions with different balance points $\alpha = \{0.25, 0.5, 0.75\}$.

We observe that, in general, when the value of β decreases the value of Rob increases, since the problem is less restricted and it is easier to find robust solutions. Also, when the size of the repair (b) increases, obviously the value of Rob is also increased, since more repairs are allowed. We can also see that when the value of α increases the percentage of optimality decreases (since the solutions are more robust, and thus less optimal), and conversely the robustness increases. Regarding the percentage of solved instances, we also see that it logically decreases when the complexity of the instances increases (with 30 bids and high values of b); still in these experiments with 30 bids, the percentage of solved instances is in all cases over 94%.

Figure 5.1 graphically shows the results with 15 bids. The X axis represents the minimum revenue percentage of the solution (and its repairs), the Y axis indicates the size of the repair b, and the Z axis plots the percentaged revenue of the solution (i.e. dividing the revenue of the solution by that of the optimal solution without robustness). Obviously the graph with $\alpha = 0$ is a plane with the 100% of optimality. When α is increased we see that the revenue of the solutions decreases but with a slow rhythm (the worst solution is still 91% optimal). Therefore, in some cases looking for robust solutions may be a good option since we see that the solution found is already very close to the optimal.

Figure 5.2 compares the same settings regarding its robustness, so in this case the Z axis represents the percentage of robustness clauses satisfied. In this case we do not see a plane for the RO because it is not always possible to find solutions that are 100% repairable, but still the RO graph is almost in all the cases the one obtaining most robustness. It is interesting that in this case the differences between the graphs are higher than in the results regarding optimality, with the RO graph obtaining 20 percentual points more robustness than OR in the instances with lower values of α.

Figures 5.3 and 5.4 show the results of optimality and robustness respectively, with different values of α. Here the number of goods is also 15 and the minimum percentage of optimality has been fixed to 50%. We can see that as the value of α is increased the optimality decreases and the other way around for the robustness, when α increases, robustness is generally also increased. Again, we observe that the optimality is much less affected than the robustness.

These results show that the optimality of the solutions when looking for robust solutions is not affected in the same level as the robustness when searching for optimal solutions. Therefore, for decision makers worried about the optimality of the solutions, it would be not bad to incorpore robustness, since the optimality would remain very close (90% of optimal in the worst case of our experiments). However, when looking for optimal solutions, the robustness is highly affected. This is a factor that decision makers wanting robust solutions should consider. The loss of optimality when searching robust solutions is lower than the loss of robustness when looking for optimal solu-

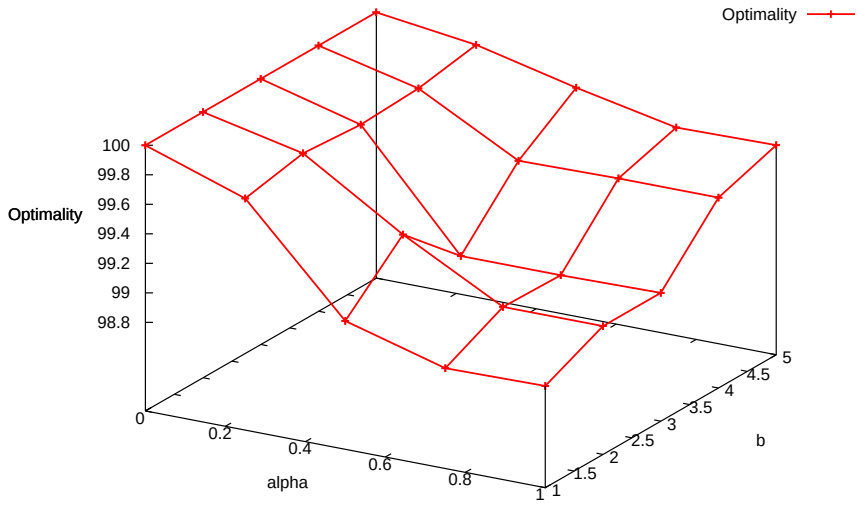

Figure 5.3: Optimality varying b and α.

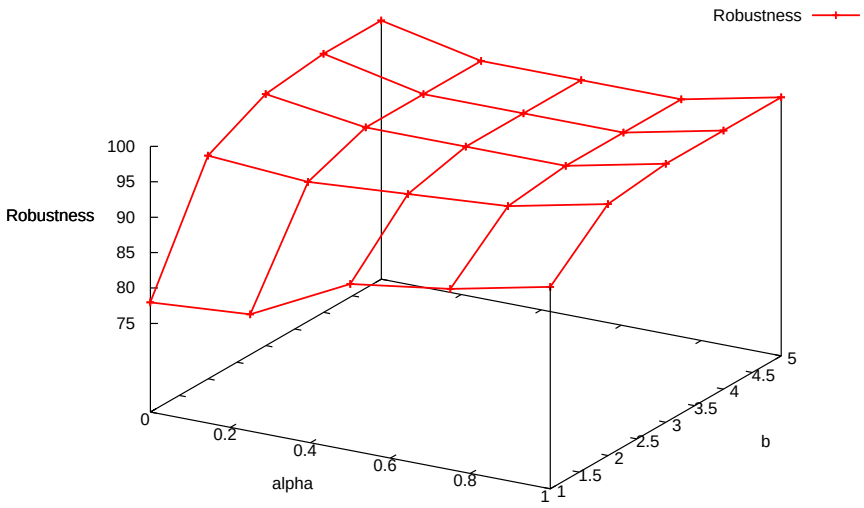

Figure 5.4: Robustness varying b and α.

tions. We have performed more experiments with other distributions (arbitrary, paths, matching and regions) obtaining similar results. This fact was also seen in the work of [34], therefore we think that our observations can be extrapolated in general to other domains.

5.4 SUMMARY

In this chapter we have presented a flexible robustness model. The flexibility allows to find solutions in overrestricted situations where no complete robust solution is possible. In such case this model is able to find the "most robust solution", i.e. the solution that satisfies most robustness clauses.

Conversely, we can find also the most optimal robust solution, i.e. the optimal solution that is most robust. Moreover, a parameter α is defined in order to easily set the desired trade-off between optimality and robustness.

In the experimentation section we have analyzed the relationship between robustness and optimality, i.e. with different values of α, and the conclusion is that the loss of optimality when searching robust solutions is lower than the loss of robustness when looking for optimal solutions. This is a result that should be taken into account when deciding the most appropriate value of α in real world problems, depending on the preferences of the decision maker (e.g. risk-seeking, risk-averse, etc.).

CHAPTER 6
Incentive Compatibility

Incentive compatible methods ensure that malicious participants cannot use strategies for manipulating the results. This chapter analyzes the strategy-proofness of our approach for robustness. We first extend the proof of non-incentive compatibility given for the bid-withdrawal problem in [37] which captures a restricted version of our problem. Then we provide two ways of looking for incentive compatibility for the generic version of the problem.

6.1 INCENTIVE COMPATIBLE MECHANISMS

Auction mechanisms with the feature that the best strategy for the bidders is to bid truthfully are called *incentive-compatible*. This feature makes impossible -theoretically- for the bidders to strategically manipulate the auction in order to gain money by decreasing the final prices of the items. This manipulation would be a gain for them but a loss for the auctioneer, therefore strategy-proof mechanisms are preferred in real world applications of auctions. Furthermore, it has been proved that no untruthful mechanism achieves better outcome than any truthful (non-manipulable) mechanism [47]. This is another reason why in many applications only incentive compatible mechanisms are considered.

The design of truthful mechanisms for combinatorial auctions is a hard task. For example, the generalized Vickrey auction (VCG) mechanism [71, 13, 27] guarantees that the best strategy for the bidders is to bid truthfully. However, its computational complexity is very high. For the robustness mechanism shown in the previous chapters, to prove either that the mechanism is incentive compatibility or that it is not, is even harder. For that reason, we will first show that the robustness mechanism is not incentive compatible in a restricted case where only a subset of the resources can fail (the breakage set does not include all the items). For the general case we will provide two procedures to support the idea that the robustness mechanism is actually incentive compatible in practical cases.

6.2 NON-INCENTIVE COMPATIBILITY WITH RESTRICTED ROBUSTNESS

In our model of robustness, the parameter a representing the size of the break included all the resources. For example, $a = 1$ meant that each one of the resources could fail (but not two of them at the same time). In the restricted robustness setting, we limit the set of resources that can fail. This is similar to the work of Alan Holland [37], where he considered that only “brittle” bids could be withdrawn. He provided the example shown in Table 6.1 to prove the non-incentive compatibility of robust solutions (weighted super solutions).

Bid	A	B	C	AB	BC	Withdrawal probability
v_1	13	0	0	0	0	0.02
v_2	0	12	0	0	0	0.03
v_3	0	0	8	0	0	0.04
v_4	0	0	0	33	0	0.15
v_5	0	0	0	0	24	0.05

Table 6.1: Combinatorial auction example of [37].

The proof of non-incentive compatibility is made using the necessary condition of monotonicity that incentive-compatible mechanisms exhibit. This condition means that if a winning bid increases its price, then it remains in the solution. Therefore, if in a given auction mechanism a bid increasing its price drops out of the solution then it means that the auction mechanism is not incentive-compatible.

For the example in [37] the threshold for the withdrawal probability (α) was set to 0.1, and therefore the only bid that has a probability higher than α (brittle bid) is the one of the fourth bidder. In this example the minimum acceptable revenue was set to 34, therefore the optimal robust solution is $\{v_1, v_5\}$ with a total revenue of 37, because the solution $\{v_3, v_4\}$ that has a higher revenue (41) cannot be repaired to form a solution of at least 34; its repair solution $\{v_1, v_2, v_3\}$ gets only 33.

Then, supposing that the first bid v_1 increases its value to 16, it turns out that now the solution $\{v_3, v_4\}$ is reparable by $\{v_1, v_2, v_3\}$, getting a revenue of 36. Therefore the new optimal robust solution becomes $\{v_3, v_4\}$. This implies that the bid v_1 that was in an optimal robust solution, when increasing its value, drops out of the solution, violating the monotonicity requirement of incentive-compatible mechanisms.

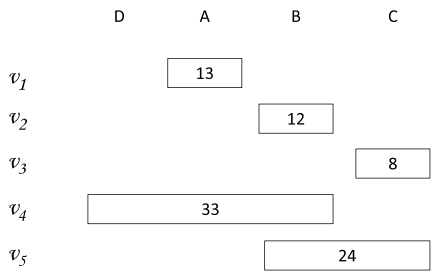

Figure 6.1: Modified example.

In our case, we can slightly modify the previous example in order to prove the non monotonicity of our restricted robustness mechanism as shown -more visually- in Figure 6.1, where each column is an individual item and the rectangles indicate the items included in each bid and its price. Let us add a new item D to the fourth bid v_4. Let us suppose that there is only one resource that can become unavailable, D. In such situation, the only bid that can fail is v_4, converting the example to be exactly the same as before. Therefore, the same proof holds, and this restricted version of our

robustness mechanism is not incentive-compatible.

6.3 Incentive Compatibility in the General Case

The previous example actually proves that when some (a subset) of the resources are brittle, the robustness mechanism is not incentive-compatible. However, the problem may not be the same when all the resources are brittle, and the previous counter-example cannot be easily generalized. Let us mention that in the case of [37], they did not consider the case of all bids being brittle, which we think that could have the same problem, i.e. the proposed counter-example does not necessarily imply that when all the bids are brittle the problem is not incentive-compatible.

In this section we propose two procedures to find a counter-example of monotonicity for the generic case of robustness where every resource can fail. The first procedure consists in the formulation of a CP model to find a counter-example (if it exists) with a given number of items. The second procedure is an iterative search process that systematically tests all the possible bid profiles in order to check whether they constitute or not a counter-example.

6.3.1 Counter-example Model

The problem of finding a counter-example of monotonicity (and consequently, of non-incentive-compatibility) in the general case of robustness can be modeled as a CSP. Given a fixed number of items k, the goal of such CSP is to find a combinatorial auction instance whose robust solution is not monotonic, i.e. such that if some bid of the optimal robust solution increases its price, then the new optimal robust solution does not include that bid. Therefore, the output of the CSP are the bids that compose such combinatorial auction. The input of the CSP will be the maximum number of items that can be auctioned (k).

We first define the bid variables $B = \{B_1, ..., B_n\}$. These are integer variables that define the price of the bids of the auction. The number of bids n is all the possible combinations of k items, that is, $2^k - 1$. The price of the bid can be any natural number. Note that a price of zero is allowed, which would actually mean that the bid does not take part in the auction. This is necessary because it is not required for the counter-example auction to be composed of all the possible bids (item combinations) but only a subset of them. After that, we also need the variables that will specify the optimal robust solution $X = \{X_1, ..., X_n\}$, which are Boolean variables indicating which of the bids are in the solution ($true$) and which are not ($false$). Then, the CSP consists in finding values for B (an instance of a combinatorial auction) such that X is its optimal robust solution, but X is not monotone, i.e. there is a bid which, if its price is increased, it drops out from the optimal robust solution. Therefore, the CSP program to find a counter example is as shown in the following CSP formulation (CSP 1), which uses the CSP functions *OPTIMAL-ROBUST-SOLUTION(B)* and *NON-MONOTONIC(X)* defined next.

CSP 1 COUNTEREXAMPLE

find values for B such that:

$X \leftarrow$ OPTIMAL-ROBUST-SOLUTION(B)

NON-MONOTONIC(X)

The function *OPTIMAL-ROBUST-SOLUTION(B)* gets an instance of an auction B and finds its robust optimal solution X. Since X has to be the optimal robust solution, we have to make that X is a solution (it satisfies the bid incompatibility restrictions), it is a robust solution (it is repairable doing at most b changes, for each break of size at most a), and it is optimal (it is the maximal revenue robust solution). Therefore we define the function *OPTIMAL-ROBUST-SOLUTION(B)* as shown in CSP 2.

CSP 2 OPTIMAL-ROBUST-SOLUTION

find values for X such that:

SOLUTION(X)

ROBUST(X, B)

BEST(X, B)

Where *SOLUTION(X)* is a function that verifies that X is a solution by checking that no bids sharing the same item are set to $true$ as shown in Algorithm 3; and *ROBUST(X, B)* examines if X is robust by checking that each possible break S is repairable, which means that an alternative solution $X_{\overline{S}}$ (a solution where the values of the variables in the breakage set S are negated) can be found, and it does not differ from the original solution X in more than b changes, as shown in Algorithm 4.

Algorithm 3 SOLUTION(X)

return $\sum_{i|g\in G_i} X_i \leq 1 \qquad \forall i \in [1..k]$

Algorithm 4 ROBUST(X, B)

$z = true$

$\forall S \in S^{a+}$

$z = z \wedge SOLUTION(X_{\overline{S}})$

$z = z \wedge MIN\_REVENUE(X_{\overline{S}}, B, \beta)$

$z = z \wedge \sum_{i=1}^{n} |X_i - X_{\overline{S}_i}| \leq b$

return z

Note that since the goal of the CSP is actually to find an auction instance whose solution is not monotonic, we cannot define an objective function to be maximized (turning the CSP into a COP) in order to find the optimal solution (as we would make if the problem goal of the problem was to find the optimal solution). Therefore, we have to assure that X is the optimal solution by brute force, that is, by imposing that X is better than any other possible solution, as shown in Algorithm 5.

Algorithm 5 BEST(X,B)

$\forall X' \neq X$

$z = true$

$z = z \wedge \sum_{i=1}^{n} (X'_i \cdot B_i) < REVENUE(X)$

$z = z \wedge \neg SOLUTION(X')$

$z = z \wedge \neg ROBUST(X', B)$

if z return $false$

return $true$

The function *MIN_REVENUE(X,B,β)* assures that the solution X has a revenue of at least β as shown in Algorithm 6; and the function *REVENUE(X)* computes the revenue of the solution X as shown in Algorithm 7.

Algorithm 6 MIN_REVENUE(X,B,β)

return $REVENUE(X) \geq \beta$

Algorithm 7 REVENUE(X)

return $\leftarrow \sum_{i=1}^{n} (X_i \dot{B}_i)$

Finally, we define the function *NON-MONOTONIC(X)* to force that when a given winning bid from X increases its price by δ then it gets out from the solution, as shown in CSP 8. To do so, we define the prices of the bids $B' = \{B'_1, ..., B'_n\}$, after some bid B_j ($j \in \{1..n\}$ is the index of the bid that increases its price) has increased its price. Then we state that the new robust solution Y, i.e. such that $SOLUTION(Y) \wedge ROBUST(Y, B') \wedge BEST(Y, B')$, satisfies that some of the previously winning bids is now a losing bid, i.e. such that $\exists i : X_i = 1 \wedge Y_i = 0$.

CSP 8 NON-MONOTONIC

find values for B' such that:

$\forall i \neq j (B'_i = B_i) \wedge (B'_j = B_j + \delta) \wedge (X_j = 1)$

$SOLUTION(Y, B') \wedge ROBUST(Y, B') \wedge BEST(Y, B')$

$\exists i : X_i = 1 \wedge Y_i = 0$

Note that the complexity of this model is extremely high. This is mainly because of the *BEST(X,B)* function, which has to assure that the solution X is the best possible, implying checking all the other possible solutions with n bids, that is 2^n possibilities (where $n = 2^k$). The exact number of variables can be computed with the expression shown in Equation 6.1.

$$NV = 2[(2^k - 1) + 2^{2^k} \cdot 2^k] \tag{6.1}$$

For $k = 1$ the total number of variables is 18, for $k = 2$ it grows to 134, for $k = 3$ the number is 4110, and for higher values of k the number of variables is extremely high ($2 \cdot 10^6$ for $k = 4$, and $2 \cdot 10^{11}$ for $k = 5$).

Therefore, this model is not very useful in practice, as it grows extremely exponentially and there is not any solver able to solve it when the number of items is more than, say, three. However, we keep this model as it is interesting theoretically, and is actually the best way to prove non-monotonicity in the general case if any solver could eventually find solutions to it efficiently.

In the following section we show a more practical (although not complete) procedure for finding such counter-example.

6.3.2 Counter-example Iterative Search

The iterative search method consists in a procedure that systematically generates and checks all the possible auction instances, in order to find a counter-example of monotonicity. For each auction instance, the procedure checks for all the possible assignments of prices to the bids (up to a given limit), called *bid profiles*. This model will eventually find a counter-example if it exists.

More concretely, the procedure begins by generating an instance of a combinatorial auction, composed by bids with its respective items but without determined prices for the bids. After that, the *bids profiles* are created with all the possible combinations of prices to the bids (up to a given limit). With each bid profile, the robust solution is found with the model previously presented in Section 4.3.3. Next, all the bids from the optimal robust solution are taken one by one and their prices are increased; the new auction is solved again and the solution is analyzed in order to check whether a counter-example has been achieved, i.e. by observing if there is any of the winning bids that increasing its price turns to a losing bid. A diagram of this approach is shown in Figure 6.2.

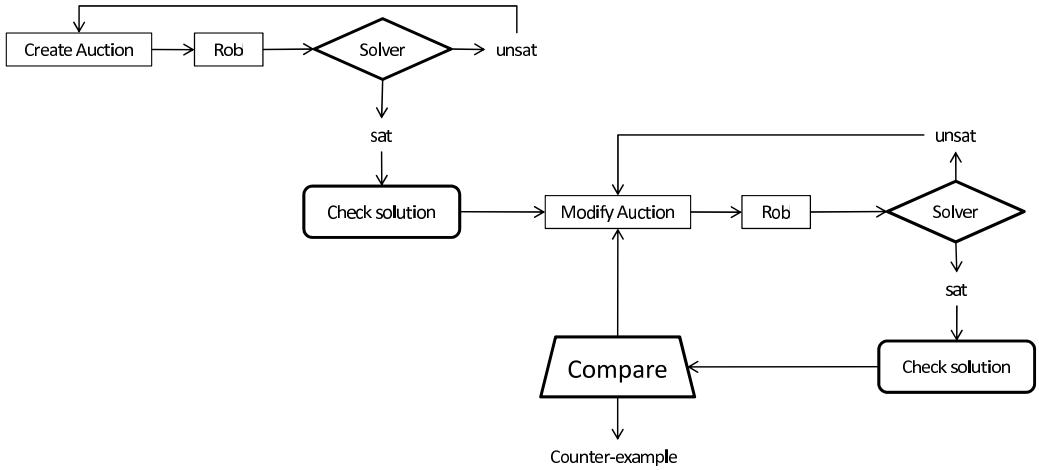

Figure 6.2: Iterative procedure for finding a counter-example.

First of all, an auction is created. The first module, "create auction", generates progressively all the possible combinatorial auction configurations, that is, given a number of items k, generates all the possible bid combinations using these bids, i.e. the enumerative combinatorics of k items. With each bid combination, the bid profiles are created, by putting all the possible combinations of prices to the bids (in a given interval). Therefore, n^p bid profiles are generated, where n is the number of bid combinations, and p is the range size of the prices interval. Symmetry breaking is considered while generating the bid profiles, in order to avoid checking duplicated profiles.

Each bid profile is encoded with the robustness model (module Rob), and passed to the solver which can be either SMT, ILP or pseudo-Boolean, depending on the generated model. If the model is unsatisfiable the next bid profile is checked, otherwise if it is satisfiable, then the solution is checked in order to find if by increasing the price of some winning bid, it drops out of the solution, which would imply that a counter example has been found. Therefore, for each one of the winning bids, a new auction is created copying the original auction and increasing the price of the winning bid by a given bid increment. Again, the auction is encoded adding robustness and solved using a solver. If the result is satisfiable, then the solution is checked and compared with the previous one in order to find if the winning bid is now a losing bid. The process is iterated until a counter example is found.

Using this procedure, we have proved that no counter example exists with less than 4 items, with a price interval between 0 and 10, and a winning bid increment of 1. For higher number of items, or wider prices intervals, this procedure is not so successful due to its exponential increase in computational time. However, we have tried setting bid combinations manually (susceptible of composing a counter-example) with 8 items at most, and then generating all the bid profiles and a counter example has neither been found.

This means that either such counter example does not exist and the robustness mechanism is indeed incentive compatible, or that the counter example would be even more complex. This is actually a good result, since as we saw in the sensitivity analysis chapter, large auctions tend to be inherently robust and therefore, robustness would not be applied in those cases where manipulation strategies could exist.

6.4 SUMMARY

Incentive compatibility issues are a crucial point in the design of mechanisms for auctions, since only strategy-proof mechanisms are actually used in real-world applications because of their impossibility of being manipulated by the participants. A necessary condition that incentive-compatible mechanisms must hold is monotonicity, which can be conversely used to prove non-monotonicity.

In this chapter we have analyzed the incentive compatibility of our robustness approach. We have firstly seen that in a restricted version, where only some of the resources are brittle, the method is not incentive compatible. We have provided a counterexample of monotonicity to prove that, following the spirit of the proof of non-monotonicity for the problem of bid withdrawal in [37].

For the generic case, however, no handmade counterexample has been found. Therefore, we have tried two different ways to find such counterexample: via reformulation and by iterative search. Through reformulation, we generate a CSP model that provably finds a counter-example (if it exists), given a maximum number of items. On the other hand, we have provided (and implemented) an iterative procedure which checks all the possible bid combinations one by one in order to find such counter-example.

The first approach has turned out to be not practical since the complexity of the model is highly exponential. The second approach, although more practical, has not found any counterexample up to now. There are two possible explanations for that, the first one is that the counter-example does not exist and therefore the method is actually incentive compatible, the second option is that the counter-example exists but it is very intricate. Both alternatives are good, since as we saw in the sensitivity analysis, large auctions tend to be inherently robust and therefore, possible manipulations in such cases could be avoided as robustness would not be used.

CHAPTER 7
Robustness for Recurrent Auctions

In this chapter we examine how to achieve robustness in a sequence of auctions. The robustness mechanism explained in the previous chapters using super solutions could also be applied, repeating it on each auction. However, in a sequence of auctions that are repeated through time with similar resources and participants, a problem arises when the resources are public and uncontrolled (see Section 2.1): the problem of agents using the resource without authorization. Therefore, the problem of resources that become unavailable is converted to agents that use the resources without authorization. We take a new approach to improve robustness that consists in learning the behavior of the agents using a trust model and using the learnt parameters in each individual auction in order to improve its robustness even more.

7.1 RECURRENT AUCTIONS

In some domains the allocation of resources to bidders are made for a specific time only [42]. Hence, short-term contract is often used in those markets. When the time of the contract expires the resource allocated becomes free. Then the auctioneer needs to allocate the resource to bidders again. Consequently, these short-term contracts are continuously repeated in what is known as a *recurrent auction*. This recurrence is also from the point of view of bidders, since each bidder repeatedly requests the resources for a specific time interval. Recurrent auctions[1] are gaining importance [42, 56, 41] since there are many applications where this recurrence appears, such as e-service oriented marketplaces.

Robustness for recurrent auctions acquires a different meaning than in single auctions. Especially in the case of public uncontrolled resources that can be used by the agents without authorization. In the domains where those kind of resources apply, the main problem is not on resources that become unavailable (because the resources are renewed at each repetition of the auction), therefore robustness would not be achieved as in single auctions by obtaining robust solutions that can be repaired if a resource fails. Instead, in these domains robustness is needed for avoiding possible conflicts in the case that the agents use the resources without authorization.

7.2 CASE EXAMPLE: THE WASTE WATER TREATMENT PLANT PROBLEM

In this section, we introduce a real world problem where robustness in recurrent auctions with uncontrolled resources takes an important role: the waste water treatment plant problem (WWTPP).

[1] Notice the difference between recurrent auctions and sequential auctions. A recurrent auction could be formed by a succession of some of the auction types described in Section 2.2.1. So a set of items can be auctioned periodically. In contrast, in a sequential auction the auctioneer does not auction all the items, only one item at a time.

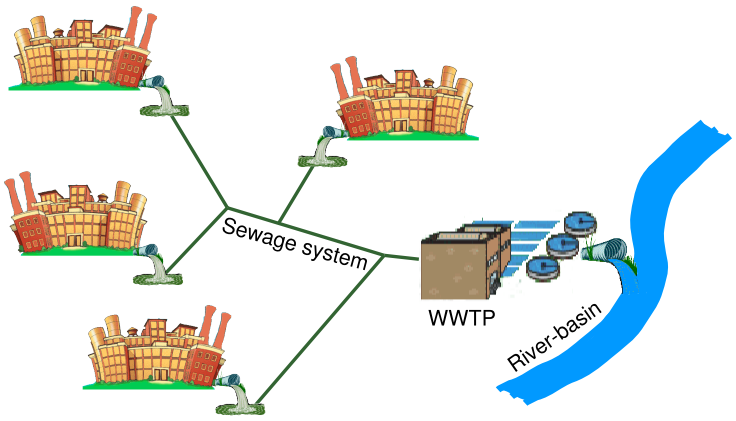

Figure 7.1: Water treatment system

The treatment of the wastewater discharged by industries into the rivers is vital for environmental quality. For this purpose, the wastewater is treated in wastewater treatment plants (WWTP). A WWTP receives the polluted wastewater discharges coming from the city and different industries. Nowadays the most common wastewater treatment is the activated sludge process. The system consists in an aeration tank in which the microorganisms responsible for treatment (i.e. removal of carbon, nitrogen and phosphorous) are kept in suspension and aerated followed by a liquid-solids separation, usually called secondary settler. Finally a recycle system is responsible for returning a fraction of solids removed from the liquid-solids separation unit back to the reactor, whereas the other fraction is wasted from the system [69].

A typical water treatment system is depicted in Figure 7.1. 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 hydraulic capacity of the plant is limited, and therefore 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. Otherwise, the wastewater could not be fully treated and the river would be polluted.

The hydraulic and contaminants capacity restrictions are defined according to its expected use (industries and cities in the surroundings that generate the waste). Currently, there exist regulations intended to achieve this goal by assigning a fixed amount of authorized discharges to each industry. However, they are not sufficient to guarantee the proper treatment of the wastewater. The problem is that, although these regulations enforce industries to respect the WWTP capacity thresholds, they do not take into account that simultaneous discharges by different industries may exceed the WWTP's thresholds. In such a case, no industry would be breaking the rules, but the effect would be to exceed the WWTP capacity.

The scheduling problem faced in this domain is to distribute the industrial discharges over time so that all the water entering the WWTP can be treated. If the discharges are done without any coordination, the amount of water arriving at the WWTP can exceed its incoming water flow threshold, and cause the overflow to go directly to the river without being treated. Moreover, if the contamination level of the water is too high, the microorganisms used in the cleaning process die, and the process has to stop until they are regenerated. Thus, in order to prevent such dangerous situations, the industrial discharges should be temporally distributed so that all of them can be fully treated.

Each industry has a tank (of a given capacity) 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 (from previous discharges).

The WWTPP 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 using the resource are the industries that perform discharges. Here 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. The goal of the auctioneer is not to exceed any of its thresholds (hydraulic capacity and contaminant levels).

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 an industry cannot use its retention tank (for maintenance purposes, for instance), or when a concrete discharge cannot be stored in the tank because of its high level of contamination, etc.

In the following sections we will study how to solve this problem with robustness, in order to avoid possible overflows caused by unauthorized discharges. The method is based in proactive robustness, where we are interested in a solution that takes into account possible changes, rather than a reactive approach where the system reacts when there is any change, finding an alternative solution. Appendix A shows how to solve optimally (without robustness) this problem using centralized approaches and comparing the performance of different modelings and solvers.

7.3 Learning Agents Behavior

In this section we describe how to add robustness by using a trust mechanism to the recurrent auction that is able to consider several possible changes on the auction. The mechanism is based on building a model of the participants in the auction that is learned in successive iterations of the recurrent auction. The robustness mechanism consists in three main components:

- **Trust model** of the agents requesting the resources
- **Risk function** of the agent selling the resources (the *auctioneer*, or *coordinator*)
- **Robust solution generation**

The first component (the trust model) is concerned with the agents requesting resources. It is a main part of the mechanism as it models the behavior of the agents by learning from their actions their behavior and the circumstances in which an agent is most likely to disobey the decisions of the coordinator and use the resource without authorization. The second component is related to the coordinator and its risk function, as the concept of a robust solution varies depending on the risk

attitude of this concrete agent. Finally, with the inputs coming from all the agents, the robustness of the system is achieved by combining the risk of the coordinator with the trust on the agents requesting the resources to generate a solution that is robust, that is, it is able to absorb (up to some level) the changes in the environment.

7.3.1 TRUST MODEL

An agent requesting resources to perform 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 needs to perform for some reason. Therefore the trust model should 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. Possible task features to build the trust model include the resources capacity requirements, the task duration, etc.

The trust model is learned during the recurrent auction by storing two main characteristics. First, the probability of disobeying of the agents, which happens when an agent uses the resource when it is not authorized to. Second, its lying magnitude, representing the difference between the requested capacity of the resources and the real used capacity, given that in some scenarios an agent may request to perform some tasks using a given capacity of resources and later use a higher capacity than requested. Consequently, the measures stored by the trust model are the following:

- **Probability of disobeying**. This value $P \in [0..1]$ could be measured in many different ways, being the most obvious the average of disobediences in relation to the total number of auctions the agent has been involved in. However, it could be measured 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 $M \in [0..\infty]$ 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 150% of the requested capacity.

A graphical representation of this trust model using only one characteristic of the task is shown in Figure 7.2 (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, but it uses a higher amount of capacity than requested (having a lie magnitude greater than 0 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 checking firstly if the task has been performed after the authorization of the auctioneer or not, that is, the agent has disobeyed the result of the coordination (the solution of the auction), and secondly if the resource capacity used is the same as what was requested.

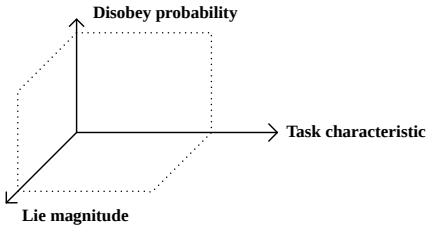

Figure 7.2: Trust model.

We use statistical procedures to learn the trust model, however, other more complex learning techniques could be used as well to fill the model such as neural networks, bayesian networks, etc.

7.3.2 RISK FUNCTION

The risk attitude of the auctioneer characterizes the tradeoff between robustness and optimality that he wants, given that robustness and optimality are contradictory objectives. The risk function of the coordinator can be also seen as his willingness to face dangerous situations.

Risk attitudes are generally categorized in three distinct classes: risk averse, neutral and proclive. 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 situations with a low probability of success. For example, a risk-averse auctioneer would consider that every request with a probability of disobeying greater than 0 is going to use the resources even if unauthorized, and thus it would auction only the remaining resources capacities over the rest of the requests. On the other hand a risk-proclive auctioneer would consider that if a request 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.

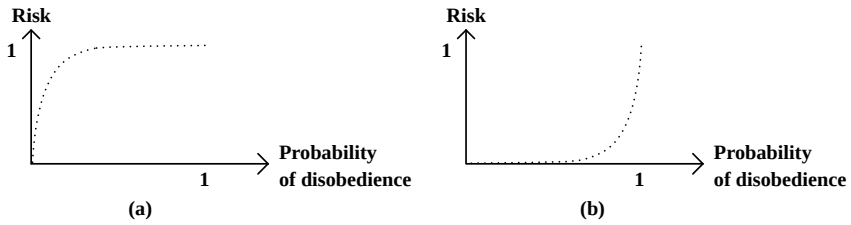

Figure 7.3: Risk attitude function: (a) averse, (b) proclive.

The risk function f_{risk} defines the risk attitude of the auctioneer (between 0 and 1) as a function of the probability of disobeying of a given agent and a given request. An example of a risk function is shown in Figure 7.3(a). In this case it represents a risk-averse auctioneer, since the resulting *risk value* is almost always 1 (it considers risky requests as if they are going to surely use the resources 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 7.3(b), and a risk-neutral one would have it set accordingly to the probability of disobeying.

7.3.3 ROBUST SOLUTION GENERATION

The trust model and the risk function of the coordinator are used to generate the *robustness constraint* that will provide robust solutions. This constraint is added to the constraint optimization problem related to the auction, in order to force the solution to be robust.

In the auction (executed each time a conflict is detected) the auctioneer is faced with a set of requests (the tasks involved in the conflict), each with trust features associated obtained from the trust model. Then the auctioneer decides which requests to authorize depending on its risk attitude.

The robustness constraint is formulated in a way that the solution finds a balance between the amount of resources required by the authorized requests and the assumed risk from the unauthorized requests (appropriately weighted by its probability of disobeying, lie magnitude and the risk function f_{risk} of the auctioneer). The objective is not to exceed the maximum capacities of the resources (Q_j). This constraint is defined as shown in Equation 7.1, where variables x_i represent whether a discharge is authorized or not, $c_{i,j}$ is the capacity required by the discharge i, and Q_j are the limits of the contaminants C.

$$\sum_{i\in[1,n]} x_i{\cdot}c_{i,j} + \sum_{i\in[1,n]} (1-x_i){\cdot}c_{i,j}{\cdot}f_{risk}(P_i){\cdot}M_i \leq Q_j \quad \forall j \in C \tag{7.1}$$

The first summatory represents the resources used by the authorized requests ($x_i = 1$), while the second summatory characterizes the resources potentially used by the unauthorized requests ($x_i = 0$). Hence, the unauthorized requests are considered as if they were performed in the cases where the probability of disobeying of the associated agent (P_i) is higher than zero. However this value (appropriately weighted with its corresponding lie magnitude M_i) is considered as a function of the risk attitude of the auctioneer f_{risk}[2]

Another way of understanding this equation is by moving the second summatory to the right side, as shown in Equation 7.2. Then it can be read as if a concrete capacity of the resources is reserved to be used by the unauthorized tasks that are likely to be disobeyed and performed anyway, which is similar to slack-based techniques [18].

$$\underbrace{\sum_{i\in[1,n]} x_i{\cdot}c_{i,j}}_{Authorized} \leq Q_j - \underbrace{\sum_{i\in[1,n]} (1-x_i){\cdot}c_{i,j}{\cdot}f_{risk}(P_i){\cdot}M_i}_{ReservedCapacity} \tag{7.2}$$

7.3.4 EXPERIMENTATION

To evaluate the robustness mechanism we have implemented a prototype of the system reproducing the coordination and communication process between plant and industries. So far we have only considered the hydraulic capacity of the plant. Industry agents calculate their bids taking into account the urgency to perform a discharge, based on the percentage of occupation of the tank. In case an

[2] 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.

industry agent is denied to perform one of its discharges, it first tries to store the rejected discharge into the tank, scheduling the discharge of the tank as its first activity after the current conflict finishes. If the industry has its tank already full, the discharge is performed anyway.

The free linear programming kit GLPK (GNU Linear Programming Kit) has been used to solve the winner determination problem related to each (multi-unit) auction, modeling it as a mixed integer programming problem. The robustness constraint is added 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 the total number of lies and truths (that is, disobedient and obedient actions), together with a value to compute the lie magnitude. These values are updated after each performed discharge in the following way: if the industry was authorized then the number of truths of the corresponding flow is incremented; otherwise the number of lies is incremented. The lie magnitude is independently computed as the difference between the used capacity and the requested capacity.

Results have been evaluated considering some quality measures based on different characteristics of the solution:

- **number of overflows (NO)** occurred during the simulation
- **maximum flow overflowed (MFO)**, measured in m^3/day
- **total volume overflowed (VO)**, in liters
- percentage of discharge denials **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) of the University of Girona. 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 and situations.

- In the first scenario there is no coordination among the industries (without coordination the industries perform its initial discharges plans, and the treatment plant does never unauthorise any discharge).
- The second scenario uses the recurrent auction to coordinate the discharges of the industries and assumes that they always obey the decisions of the plant, as long as they have enough tank capacity.
- In the third scenario 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,

			NO	MFO	VO	%IO
No coordination			80	9826	$15.21 \cdot 10^6$	-
Obey			28	4996	$3.74 \cdot 10^6$	98.95
Low Disob.	No Robustness		77.60 (4.12)	14432 (865.93)	$11.5 \cdot 10^6$ (216866)	98.55 (0.12)
	Robust.	Averse	78.70 (7.15)	14360 (1522)	$11.3 \cdot 10^6$ (261362)	98.27 (1.57)
		Neutral	79 (7.83)	13531 (1396)	$11.4 \cdot 10^6$ (260669)	98.19 (0.24)
		Proclive	84.1 (5.16)	14052 (1006)	$11.3 \cdot 10^6$ (251712)	98.15 (0.17)
Medium Disob.	No Robustness		126.60 (6.13)	14398 (1604)	$13.3 \cdot 10^6$ (363484)	96.48 (0.31)
	Robust.	Averse	126.60 (6.13)	14398 (1604)	$13.3 \cdot 10^6$ (363484)	96.48 (0.31)
		Neutral	122.9 (6.84)	13966 (803)	$13.2 \cdot 10^6$ (403934)	96.61 (0.32)
		Proclive	121.3 (7.94)	14233 (1358)	$13.2 \cdot 10^6$ (374673)	96.58 (0.41)

TEXTILE INDUSTRY ALWAYS DISOBEYING						
No coordination			80	9826	$15.21 \cdot 10^6$	-
Obey	No Robustness		112	6523	$6.89 \cdot 10^6$	90.84
	Robustness		58	6590	$5.47 \cdot 10^6$	96.77
Low Disob.	No Robustness		112 (6.09)	14955 (1201.58)	$12.6 \cdot 10^6$ (233076)	90.98 (0.2)
	Robust.	Averse	77.70 (3.68)	14225 (1212)	$11.8 \cdot 10^6$ (205150)	96.69 (1.57)
		Neutral	82.5 (7.66)	15110 (997)	$11.9 \cdot 10^6$ (199074)	96.66 (0.16)
		Proclive	81.2 (4.44)	14018 (1596)	$11.8 \cdot 10^6$ (133988)	96.68 (0.18)
Medium Disob.	No Robustness		119.70 (4.72)	14819 (1373.74)	$14.3 \cdot 10^6$ (263955)	89.96 (0.28)
	Robust.	Averse	109.50 (3.95)	14150 (1310)	$13.6 \cdot 10^6$ (242619)	95.19 (0.17)
		Neutral	113.5 (5.5)	13708 (1040)	$13.6 \cdot 10^6$ (445501)	95.16 (0.37)
		Proclive	110.9 (8.16)	14522 (1571)	$13.6 \cdot 10^6$ (338985)	95.31 (0.29)

Table 7.1: Simulation results.

the higher the chances of disobeying); a graphical representation of this function is shown in Figure 7.4. Two variations of the disobeying probability of disobeying have been tested, the

first one is exactly the same as the one shown in the figure (Low Disobedience), and the second starts at a probability of 0.1, instead of 0 (Medium Disobedience).

- Additionally, we have tested the above scenarios setting one single industry (the textile, chosen randomly) always disobeying the decisions of the plant if any of its discharges is unauthorized.

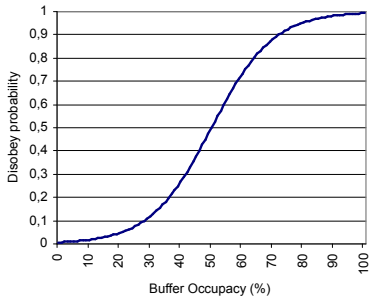

Figure 7.4: Disobey probability function.

Every scenario has been tested activating and deactivating the robustness mechanism and with different risk attitudes of the coordinator (averse, neutral and proclive).

The outcome of all the scenarios is shown in Table 7.1, with the average and deviation (in brackets) of 10 simulations performed for each scenario. Concretely, we can notice that the non-coordinating scenario produces the worst results regarding volume overflowed (which is the most important indicator), while the auction-based system improves the results, principally when all the industries obey (this reflects the best possible circumstances). With disobeying industries we can notice a subtle improvement when using the robustness mechanism in both the volume and maximum flow overflowed yet the difference is not much relevant, and the number of overflows is generally higher. Regarding the risk attitude of the coordinator we do not observe its effects in this scenario.

In the environment where there is one industry always disobeying, the robustness mechanism seems to mark differences given that all the indicators are significantly improved, specially regarding the volume overflowed and percentage of obedience. However, in this scenario, as in the previous, the different risk attitudes do not produce clear differences in the outcome.

7.4 SUMMARY

In this chapter we have analyzed robustness for recurrent auctions. In recurrent auctions the problem of resources that become unavailable is converted to agents that use the resources without authorization. Robustness in this setting means to find a solution that when some agents use the resources without authorization the solution is still valid and the resources are not overused. We have proposed an approach to achieve that which uses a trust model that learns the behavior of the agents in order to know the situations in which they are most likely to use the resource even if unauthorized.

We have presented the Wastewater Treatment Plant Problem (WWTPP), a real world problem that fits perfectly in this setting, as the resource (water flow) can be overflown in the case that the industries perform unauthorized discharges into the river. Our approach has been tested in this scenario obtaining successful results.

The method effectively learns the behavior of the agents that perform more unauthorized discharges

and decides the solution in each one of the recurrent auctions taking that into account, achieving much less overflows than the classical approach (without any robustness mechanism).

CHAPTER 8
Conclusions and Future Work

In this final chapter we first make a brief summary of the thesis. Then, we review the objectives initially set to this thesis and compare them with the final work and contributions achieved. Finally, we propose a set of topics that could be extended in future works.

8.1 SUMMARY

Robustness is a key issue when dealing with real world applications, where uncertainty is almost always present. In this thesis we have analyzed robustness for resource allocation problems modeled as auctions, where it has been rarely taken into account. In particular, we have focused on the possibility of some resources becoming unavailable once the auction has already been cleared, and we have provided a mechanism to proactively look for solutions that can be easily repaired when such unexpected events happen.

A sensitivity analysis has been performed to see that resources becoming unavailable can produce big loses in the benefit of the auctioneer. This fact is a strong motivation for our research in robustness.

We have presented a notion of robustness that balances the number of allowed repairs when a break occurs and the loss of revenue for the auctioneer, by defining (a, b, β)-super solutions. This approach allows the auctioneer to choose the more convenient values for each parameter, depending on how conservative or risk seeking his strategy is. We have shown that finding an (a, b, β)-super solution for an auction can be reduced to modeling an auction as a weighted Max-SAT formula and the looking for a supermodel of this formula. This results into the new problem of robust weighted Max-SAT. We have faced these problems following the same approach as supermodels for SAT. However, since SAT does not allow to easily encode formulas with arithmetic operations, needed to achieve robustness, we have moved the problem to the richer logical framework of Satisfiability Modulo Theories (SMT).

We have analyzed the relationship between robustness and optimality and proposed a mechanism to define the trade-off between them with a parameter α. In the experimentation section we have analyzed the results for different values of α and the conclusion is that the loss of optimality when searching robust solutions is lower than the loss of robustness when looking for optimal solutions. This is a result that should be taken into account when deciding the most appropriate value of α in real world problems.

Some experimental results have been performed, showing the feasibility of our approach with different frameworks and solvers, namely, SMT, pseudo-Boolean and Linear Programming. The results obtained are quite successful, especially if we consider them in relation with other works on ro-

bustness. As far as we know, there are very few results on reformulation approaches, and they are restricted to (1,0)- and (1,1)-supermodels. Regarding results on search-based approaches, they are also restricted to find at most (1,3)-super solutions to CSP problems. Although extensive experimentation with harder instances and other kind of problems than that of auctions, for instance the job-shop problem, should be done in order to better assess the scalability of our approach, we think that achieving a reasonable performance for finding up to (3,8)-supermodels as we do is a very successful result.

8.2 CONTRIBUTIONS

The definition and modeling of robustness for resource unavailability with repairable solutions in resource allocation problems (encoded as combinatorial auctions) is the main topic of this thesis. However, in the introduction we established a set of objectives to this work. Now we analyze these objectives and discuss the work developed for each of them, and the conclusions that can be derived.

The first objective was the quantification of the negative effects that resource unavailability produces in combinatorial auctions. Resources that become unavailable once a solution has been found produce losses in the revenue for the auctioneer. We have performed an extensive **sensitivity analysis** with several combinatorial auction distributions generated with CATS, and studied the loss in revenue in the optimal solution when some resources fail. The conclusion is that optimal solutions are usually sensitive to changes in resource availability, making them hard to be repaired. Although some distributions are more affected than others and more bids make the repair easier to restore the lost revenue, the loss in revenue in all the cases is considerable and therefore, robust solutions would be desired. Furthermore, we have analyzed the size of the repair and saw that repair sizes should be also limited as it could be arbitrarily high.

The second objective was to design a mechanism to incorporate such robustness based on repair solutions. We have defined robustness in a similar way as supermodels for SAT, super solutions and weighted super solutions for CP. Our **(a, b, β)-super solutions** are a new kind of robust solutions that actually generalize them, enabling to find the robust solutions that we are interested in. We presented a mechanism based in propositional logic to find such robust solutions, which uses reformulation to create a formula that is satisfiable if and only if the problem has a robust solution. We have made a large amount of experiments with this model on a wide variety of combinatorial distributions and several solvers, and changing the values of the parameters. The conclusion is that it is effectively applicable to many practical instances.

Actually, our mechanism for solving weighted Max-SMT robustness is generic and can be applied to any weighted Max-SAT problem, auctions being a particular case. Moreover, since our approach is based on reformulation, there is no need of developing new algorithms to solve the robust version of the problem, and so we can take advantage of the advances in SAT and SMT solvers that we think are going to improve drastically its performance in the next few years.

The third objective of this thesis was to add **flexibility** to the solutions in order to study the compromise between optimality and robustness. This flexibility is also useful for hard instances, where the previous mechanism did not provide any solution due to its strictness. We adapted the robustness mechanism by using a concept similar to soft-constraints. We also incorporated a new parameter α to easily switch between robust and optimal solutions. This flexibility allows to find solutions in

over-restricted situations, and furthermore to easily define the desired trade-off between optimality and robustness by setting the appropriate value of α. A set of experiments was performed with this parameter for analyzing the balance between optimality and robustness (fifth objective of the thesis) and the conclusion is that the loss of optimality when searching robust solutions is lower than the loss of robustness when looking for optimal solutions. This is a result that should be taken into account when deciding the most appropriate value of α in real world problems.

Incentive-compatibility issues (fourth objective of this thesis) were analyzed in the sixth chapter. We proved that although a restricted version of our problem was not incentive compatible, the same proof did not succeed for the generic case. Two approaches were developed for finding counter-examples of monotonicity, which would imply that the mechanism is not incentive-compatible. Both methods did not find any counter-example. This means that either the counter-example does not exist and therefore the method is actually incentive compatible, or that the counter-example exists but it is not simple. Both alternatives are good, since we saw in the sensitivity analysis that large auctions (where the counter-example could appear) tend to be already inherently robust.

8.3 Future Work

Although the initial objectives of this thesis have been accomplished, during this research we have realized that there is still more work to do in various directions. In this section we give an idea of the topics that could be extended.

8.3.1 Robustness Notions

The use of logic languages and techniques to deal with auction robustness gives us a high level of expressiveness, which enables us to deal with distinct robustness variants. For instance, in the auctions setting we have considered, we have focussed on robustness with respect to good unavailability. However, for having robustness with respect to bid withdrawal, we simply need to set the breakable variables to the bids. Our approach can be seen as a generic framework for robustness through reformulation, since with only slight changes in the encoding, we can achieve other notions of robustness.

Thus, an open line of research is to study these notions and their encodings to different frameworks. For example, a robustness variant could be to directly designate the potential breaks to handle: instead of using all the variables, we could decide what (combinations of) breaks deserve being repaired. This would be useful if we only want to consider those breaks having a non negligible probability of occurring. We could also think of a robustness notion where each breakable variable has a corresponding set of associated repairable variables. This could serve, for instance, in the presence of scheduling, where one should only look for repairs on the forthcoming assigned resources, or in an auction scenario where it is not permitted to make repairs by switching a winning bid to a loser one. We could study also how to define the notion of robustness for multi-unit auctions, or in scenarios where failure probabilities are taken into account, among others. Again, we envisage that thanks to the high expressiveness of our modeling, these features should be feasible.

8.3.2 Quantified COP

Quantified COP (QCOP) is a generalization of COP where the variables may be universally quantified over their domains [3]. This framework seems suitable for finding robust resource allocations. Using QCOP might lead us to a more compact reformulation of the problem. This approach might be particularly useful when dealing with cumulative resources (i.e. resources that can be simultaneously used by several agents).

However, the expressive power of QCOP comes at a cost. While COP is solved by just assigning values for its (existentially quantified) variables such that all the constraints are satisfied, a QCOP is solved by exhibiting *winning strategies*. Therefore, it would be interesting to elaborate an study of the suitability of QCOP for robustness.

8.3.3 Scalability

The size complexity of the proposed mechanism for robustness based on reformulation is exponential in a even when using cardinality constraints. Therefore, it may be worth performing a more extensive analysis on the scalability of our approach, and the growth in problem size with the number of items and bids.

8.3.4 Search Algorithms

We have proposed a method based on reformulation to find robust solutions. The model can be generated in different encodings so that it can be then solved using various solvers, such as SMT, pseudo-Boolean and Integer Linear Programming. We have seen that this approach obtains good results.

It is left as future work to look for specialized search algorithms or operational research techniques for solving the robust weighted Max-SAT problem and make corresponding performance comparisons. This would be specially important for those cases with a large number of bids and/or higher values of a, where our approach could be inefficient.

8.3.5 Incentive-Compatibility

In the chapter dedicated to incentive compatibility, we saw that although the robustness mechanism was not incentive compatible in a restricted case (because a counter-example of monotonicity was found), no proof of non-monotonicity was found in the generic case. Two approaches for finding such counter-example were proposed, but they did not find any counter-example. This points to the possibility that the mechanism is indeed incentive-compatible, however, a formal proof should be given to ensure that.

After that, the remaining necessary conditions (exactness, participation and critical) for proving the truthfulness of the mechanism could be studied in order to prove the truthfulness of the mechanism.

8.3.6 Recurrent Auctions

For recurrent auctions, we proposed an approach of achieving robust solutions based on a trust model that learns the behavior of the agents. However, other approaches could be used. Concretely, the robustness mechanism proposed for single shot auctions could be extended in order to deal more naturally with recurrent auctions.

Alternatively, the parameters learnt by the trust model could be used in order to set the probability of failure (breakage) in the flexible robustness approach. This would be a good work to tie up all loose ends.

Appendices

APPENDIX A
Benchmarks with the WWTP Problem

In this chapter we deal with the Waste Water Treatment Plant Problem (WWTPP) introduced in Chapter 7. We define it formally, and use it to test the efficiency of the different tools and solvers available with it. Although we do not handle robustness issues in this case, we think that the comparison of the different solvers is interesting, since all of them could be used to solve the robustness problem presented throughout the thesis.

A.1 THE WWTPP PROBLEM

The Waste Water Treatment Plant Problem (WWTPP) was introduced in Section 7.2. We focus on the decision variant of the problem, i.e., in finding a feasible solution not exceeding an overall deadline, instead of in minimizing the makespan. Actually, in the real case, it is sufficient that all discharges are rescheduled within the same day for which they were originally scheduled (and, in fact, the minimization of the makespan could be not good for the WWTP, as it is preferably, for the microorganisms' functioning, that the discharges are homogeneously distributed throughout time). We will address this problem taking into account only the water flow, therefore we assume having a single resource of given capacity.

The problem can be roughly defined as: given a list of all the discharges to be performed (each one with a given duration, release time, deadline and resource capacity requirement), we are asked to find a start time of each discharge between its release time and its deadline, such that, at any time, the sum of resource requirements of the discharges scheduled at that time does not exceed the WWTP capacity. There is some precedence relation (presumably, a chain) between the tasks of each single industry. Since the delays introduced in the discharges (in order to find a feasible schedule) should not stop or delay the production processes of the industries, the idea is to keep those discharges temporarily in a retention tank in the industry itself, and to discharge them to the river later on, possibly in disjoint intervals, because a discharge coming from a tank can be interrupted.

In summary, an instance of the Wastewater Treatment Plant Problem (WWTPP) is given by:

- a single resource of given capacity,
- a set of tasks, each one with a given duration, release time and resource capacity requirement,
- a chain-like precedence relation between the tasks,

- for any such chain of tasks, a buffer (or retention tank) of given capacity and output rate (we assume that the input rate is flexible) and
- an overall deadline (greater than all release times).

The question is to find a schedule where:

- each task is either scheduled at its release time (and does not exceed the deadline), or else it is redirected to its corresponding buffer with a volume equal to its resource capacity requirement multiplied by its duration,
- the capacity of each buffer is not exceeded at any time,
- each buffer is emptied, preemptively, at its corresponding rate,
- each buffer is empty at the deadline, and
- at any time, the sum of required capacities of the tasks scheduled at that time, together with the required capacities of the emptying of the buffers at that time, does not exceed the capacity of the single resource.

Notice that nothing prevents a buffer from being emptied and filled at the same time, and also from being emptied at the same time at which one of the tasks is scheduled.

A.2 Modeling the WWTPP with SMT

In this section we give an encoding of a WWTPP instance into a SAT modulo unquantified Linear Integer Arithmetic (LIA) instance. As we will see, SAT modulo LIA nicely captures all constraints. Afterwards we translate this encoding into an Integer Programming problem, with the aim of comparing the performance of state-of-the-art solvers on both approaches.

A WWTPP instance can be easily encoded as a SAT modulo unquantified Linear Integer Arithmetic instance as follows.

A.2.1 Constants

We have the following non-negative integer constants:

- $PlantCapacity$ denotes the capacity of the wastewater treatment plant at each time period.
- Given a set of k industries, $TankCapacity_i$ and $TankFlow_i$ denote respectively the capacity and the emptying rate of the buffer associated to industry i, $\forall i \in 1 \dots k$.
- Given a set of discharges from k industries to be scheduled within m time periods, d_{ij} denotes the scheduled flow of discharge for industry i during time period j, $\forall i \in 1 \dots k, j \in 1 \dots m$.

A.2.2 VARIABLES

Given a set of discharges from k industries to be scheduled within m time periods, we have the following integer variables $\forall i \in 1 \dots k, j \in 1 \dots m$:

- For every $d_{ij} > 0$, c_{ij} denotes the actual "capacity requirement" of industry i during time period j, corresponding to a scheduled discharge. That is, for every $d_{ij} > 0$, either $c_{ij} = d_{ij}$, or $c_{ij} = 0$ and the discharge is redirected to that industry's buffer.
- $Bout_{ij}$ denotes the flow discharged from buffer (of industry) i during time period j.
- Buf_{ij} denotes the flow stored in buffer i at the end of time period j.

A.2.3 CONSTRAINTS

We next define the set of constraints. The explanation of each constraint is given at the end.

$$\forall j \in 1 \dots m : \sum_{i=1}^{k} c_{ij} + Bout_{ij} \leq PlantCapacity \tag{A.1}$$

$$\forall i \in 1 \dots k : Buf_{i1} = d_{i1} - c_{i1} \tag{A.2}$$
$$\forall i \in 1 \dots k, j \in 2 \dots m : Buf_{ij} = Buf_{ij-1} - Bout_{ij} + d_{ij} - c_{ij} \tag{A.3}$$
$$\forall i \in 1 \dots k, j \in 2 \dots m-1 : Buf_{ij} \leq TankCapacity_i \tag{A.4}$$
$$\forall i \in 1 \dots k : Buf_{im} = 0 \tag{A.5}$$

In constraints A.2 and A.3, the difference $d_{ij} - c_{ij}$ is replaced by 0 if $d_{ij} = 0$ (recall that variables c_{ij} have been defined only for corresponding constants $d_{ij} > 0$).

$$\forall i \in 1 \dots k : Bout_{i1} = 0 \tag{A.6}$$
$$\forall i \in 1 \dots k, j \in 2 \dots m : Bout_{ij} = 0 \tag{A.7}$$
$$\vee \left(Bout_{ij} = TankFlow_i \wedge Buf_{ij-1} \geq TankFlow_i\right) \tag{A.8}$$
$$\vee \left(Bout_{ij} = Buf_{ij-1} \wedge Buf_{ij-1} \leq TankFlow_i\right) \tag{A.9}$$

For every discharge from an industry i, spanning from time period a to time period b, we state:

$$(c_{ia} = 0 \wedge \dots \wedge c_{ib} = 0) \vee (c_{ia} = d_{ia} \wedge \dots \wedge c_{ib} = d_{ib})^1 \tag{A.10}$$

Finally, the following (obvious) redundant constraints can be added in order to help orienting the search:

[1] Notice that $d_{ia} = \dots = d_{ib} > 0$.

$$\forall i \in 1 \dots k, j \in 2 \dots m : 0 \leq Bout_{ij} \leq TankFlow_i \tag{A.11}$$

$$\forall i \in 1 \dots k, j \in 2 \dots m : Bout_{ij} \leq Buf_{ij-1} \tag{A.12}$$

Constraints A.1 state that the capacity of the WWTP is not exceeded at any time. Constraints A.2 and A.3 define the amount of water inside every buffer at every time interval, taking into account the amount of water inside each buffer at the previous time interval, and the current output and input flows for this buffer. Constraints A.4 require the capacity of each buffer not being exceeded at any time, and constraints A.5 impose all buffers being empty at the deadline. Constraints from A.6 to A.9 are restrictions on the output flow from the buffers (or retention tanks): the output flow at the first time interval must be zero (as the buffer is empty) and, at subsequent time intervals, it can be either zero, or it can be equal to the tank flow (provided that there is enough water inside the buffer) or it can be equal to the remaining water inside the buffer if this is less or equal than the tank flow. Constraints A.10 express the dichotomy of throwing each discharge to the river or redirecting it to a buffer.

Constraints A.11 and A.12 are unnecessary, but have proved to be helpful in our experiments. Notice that, although the value of the $Bout$ variables is perfectly defined by constraints A.6 to A.9, restricting the domain of the $Bout$ variables can help in the search for solutions.

A.3 IP MODELING

In order to obtain an IP instance from the previous SMT instance, we need to convert logical combinations of linear constraints into conjunctions of linear constraints. We use standard transformations like the ones of [74].

We define, $\forall i \in 1 \dots k, j \in 1 \dots m$, binary variables r_{ij} denoting whether discharge from industry i at time period j is actually scheduled or else redirected to a buffer. Then we replace c_{ij} with $r_{ij} \cdot d_{ij}$ inside constraints A.1, A.2 and A.3. Constraints A.4, A.5 and A.6 remain the same. The binary variables r_{ij} allow constraint A.10 to be translated into

$$r_{ia} + \dots + r_{ib} = 0 \vee r_{ia} + \dots + r_{ib} = b - a + 1.$$

This can then be encoded as a conjunction of linear constraints by defining additional binary variables δ_{iab} for every discharge from an industry i spanning from time period a to time period b, and stating:

$$r_{ia} + \dots + r_{ib} + (b - a + 1) \cdot \delta_{iab} \leq b - a + 1 \tag{A.13}$$

$$-(r_{ia} + \dots + r_{ib}) - (b - a + 1) \cdot \delta_{iab} \leq -(b - a + 1) \tag{A.14}$$

The disjunction of constraints A.7, A.8 and A.9 can be expressed as

$$\delta'_{1ij} \rightarrow Bout_{ij} = 0 \tag{A.15}$$

$$\delta'_{2ij} \rightarrow Bout_{ij} = TankFlow_i \wedge Buf_{ij-1} \geq TankFlow_i \tag{A.16}$$

$$\delta'_{3ij} \rightarrow Bout_{ij} = Buf_{ij-1} \wedge Buf_{ij-1} \leq TankFlow_i \tag{A.17}$$

where δ'_{1ij}, δ'_{2ij} and δ'_{3ij} are again binary variables, and

$$\delta'_{1ij} + \delta'_{2ij} + \delta'_{3ij} \geq 1 \tag{A.18}$$

Then constraints A.15, A.16 and A.17 can be transformed into a conjunction of linear constraints by using $Big - M$ like constraints[2]. In this way, constraint A.15 becomes

$$Bout_{ij} + TankFlow_i \cdot \delta'_{1ij} \leq TankFlow_i \tag{A.19}$$

and constraint A.16 becomes

$$TankFlow_i \cdot \delta'_{2ij} - Bout_{ij} \leq 0 \tag{A.20}$$

$$-Buf_{ij-1} + TankFlow_i \cdot \delta'_{2ij} \leq 0 \tag{A.21}$$

Notice that these constraints work in conjunction with constraints A.11, which are mandatory here: on the one hand, from A.19 we get $Bout_{ij} \leq 0$ whenever $\delta'_{1ij} = 1$, which together with $0 \leq Bout_{ij}$ (from A.11) gives us $Bout_{ij} = 0$ as we need; on the other hand, from A.20 we get $TankFlow_i \leq Bout_{ij}$ whenever $\delta'_{2ij} = 1$, which together with $Bout_{ij} \leq TankFlow_i$ (from A.11) gives us $Bout_{ij} = TankFlow_i$ as we need.

Finally, constraint A.17 becomes

$$Buf_{ij-1} - Bout_{ij} + TankCapacity_i \cdot \delta'_{3ij} \leq TankCapacity_i \tag{A.22}$$

$$Buf_{ij-1} + TankCapacity_i \cdot \delta'_{3ij} \leq TankCapacity_i + TankFlow_i \tag{A.23}$$

Constraints A.12 are mandatory for similar reasons as before, since they work in conjunction with A.22.

A.4 BENCHMARKING

Here we comment on some benchmarking we have performed, showing that state-of-the-art SMT solvers outperform best IP solvers with the previous modeling of the WWTPP. We worked with two sets of benchmarks, one coming from real data and another coming from randomly generated data[3].

In the real set of benchmarks we used data coming from 8 industries (each one having its own retention tank), with a total of 94 discharges planned within a period of 24 hours. We took a time discretization of one hour and an overall deadline of 24 hours for the schedule. Different problem instances were generated with different capacities of the wastewater treatment plant, ranging from 2000 units to 10000, at increments of 20. In this way, an easy-hard-easy transition was observed (as already noted by [11, 33] for similar scheduling problems) with a transition from unsatisfiability to satisfiability taking place at 5000 units of capacity.

For the random set of benchmarks we considered a total of 114 discharges from 10 industries (having again each one an associated retention tank), all of them being planned within a period of 24 hours. Although randomly generated, both the magnitude and duration of the discharges and the size of the retention tanks was restricted to be within reasonable limits. We took a time discretization of one hour and an overall deadline of 26 hours for the schedule. From this data different problem instances were generated, with a capacity of the wastewater treatment plant ranging from 5000 to 30000 units, at increments of 100, resulting into a transition from unsatisfiability to satisfiability at 14500 units.

All the benchmarks, written according to the modeling of section A.2 in the SMT-LIB standard

[2]The idea of $Big - M$ constraints is the following: a disjunction like, e.g., $(x \leq 0) \vee b$, where b is a propositional variable, can be converted into $x \leq ubound(x)b$, where $ubound(x)$ denotes an upper bound of x.

[3]The data used in both sets of benchmarks can be found in `http://ima.udg.edu/~mbofill/wwtpp.tar`

Table A.1: SMT vs. IP

	Real set		Random set	
Solver	% Solved	Time	% Solved	Time
Yices	**100.0**	**1227.4**	**100.0**	**5.2**
Z3	99.8	1152.7	100.0	285.2
CPLEX[a]	97.6	1855.5	98.8	594.8
CPLEX[b]	93.5	1811.5	92.9	25.0

[a] Minimizing sum of buffer contents.
[b] Without objective function.

language, were submitted in 2009 to the SMT library[4], and some of them were chosen for the annual SMT competition[5] in the corresponding category.

Table A.1 shows the percentage of solved benchmarks and the total time spent by IBM ILOG CPLEX 11, Z3.2$^{\alpha}$ (SMT-COMP'08 QF_LIA division winner) and Yices 2 (SMT-COMP'09 QF_LIA division winner), with a time out of 1800 seconds for each instance in the real set, and of 300 seconds in the random set. All benchmarks were executed on a 3.80 GHz Intel Xeon machine with 3.5 GB of RAM running under GNU/Linux 2.6. The modeling given in section A.3 was used for CPLEX.

As it can be seen, state-of-the-art SMT solvers clearly outperform CPLEX on this benchmarks. It is specially remarkable that Yices solves all the benchmarks, and Z3 only fails in solving one from the real set around the phase transition. Moreover, Yices is able to solve all the 251 benchmarks from the random set in only 5.15 seconds, being almost insensitive to the phase transition. With respect to CPLEX, although it has very good performance in many instances, it fails to solve some of them around the phase transition. Since SMT solvers, as it does CPLEX, use a simplex procedure for handling atomic linear constraints, other elements of SMT technology such as conflict-driven lemma learning, backjumping or restarts can be playing a central role in this problem.

It is worth noting that worse results are obtained by CPLEX if no objective function is used. After trying with several objective functions, we obtained the best results by minimizing the sum of buffer contents. This somehow corresponds to an eager strategy consisting in avoiding the use of buffers if possible (and hence prioritizing discharges of wastewater at their preliminarily scheduled times) and emptying the buffers as soon as possible. Notice however that no objective function or user-given search strategy is possible with SMT solvers, which are completely black-box for the user and, still, better results are obtained.

A.5 Comparison with Constraint Programming

For the sake of completeness, in this section we detail the results obtained with several Constraint Programming (CP) tools on our benchmarks.

In order to do the benchmarking, our modeling needs to be translated into several CP dialects. For the comparison to be fair, in all cases we must choose an encoding as similar as possible to the one described in section A.2. This implies avoiding the use of global constraints and sophisticated search

[4] `http://www.smt-lib.org`
[5] `http://www.smt-comp.org`

Table A.2: SMT vs. CP

	Real set		Random set	
Solver	% Solved	Time	% Solved	Time
SICStus[a]	68.8	258.9	81.7	27.7
Comet[b]	76.3	744.5	53.8	196.0
Comet[c]	46.4	43.8	71.7	27.1
Tailor + Minion	81.3	547.6	44.6	98.3
mzn2fzn + G12	28.9	32.2	74.9	77.1
mzn2fzn + Gecode	0.0	0.0	37.1	9.8
mzn2fzn + ECL^{i}PS^e	0.0	0.0	0.0	0.0
mzn2fzn + SICStus	0.0	0.0	37.1	345.8
mzn2fzn + fzn2smt + Z3	99.8	4735.8	100.0	159.0
mzn2fzn + fzn2smt + Yices	**99.8**	**702.8**	**100.0**	**40.5**

[a] With labeling options: max, down.
[b] Using CP engine.
[c] Using LP engine.

strategies that can be available in CP tools. For this reason, we have only used labeling strategies[6]. Results on a different encoding, using the `cumulative` global constraint, are given in the next section.

Since the translation of the encoding described in section A.2 into a CP program over finite domains is almost direct, the encodings obtained for each CP tool are very similar and hence we do not detail them here. Moreover, for solvers providing a FlatZinc front-end, we have used the same MiniZinc model: MiniZinc [53] proposes to be a standard CP modeling language that can be translated into an intermediate language called FlatZinc. FlatZinc instances can be obtained from MiniZinc instances by using the MiniZinc-to-FlatZinc translator `mzn2fzn`, and then can be plugged into any solver providing an specialized front-end for FlatZinc.

Table A.2 shows the results obtained by several CP solvers on the benchmarks described in Section A.4, except for the last two entries, which show the results obtained by the same SMT solvers used in Section A.4, but where SMT instances have been obtained from FlatZinc instances through an experimental compiler `fzn2smt`[7]. The table refers only to the solving time (we do not include translation times since we are interested in comparing solving times, regardless of the input language). All benchmarks were executed on a 3 GHz Intel Core 2 Duo machine with 1 GB of RAM running under GNU/Linux 2.6.

At a first glance we can observe that SMT solvers are far better than other tools on these benchmarks. It is remarkable that, after the two step translation from MiniZinc-to-FlatZinc-to-SMT, we obtain similar (and in some case even better) results to the ones in Section A.4.

We tried different labeling strategies with CP solvers, but almost identical results were obtained. Hence, unless contrarily indicated, the results in Table A.2 are for the default strategy, which is

[6] Notice that there is always a default labeling strategy in these tools and, hence, trying with some labeling options does not imply doing any change in the encoding.
[7] Available at `http://ima.udg.edu/recerca/grupESLiP.html`

usually first-fail: selecting the leftmost variable with smallest domain next, in order to detect infeasibility early. This is often a good strategy. However, with SICStus Prolog we obtained significantly better results when using the `max` and `down` options: selecting the leftmost variable with the greatest upper bound next, and exploring its domain in descending order. In our program, this translates to a strategy consisting in giving priority to the biggest discharges, and keeping them in buffers as least as possible. Notice that this roughly coincides with the objective function giving best results in the IP approach of Section A.4.

The concrete versions of the CP solvers we used are: SICStus Prolog 4.0.1 (for the first entry in the table), SICStus Prolog 4.1.1 (with FlatZinc support, for the MiniZinc case), Comet 2.0, Minion 0.9, G12 MiniZinc 1.0.3, Gecode 3.2.2, and ECL^{i}PS^e 6.0. For the case of Minion, we used Tailor as a translator from the ESSENCE [23] high-level language to the Minion language, in the same spirit of using the Minizinc-to-Flatzinc translator `mzn2fzn`. This allowed us to use an almost identical model. Comet already supports a high-level language which allowed us to express the constraints in a very similar way. Moreover, for the case of Comet we tried both the CP engine and the LP engine, with no clear winner. We want to remark that we are aware of IBM ILOG CP Optimizer, which uses constraint programming to solve detailed scheduling problems and combinatorial problems not easily solved using mathematical programming methods. Unfortunately we were not able to test this tool on our benchmarks, since the trial version has severe limitations in the number of variables and in the number of allowed constraints.

A.6 A DIFFERENT APPROACH FOR CONSTRAINT PROGRAMMING

An alternative approach is to solve the WWTPP by exploiting the use of the `cumulative` constraint within a CP system, since this constraint is closely related to our problem. Many CP systems, such as CHIP V5, ECL^{i}PS^e, B-Prolog and SICStus Prolog, include the `cumulative` global constraint in their finite domain library. This constraint was originally introduced into the CHIP programming system to describe and solve complex scheduling problems [1].

Its habitual syntax is `cumulative(Starts,Durations,Resources,Limit)`, where `Starts`, `Durations`, and `Resources` are lists of integer domain variables or integers of the same length, and `Limit` is an integer. The declarative meaning is: if the lists denote respectively the start times, durations and resource capacity requirements of a set of tasks, then the sum of resource usage of all the tasks does not exceed `Limit` at any time. One should expect that, by using this constraint adequately, the performance of a CP system on the previous problem will be better (or, at least, not worse) than if not using it.

Our modeling using the `cumulative` constraint goes as follows. Given a discharge i of duration d_i and resource capacity requirement c_i, since it can either go directly to the river or be redirected to a retention tank of certain output rate r, we create a set of new n discharges of duration 1 and capacity requirement r, and one discharge of duration 1 and non-negative requirement capacity $r' \leq r$ (the remainder), such that $d_i c_i = rn + r'$. Observe that by dividing the discharges into a number of discharges of duration 1 we get rid of preemption. Then, by using reified constraints, we state that the capacity requirements of those $n + 1$ new discharges is actually 0 if and only if the associated original discharge i goes to the river.

Notice that a set of remainders (each of them coming from a different original discharge of the same

Table A.3: Cumulative modeling

	Real set		Random set	
Solver	% Solved	Time	% Solved	Time
SICStus[a]	76.6	3231.9	95.2	1347.2
mzn2fzn + G12	67.6	64.0	12.8	8.5
mzn2fzn + Gecode	72.6	1709.2	61.4	24.8
mzn2fzn + ECL^{i}PS^e	23.2	3255.7	17.9	637.0
mzn2fzn + SICStus	69.3	2029.8	51.4	432.4

[a] With labeling options: max, down.

industry) could eventually be redistributed, forming a new set of discharges of resource capacity r plus one single remainder. However, such redistribution should be made for the remainders being available at each time, i.e., dynamically, and this does not go in the direction of an encoding using the `cumulative` constraint, which requires a fixed set of resources. Therefore, here we do not consider the possibility of redistributing the remainders. Although this is an inexact formulation of the problem, in practice it results a very few times in a smaller set of solutions than with the encoding used in the previous sections. And, in any case, since this simplification results in a smaller search space, it is likely to favour this approach.

Then, apart from stating the obvious release time, precedence and finishing time constraints, we use the `cumulative` constraint two-fold. On the one hand, we use it in order to assure that the WWTP capacity is not exceeded. On the other hand, we use it in order to assure that the output rate and capacity of every retention tank is not exceeded. This second use implies stating two `cumulative` constraints for each industry, in the following way:

Let `[I1,...,In]` be a list with the initial times of the discharges kept in the retention tank of a industry, let `[H1,...,Hn]` be the times at which they are respectively flushed out from the tank, let `[C1,...,Cn]` be their resource capacity requirements, let `r` be the output rate of the tank, and let `c` be the capacity of the tank. Then we state

```
cumulative([H1,...,Hn],[1,...,1],[C1,...,Cn],r)
```

in order that the output rate of the tank is not exceeded, and

```
cumulative([I1,..., In],[H1-I1,...,Hn-In],[C1,...,Cn],c)
```

in order that the capacity of the tank is not exceeded.

Finally, for symmetry breaking, we state ordering constraints between (indistinguishable) discharges from each retention tank. Since all these discharges are of duration 1, this improvement dramatically reduces the search space.

Table A.3 shows the results obtained by the CP solvers supporting the `cumulative` global constraint on the same benchmarks as in the previous sections. Again, we used the possibility of sharing a unique MiniZinc model, except for the first entry, where we directly built a Prolog program. We

can observe that, in general, the results are better than with the previous encoding for the same CP solvers (with the only exception of G12 in the random set). However, these results are still far from the ones obtained by SMT solvers. This can be due to the fact that we are using two `cumulative` constraints for each industry (for assuring, respectively, that the output rate and the capacity of each retention tank is not exceeded), plus one `cumulative` global constraint (for assuring that the WWTP capacity is not exceeded) and, moreover, we are using many reified constraints (for the dichotomy of sending the discharges either to the river or to a retention tank), making thus difficult for the CP solvers to take profit of their algorithms for the `cumulative` constraint.

A.7 SUMMARY

We have presented the Wastewater Treatment Plant Problem (WWTPP), a real scheduling problem, and have compared several techniques for solving it. The encoding of the WWTPP into SAT modulo linear integer arithmetic, and using a high-performance SMT solver as a black-box for solving it, has turned out to be one of the best approaches. Specifically, we have seen that state-of-the-art SMT solvers are competitive with current best IP solvers, and even better on difficult instances of this problem (i.e., the ones around the phase transition). These results show that current SMT solvers are ready to solve real problems outside the verification area, and that they provide a nice compromise between expressivity and efficiency.

Let us recall that SMT solvers, like IP tools, use a simplex procedure for handling atomic linear constraints. However, the particular treatment of bound constraints of the form $x \leq k$ or $x \geq k$ inside a simplex procedure like the one of Yices, must be a key ingredient for the good results obtained in this problem by this solver (notice that many constraints in this problem are of this form). Also, we think that usual SMT techniques such as backjumping, restarts, and conflict-driven lemma learning must be a key ingredient for the good results obtained around the phase transition.

Moreover, in our point of view, the encoding of the WWTPP as an SMT problem is simpler than as an IP problem (where logical combinations of linear constraints must be translated into conjunctions of linear constraints, with the addition of zero-one variables). Compared to CP, the SMT approach is not that simple (since most CP tools provide a high-level language front-end), but far more efficient. The performance of SMT solvers on this problem is still more significant if we take into account that they are completely black-box, and one cannot provide neither labeling strategies nor local search algorithms for guiding the search.

APPENDIX B
Winner Determination Algorithm for Single-unit Combinatorial Auctions

In this chapter we present an algorithm for solving the winner determination problem related to single-unit combinatorial auctions. The algorithm is divided in three main phases. The first phase is a pre-processing step with some reduction techniques. The second phase calculates an upper and a lower bound based on a linear programming relaxation in order to prune the search. Finally, the third phase is a branch and bound depth first search where the linear programming relaxation is used as upper bounding and sorting strategy. Experiments against specific solvers like CASS and general purpose MIP solvers as GLPK and CPLEX show that our algorithm is in average the fastest free solver (CPLEX not included), and in some instances drastically faster than any other.

B.1 INTRODUCTION

Since 1998 there has been a surge of research on designing efficient algorithms for the WDP in combinatorial auctions (see [20, 16] for a more extended survey). Given that the problem is *NP-Hard* in the strong sense, any optimal algorithm will be slow on some problem instances. However, in practice, modern search algorithms can optimally solve the WDP in a large variety of practical cases. There exist typically two different ways of solving it. On one hand there exist specific algorithms that have been created exclusively for this purpose, such as CASS [24] and CABOB [66]. On the other hand, the WDP can be modeled as a mixed integer linear problem (MIP) and solved using a generic MIP solver. Due to the efficiency of actual MIP solvers like GLPK (free) and specially CPLEX (commercial), the research community has nowadays mostly converged towards using MIP solvers as the default approach for solving the WDP. There also exist sub-optimal algorithms for solving the winner determination problem that find quick solutions to combinatorial auctions [67, 38]. However we will focus only on optimal solutions.

An interesting thing to be noted about the modeling of the WDP as a MIP is that if bids were defined in such a way that they could be accepted partially, the problem would become a linear program (LP) which, unlike MIP, can be solved in polynomial time. We have kept this idea in mind to design a new algorithm, which combines LP, search and several reduction techniques to obtain better results than other solvers, even CPLEX in some particular instances.

B.2 NOTATION

Here we introduce a few notation that is going to be used through this chapter. In a single-unit 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.

B.3 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 divided in three main phases:

- The first phase performs a fast preprocessing (polynomial time) 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.
- The second phase consists in calculating upper and lower bounds for each bid. The upper bound of a bid is computed by formulating a relaxed linear programming problem (LP), while the lower bound is computed generating a solution quickly. 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. In this phase the two previous phases are used also as heuristic and for pruning.

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 (which happens more frequently 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.

B.3.1 First phase: Pre-processing

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

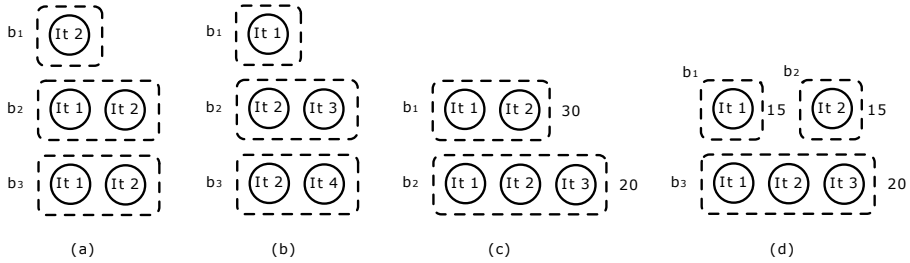

Figure B.1: Examples of (a) dominated item (it_1), (b) solution bid (b_1), (c) dominated and (d) 2-dominated bids.

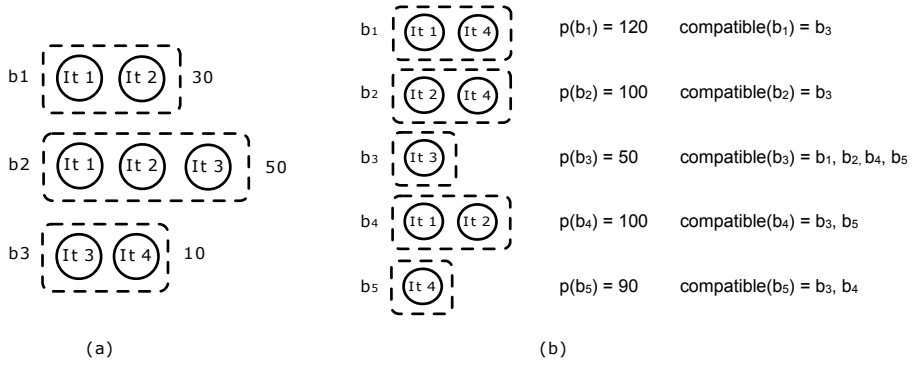

Figure B.2: Left: Example of pseudo-dominated bid (b_1 is pseudo-dominated). Right: Example of compatibility-dominated bid (b_2 is compatibility-dominated by b_1).

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: Dependent 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 *dependent*. Then, the former can be removed without any loss of information. Hence, this step deletes (leaves out of consideration) dependent 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 B.1 (a) shows an example of this situation; here item it_1 can be deleted given that 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 B.1 (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.

- **Step 4: Dominated bids.** This is the same pre-processing step that CASS [24] and CABOB

[66] 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 B.1 (c) 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 (also noticed in [65]), 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 B.1 (d) 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, for many reasonable distributions, 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 so useful for $n > 2$.

- **Step 6: Pseudo-dominated bids.** This step is an even 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_i together with its best compatible bid; therefore b_i does definitely not belong to the optimal solution and might be removed. Figure B.2 (a) 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.

- **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)[1], since they cannot improve the best solution already found.

  The upper bound u of a bid b_x is calculated according to Equation B.1 where $C'(b_x, it_k)$ is the set of compatible bids of b_x including item it_k. Roughly speaking, 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 constructing 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. 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.

$$u(b_x) = p(b_x) + \sum_{\forall i \notin g(b_x)} \max_{\forall j \in C'(b_x, it_k)} \frac{p(b_j)}{n(b_j)} \tag{B.1}$$

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

- **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 B.2 (b) 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.

B.3.2 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 [17], which solves most of the instances in polynomial-time. The relaxed version does not contains neither the current bid b_i nor 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 at first glance, 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 (partial) 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.

B.3.3 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 B.3. 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).

```
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 B.3: Pseudo-code algorithm of iCabro procedure

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.
- 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

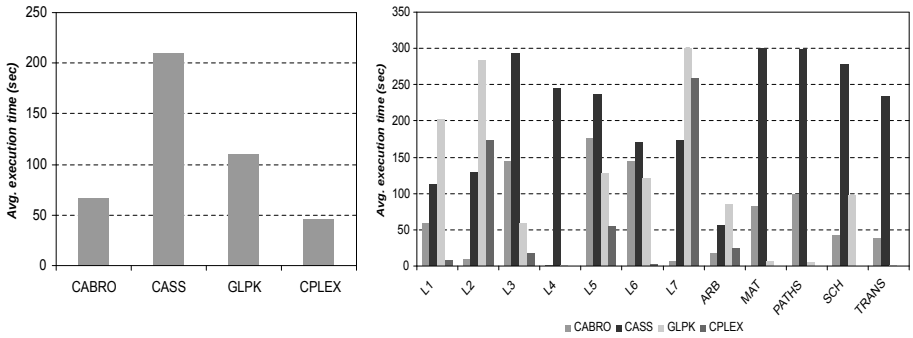

Figure B.4: Left: Global comparative. Right: Comparative over distributions.

branch is finished. Alternatively, if this condition does not apply, 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 $L2$.

B.4 Results

To evaluate the CABRO algorithm we have compared it against both specific algorithms and general purpose MIP solvers. We have chosen CASS for the specific solver instead of CABOB because although their authors claim that it outperforms CASS, there is no implementation of it available publicly. 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) [44], which creates realistic auction instances. The CATS suite generates instances following five real-world situations and seven previously published distributions by different authors (called legacy).

We have also created a new distribution called transports (TRANS) based on a real problem: the road transportation problem. 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 [52] 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.

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, while 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.

Figure B.4 (left) shows the average execution 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 the other solvers. Recall that the machine used for CPLEX is considerably faster than the one used for the others; however, we believe that the results on equal machines would not change significantly. Yet CABRO spends less time than the free solvers GLPK and CASS.

Figure B.4 (right) shows the results in each of the distributions comparing the average time required (in seconds) to solve all the instances of each distribution with the four methods. Here we can observe that in two distributions (L2 and L7) CABRO is clearly the best algorithm and in other one (ARB) is also the best solver but CPLEX is very close. In the rest of distributions CPLEX is the best. Regarding the free solvers, GLPK is cleary the best solver in L3, L5, TRANS, MAT and PATHS while CABRO is cleary the best in L1, L2,L7, ARB and SCH. CASS is only rather competitive in L1, L2, L7 and ARB distributions.

	CABRO			*CASS*			*GLPK*			*CPLEX*		
	F	$\neg F$	%	F	$\neg F$	%	F	$\neg F$	%	F	$\neg F$	%
$L1$	80	15	84.2	67	28	70.5	39	56	41.1	95	0	100.0
$L2$	100	0	100.0	90	10	90.0	7	93	7.0	50	50	50.0
$L3$	54	46	54.0	3	97	3.0	84	16	84.0	98	2	98.0
$L4$	100	0	100.0	22	78	22.0	100	0	100.0	100	0	100.0
$L5$	44	56	44.0	23	77	23.0	61	39	61.0	90	10	90.0
$L6$	53	47	53.0	46	54	46.0	70	30	70.0	100	0	100.0
$L7$	100	0	100.0	68	32	68.0	0	100	0.0	15	85	15.0
ARB	96	4	96.0	86	14	86.0	81	19	81.0	99	1	99.0
MAT	81	19	81.0	0	100	0.0	100	0	100.0	100	0	100.0
$PATHS$	55	17	76.4	1	71	1.4	72	0	100.0	72	0	100.0
SCH	98	2	98.0	9	91	9.0	84	16	84.0	100	0	100.0
$TRANS$	94	6	94.0	24	76	24.0	100	0	100.0	100	0	100.0
$TOTAL$	955	212	81.8	439	728	37.6	798	369	68.4	1019	148	87.3

Table B.1: Finished auctions (F), not finished auctions ($\neg F$) and percentage of finished auctions ($\%$) before the timeout.

Table B.1 shows the number of auctions finished (F), the number of auctions not finished ($\neg F$) and the percentage of finished auctions ($\%$) 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, 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 performs better for the weighted random and binomial distributions, solving 100% of the instances, while CPLEX only solves 15% in L7 and 50% in L2.

B.5 Conclusions

An algorithm for solving combinatorial auction problems has been presented. It uses many reduction techniques, together with an heuristic function based on linear programming techniques that provides more pruning. We have compared its performance with other existing algorithms obtaining encouraging results, particularly for weighted random and binomial distributions.

There is a lot of room for improvements in the algorithm, such as new reduction strategies, a better integration in the search phase, improvements in the upper bound function used in the first phase, other sorting criteria to obtain better lower bounds. Also, a better understanding of the different characteristics of the domains and its influence in the solution time could help to theoretically char-

acterize domains where CABRO outperforms CPLEX and work in the domains where it does not.

Another interesting point would be to extend this algorithm to deal also with multi-unit combinatorial auctions, as there are not many specific algorithms for this kind of auctions. Finally, a comparison of the anytime behavior and the memory consumption could be performed, as it is known to be a drawback of MIP solvers.

Bibliography

[1] A. Aggoun and N. Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathematical Computer Modelling, 17(7)*, pages 57–73, 1993.

[2] J. Argelich, I. Lynce, and J. Marques-Silva. On solving boolean multilevel optimization problems. *In Proceedings of IJCAI 09*, pages 393–398, 2009.

[3] M. Benedetti, A. Lallouet, and J. Vautard. Quantified constraint optimization. In *CP*, pages 463–477, 2008.

[4] T. Berthold, S. Heinz, and M.E. Pfetsch. Solving pseudo-boolean problems with scip. *ZIB-Report 08-12*, 2009.

[5] D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.

[6] M. Bichler, A. Davenport, G. Hohner, and J. Kalagnanam. *Combinatorial Auctions*, chapter Industrial Procurement Auctions, pages 593–612. MIT Press, 2006.

[7] A. Biere, A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 2009.

[8] M. Bofill, R. Nieuwenhuis, A. Oliveras, E. Rodrguez-Carbonell, and A. Rubio. The barcelogic smt solver. *Lecture Notes in Computer Science*, 5123/2008:294–298, 2008.

[9] G. Boolos and R. Jeffrey. *Computability and Logic*. Cambridge University Press, 1974.

[10] R. Bruttomesso, A. Cimatti, A. Franzn, A. Griggio, and R. Sebastiani. The mathsat 4 smt solver. *In Proceedings of CAV, LNCS*, 5123, 2008.

[11] Y. Caseau and F. Laburthe. Cumulative scheduling with task intervals. *Joint International Conference and Symposium on Logic Programming*, pages 363–377, 1996.

[12] 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.

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

[14] S. H. Clearwater. Market-based control: a paradigm for distributed resource allocation. *World Scientific Publishing Co., Inc., River Edge, NJ, USA*, 1996.

[15] CPLEX. http://www.ilog.com/products/cplex/.

[16] P. Cramton, Y. Shoham, and R. Steinberg. *Combinatorial Auctions*. MIT Press, 2006.

[17] G.B. Dantzig. The simplex method. *RAND Corp*, 1956.

[18] 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)*, 2001.

[19] L. de Moura and N. Bjrner. Z3: An efficient smt solver. *Lecture Notes in Computer Science*, 4963/2008:337–340, 2008.

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

[21] B. Dutertre and L. De Moura. The yices smt solver. Technical report, 2006.

[22] N. Een and N. Sorensson. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, 2006.

[23] A.M. Frisch, W. Harvey, C. Jefferson, B. Martínez-Hernández, and I. Miguel. Essence: A constraint language for specifying combinatorial problems. *Constraints, 13(3)*, pages 268–306, 2008.

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

[25] M.L. Ginsberg, A.J. Parkes, and A. Roy. Supermodels and robustness. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 334–339, Menlo Park, CA, USA, 1998.

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

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

[28] E. Hebrard, B. Hnich, B. O'Sullivan, and T. Walsh. Finding diverse and similar solutions in constraint programming. *In AAAI'05: Proceedings of the 20th national conference on Artificial intelligence*, pages 372–377, 2005.

[29] E. Hebrard, B. Hnich, and T. Walsh. Robust solutions for constraint satisfaction and optimization. In *ECAI*, pages 186–190, 2004.

[30] E. Hebrard, B. Hnich, and T. Walsh. Super solutions in constraint programming. In *CPAIOR*, pages 157–172, 2004.

[31] E. Hebrard, B. Hnich, and T. Walsh. Improved algorithm for finding (a, b)-super solutions. *In Proc. of Workshop on Constraint Programming for Planning and Scheduling*, pages 236–248, 2005.

[32] F. Heras, J. Larrosa, S. de Givry, and T. Schiex. 2006 and 2007 max-sat evaluations: Contributed instances. *Journal of Satisfiability, Boolean Modeling and Computation*, 4:239–250, 2008.

[33] W. Herroelen and B. De Reyck. Phase transitions in project scheduling. *Journal of the Operational Research Society, 50(2)*, pages 148–156, 1999.

[34] A. Holland. *Risk Management for Combinatorial Auctions*. PhD thesis, Department of Computer Science, National University of Ireland, Cork, 2005.

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

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

[37] A. Holland and B. O'Sullivan. Truthful risk-managed combinatorial auctions. In *IJCAI*, pages 1315–1320, 2007.

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

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

[40] S. Krauss. Strategic negotiation in multiagent environments. *MIT Press*, 2001.

[41] J.S. Lee and B.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.

[42] J.S. Lee and B.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.

[43] D. Lehman, L.I. O'Callaghan, and Y. Shoham. Truth revelation in aproximately efficient combinatorial auctions. *Journal of the ACM, 49(5)*, pages 577–602, 2002.

[44] K. Leyton-Brown, M.Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In *ACM Conference on Electronic Commerce*, pages 66–76, 2000.

[45] L. Liu and M. Truszczynski. Satisfiability testing of boolean combinations of pseudo-boolean constraints using local-search techniques. *Constraints, 12(3)*, pages 345–369, 2007.

[46] V.M. Manquinho and J. Marques-Silva. Effective lower bounding techniques for pseudo-boolean optimization. In *In Proc. of the conference on Design, Automation and Test in Europe*, pages 660–665. IEEE Computer Society, 2005.

[47] A. Mas-Colell, M.D. Whinston, and Green J. R. *Microeconomic theory*. New York: Oxford University Press, 1995.

[48] T. Matsuo, T. Ito, R.W. Day, and T. Shintani. A robust combinatorial auction mechanism against shill bidders. *Fifth international joint conference on Autonomous agents and multiagent systems, pages 1183-1190, New York, NY, USA*, 2006.

[49] J. McMillan. Selling spectrum rights. pages 145–162, 1994.

[50] P. Modi, W. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence Journal*, 161:149–180, 2005.

[51] V. Munoz and J. Murillo. Cabro: Winner determination algorithm for single-unit combinatorial auctions. In *Proceeding of the 2008 conference on Artificial Intelligence Research and Development*, pages 303–312, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press.

[52] J. Murillo and B. Lopez. An empirical study of planning and scheduling interactions in the road passenger transportation domain. *Proceedings of PlanSIG 2006*, pages 129–136, 2006.

[53] N. Nethercote, P.J. Stuckey, R. Becket, S. Brand, G.J. Duck, and G. Tack. Minizinc: Towards a standard CP modelling language. *13th International Conference on Principles and Practice of Constraint Programming, CP'07, volume 4741 of LNCS*, pages 529–543, 2007.

[54] N. Nisan and A. Ronen. Computationally feasible VCG mechanisms. In *ACM Conference on Electronic Commerce*, pages 242–252, 2000.

[55] N. Nisan and A. Ronen. Algorithmic mechanism design. *Games and Economic Behavior, 35*, pages 166–196, 2001.

[56] T.R. Payne, E. David, N.R. Jennings, and M. Sharifi. Auction mechanisms for efficient advertisement selection on public displays. In *ECAI*, pages 285–289, 2006.

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

[58] R. Porter, A. Ronen, Y. Shoham, and M. Tennenholtz. Fault tolerant mechanism design. *Artificial Intelligence, 172(15):1783-1799*, 2008.

[59] S.D. Ramchurn, C. Mezzetti, A. Giovannucci, J.A. Rodriguez, R.K. Dash, and N.R. Jennings. Trust-based mechanisms for robust and eficient task allocation in the presence of execution uncertainty. *Journal of Artificial Intelligence Research, 35:119-159*, 2009.

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

[61] F. Rossi, P. Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

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

[63] A. Roy. Fault tolerant boolean satisfiability. *Journal of Artificial Intelligence Research, 25:503-527*, 2006.

[64] M.A. Salido and F. Barber. Distributed csps by graph partitioning. *Applied Mathematics and Computation*, 183:491–498, 2006.

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

[66] T. Sandholm, S. Suri, A. Gilpin, and D. Levine. CABOB: A fast optimal algorithm for combinatorial auctions. In *IJCAI*, pages 1102–1108, 2001.

[67] D. Schuurmans, F. Southey, and R. C. Holte. The exponential subgradient algorithm for heuristic boolean programming. *IJCAI*, 2001.

[68] R. Sebastiani. Lazy satisability modulo theories. *Journal on Satisfiability, Boolean Modeling and Computation, 3(3-4):141-224*, 2007.

[69] G. Tchobanoglous, F.L. Burton, and H.D. Stensel. Wastewater engineering. treatment and reuse. *Metcalf and Eddy, Inc., 4th edition, McGraw-Hill, New York*, 2003.

[70] K. Tsuchida, H. Oka, Y. Ikkai, and N. Komoda. A robust scheduling method for a job shop problem in production by using data carriers. In *SMC (2)*, pages 1464–1468, 2004.

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

[72] R. Weigel and C. Bliek. On reformulation of constraint satisfaction problems. *In Proceedings of ECAI*, pages 254–258, 1998.

[73] M.P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.

[74] H.P. Williams. Model building in mathematical programming. *J. Wiley and Sons, New York*, 1978.

[75] M. Yokoo, Y. Sakurai, and S. Matsuraba. Robust combinatorial auction protocol against false-name bids. *Artificial Intelligence Journal, 130(2):167-181*, 2001.

[76] G. Yu. On the max-min 0-1 knapsack problem with robust optimization applications. *Operations Research, 44:407-415*, 1996.