Economic and Computationally Efficient Algorithms for Bidding in a Distributed Combinatorial Auction

By

Benito Mendoza Garcia

Master of Artificial Intelligence
University of Veracruz, 2001

Submitted in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy in

Computer Science and Engineering

College of Engineering and Computing

University of South Carolina

2009

ACKNOWLEDGMENTS

Thanks to my advisor, Professor Vidal, for his patience and willingness to listen, for his always correct advisement, encourage, and support. Thanks to Professor Huhns for his faithful advisement, and support. Thanks to Professor Valtorta for his support. Thanks to Professors Nair and Professors O'Kane for being part of my dissertation committee. Acknowledgment is made to the University of South Carolina's High Performance Computing Group for the computing time used in this research. Thanks to Dr. Buell for encourage me to use the HPCG resources and to Paul Sagona for his help when using those resources. Thanks to all the professors, friends, classmates, and the staff of the computer science department for making my stay in there a stimulating and rewarding time of my life—special acknowledgments for Bervely, Barb, and Jewel for the kindness and friendship the always showed to me and my family. Finally, thanks to the two most important persons in my live: Laura, my wife, for her love, support, help, and for being the source of inspiration that makes me keep going; and Diana, my daughter, who has brought very special things to my live.

ABSTRACT

Combinatorial auctions (CAs)—auctions that allow bids for bundles of items—have generated significant interest as automated mechanisms for buying and selling bundles of scarce resources, since they provide a great way of allocating multiple distinguishable items amongst bidders whose perceived valuations for combinations of those items differ. However, CAs require the establishment of a central auctioneer who receives the bids and carries out all the computation to find the optimal allocation of items to bidders–an NP-complete problem.

The motivation for this dissertation was the vision of **distributed combinatorial auctions** as incentive compatible peer-to-peer mechanisms to solve the allocation problem, where the bidders are the ones who carry out the needed computation to solve the problem; consequently, there is no need for a central auctioneer. The core contributions of this dissertation consist on a set of bidding algorithms for distributed combinatorial auctions. I have developed two type of bidding algorithms: *myopic-optimal*, which optimally find the set of bids that maximize the bidders utility at a certain time with no consideration about the future; and *heuristic-approximate*, which are not guaranteed to find the utility-maximizing set of bids but require only a small fraction of the others' computational time. These algorithms have shown that it is feasible to implement distributed combinatorial auctions.

Experimental results indicate that the approximate algorithms are a realistic tool for the development of large-scale distributed combinatorial auctions. Furthermore, the outcome of a game theoretical analysis indicates that they are dominant strategies over the myopic-optimal ones, since in addition to be faster, they provide higher bidder's utility (although the seller's revenue is not optimal). Empirical analyzes over different problem domains show that these algorithms find highly (allocative) efficient solutions. This makes them suitable to solve, effectively and distributively, complex coordination problems such as multirobot task allocation, automated negotiation in B2B electronic commerce, automated supply chains, and routing mechanisms.

CONTENTS

LIST OF FIGURES

Part 1

Overview

CHAPTER 1

INTRODUCTION

The Internet is used for many of the activities of our daily life and it has become the preferred medium to deliver resources like information and services. The Internet and the increasing development of computer networks have led to an explosive interconnection of devices/machines, systems, and people. This raises many research issues with regards to the ways in which all these distributed components can interact effectively with each others [**15**]. Because the entities (individuals, organizations, programs) that operate those networked components are autonomous, they will generally act to maximize their own self-interest. The interaction of these intelligent and autonomous entities leads to conflicts that have to be solved to achieve their goals. For example, conflicts over the usage of joint resources or task assignments, conflicts concerning document allocation in multi-server environments, and conflicts between a buyer and a seller in electronic commerce.

In all the examples mentioned above, the information needed to make good decisions and the control necessary to implement those decisions is distributed among various heterogeneous components. While many current solutions for these problems assume that all the necessary information can be aggregated at some central location, where effective global decisions are made, such approaches soon will no longer be appropriate for creating complex computer systems. The next generation of computer systems will have to address the existing heterogeneity, autonomy, and dynamism amongst their components and subsystems. **Multiagent systems** focuses on the study of systems in which many intelligent agents—autonomous entities, such as software programs or robots—interact with each other. According to their interactions these agents can be cooperative, sharing a common goal (e.g. an ant colony); or they can be selfish, pursuing their own interests (as in the free market economy) [**42**]. In multiagent systems, the agents have to negotiate with each other in order to solve the conflicts that appear when they try to achieve their individual goals.

Negotiation is a method of competitive (or partially cooperative) allocation of goods, resources, or tasks between agents. It has been a subject of central interest in multiagent systems, as it has been for decades a central subject of study in disciplines such as economics, political science, game theory, and management science. Although the word has been used in a variety of ways, in general it refers to the communication processes that leads to coordination and cooperation amongst agents[**20**]. When discussing negotiation, it is important to distinguish between negotiation protocol and negotiation strategy [**6**]. The **protocol** determines the flow of messages between the negotiating agents, dictating who can say what and when; it provides the rules that the negotiating parties must follow if they are to interact. The protocol is necessarily public and open. The **strategy**, on the other hand, is the way in which a given agent acts within those rules in an effort to get the best outcome of the negotiation. For example, when and what to concede, and when to hold firm. The strategy of each participant is therefore necessarily private.

Through the negotiation literature, we find two main types of allocation mechanisms: bargaining and auctions. **Bargaining mechanisms**—which are known or referred to by most people as negotiation mechanisms—allow the use of more decentralized, flexible protocols. In bargaining mechanisms agents can use incomplete information about their opponent (and their own) preferences. They allow customized and complex agreements and their focus is on designing agent strategies, not the mechanism itself. On the other hand, **auctions**, in general, consist of a fixed protocol and rules, mainly centralized. Via auctions it is possible to design optimal mechanisms that guarantee certain desirable properties, especially in one-shot settings. These mechanisms often target direct revelation—bidders reveal the prices or valuations for preferred combinations—and require the presence of a trusted auctioneer. Auctions have been widely used for real applications, particularly combinatorial auctions.

In recent years, combinatorial auctions (CAs)—auctions that allow bids for bundles of items—have generated significant interest as automated mechanisms for buying and selling bundles of scarce resources. Combinatorial auctions provide a great way of allocating multiple distinguishable items amongst bidders whose perceived valuations for combinations of those items differ. The bundle bidding enabled by these mechanisms allows bidders to benefit from combining the complementarities of the items being auctioned and to better express the value of any synergies. Companies and governments around the world have left behind administered allocation systems, single item auctions, and other ad hoc mechanisms; traditionally used to sell valuable commodities, solve sourcing problems, or allocate scarce public resources; to take advantage of the power of combinatorial auctions based market mechanisms, maximizing revenue and minimizing cost of sales [**39**]. Different from consumer auctions platforms like eBay, combinatorial auctions platforms are used in a high stakes B2B environments [**39**]. For example, recently the Federal Communications Commission (FCC) has risen close to \$20 billion in its 700-MHz auction using these mechanisms. Thus, combinatorial auctions have been fundamentally changing the way of selling valuable resources, allowing the creation of electronic markets to supplement traditional sales channels like bilateral negotiated contracts, and also, creating totally new markets for scenarios where there was no one before.

In a combinatorial auction, once the bidders place their bids, it is necessary to find the allocation of items to bidders that maximizes revenue. For example, if we have a set of 5 figurines, one each of a different X-Mens and we received 6 different combinatorial bids, as shown in Figure 1.1. The question we then face is how to determine which are the winning bids so as to maximize the amount of revenue we receive—this problem is known as the **winner determination problem** (WDP). Note that we can sell each item only once since we only have one of each. In the figure, the correct solution would be to accept both the \$9 bid and the \$7 bid.

The winner determination problem, is a combinatorial optimization problem and is NP-Hard [**34**]. Nevertheless, several algorithms that have a satisfactory performance for

Price	Bid items
$2	Wolverine
$4	Magneto
$6	Rogue, Beast
$7	Gambit, Magneto
$8	Gambit, Wolverine
$9	Rogue, Wolverine

FIGURE 1.1. An example of a combinatorial auction. X-Mens figurines: (from left to right) Wolverine, Gambit, Magneto, Rogue, Beast. The set of combinatorial bids received for them in on the table at the right. The revenue-maximizing solution would be to accept both the $9 bid and the $7 bid.

problem sizes and structures occurring in practice have been developed. However, most of the existing winner determination algorithms for combinatorial auctions are centralized, meaning that they require all agents to send their bids to a centralized auctioneer who then determines the winners (see Figure 1.2 for an illustration).

One problem with the centralized winner determination algorithms, aside from the bottleneck formed in the auctioneer side, is that they require the use of a trusted auctioneer who will perform the needed computations. We believe that distributed solutions to the winner determination problem should be studied as they offer a better fit for some applications as when, for example agents do not want to reveal their valuations to the auctioneer, the auctioneer does not want to perform all the computation or when it is difficult to establish a trusted auctioneer.

1.1. MOTIVATION

The motivation for this dissertation is the vision of **distributed combinatorial auctions** as incentive compatible peer-to-peer mechanisms to solve the allocation problem,

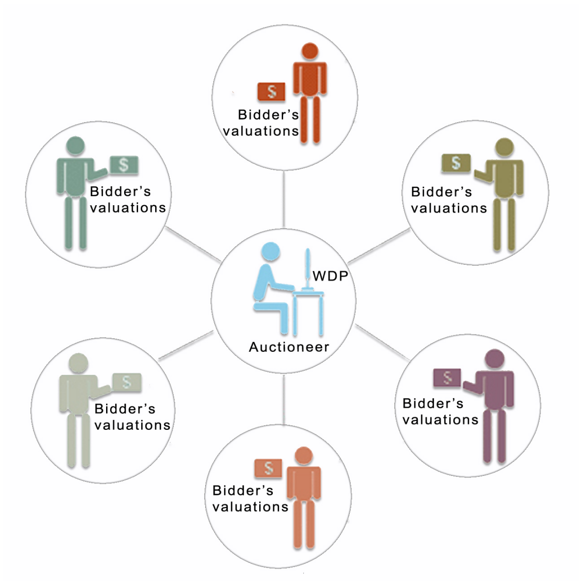


FIGURE 1.2. The centralized model of a combinatorial auction. Once the bidders place their bids, the auctioneer runs a winner determination algorithm to decide the allocation. This model has three major drawbacks: i) the auctioneer has to do all the computation, ii) the bidders have to reveal their valuations to the auctioneer, and ii) some times it is difficult to establish a trusted auctioneer.

where the bidders are the ones who carry out the needed computation, and consequently, there is no need for a central auctioneer. For example, imagine a distributed combinatorial eBay—a Web 2.0 variation on this idea is known as zBay [**43**]—a distributed electronic market place where sellers can advertise their goods and buyers can place combinatorial bids and distributively find near optimal clearings. A more detailed example is a distributed combinatorial reverse auction—an auction used for buying instead of selling—in a B2B scenario. Computer manufacturers (buyers) publish their requirement for computers' components such as memory chips, hard drives, processors, and mother boards. Component manufacturers (sellers) develop agents which try to sell their particular goods by placing combinatorial bids. The initial bids are disclosed to all the sellers. Then, using her own and other sellers bids, each seller agent search for a set of bids (allocations or deals) for which they can get some utility and satisfy all the buyer's requirements. Notice that, if the deal accepted by the buyer does not include a bid from a given agent, this agent gets zero utility. Thus, seller agents negotiate (indirectly) by proposing new bids or adjusting the price of

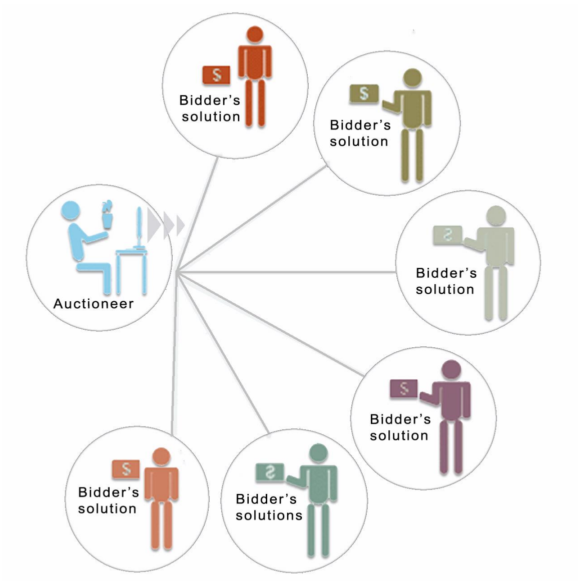


FIGURE 1.3. In the PAUSE auction the task of the auctioneer is reduced to make sure that the bidders follow the rules. The auctioneer is not any more the one running the WDP algorithm.

their current ones to create deals where they maximize their own utility and at the same time reduce the price the buyer has to pay.

A system using the above described mechanism can also effectively and distributively calculate the solution to complex coordination problems. Consider multirobot environments where robots are responsible for accomplishing tasks such as transporting equipments within a manufacturing plant, delivering packages in an office, rescuing victims in situations inaccessible by humans, or tracking enemy targets in battlefields. For such a system to exist, we will need a protocol that distributes the computational task of solving the allocation problem amongst the bidders and strategies for bidding behavior [**28**].

The PAUSE (Progressive Adaptive User Selection Environment) mechanism [**18, 21**] (addressed in more detail in Chapter 7) is an increasing price combinatorial auction that naturally distributes the problem of winner determination amongst the bidders in such a way that they have an incentive to perform the calculation. Thus, the task of the auctioneer is reduced to only make sure that the bidders follow the rules established by PAUSE (Figure 1.3). We can even envision completely eliminating the auctioneer and, instead,

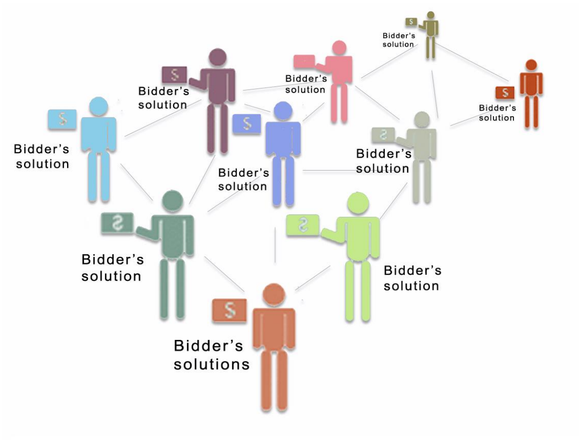


FIGURE 1.4. It is possible to eliminate the PAUSE auctioneer by having every agent to perform the auctioneers task.

have every agent perform the task of the auctioneer (Figure 1.4). A system implementing a PAUSE auction, would allow to achieve much more efficient allocation than would be possible with sequential or simultaneous single item auctions—some of the approaches used to solve combinatorial auctions in a distributed way [**13**]—with no need to rely in a central auctioneer, with no need for the bidders to reveal their true valuations, and eliminating the exposure problem—the problem of exposing the bidders to the possibility that they will win some, but not all the items they desire [**7**].

1.2. CONTRIBUTIONS

PAUSE, as an auction mechanism, establishes the rules the participants have to adhere to so that the work is distributed amongst them. However, it is not concerned with how the bidders determine what they should bid (strategy). The core of the contributions of this dissertation consists on a set of bidding algorithms for the PAUSE auction. I have developed two type of bidding algorithms: **myopic-optimal**, which optimally find the set of bids that maximize the bidders utility at a certain time with no consideration about the future; and **heuristic-approximate**, which are not guaranteed to find the utility-maximizing set of bids but require only a small fraction of the other's computational time. I have implemented them and carried out an experimental study with different types of problems.

These bidding algorithms have shown that it is feasible to implement distributed combinatorial auctions without having to resort to a central auctioneer. Furthermore, the allocations obtained with the PAUSE auction, when using the mentioned algorithms, are very efficient. Thus, a rational bidder agent that aims to maximize her utility would prefer joining to a PAUSE auction over joining to a centralized auction; her incentive is the opportunity of obtaining the items at a lower price, although she has to perform some computational work. I have carried out an experimental game theoretical analysis that shows that the approximate algorithms represent a non-strict Nash equilibrium strategy for the bidders. Thus, a population of bidders will want to use our approximate bidding algorithms rather than performing a complete search to find the optimal bid. I have also analyzed the scalability of our algorithms with respect to the number of goods, bids and bidders; and compare the revenue, allocative efficiency, and bidders' expected utility of the solution found by PAUSE with those of the revenue-maximizing solution. Thus, I have shown that the PAUSE auction along with our heuristic bidding algorithms is a viable method for solving combinatorial allocation problems without the use a centralized auctioneer.

1.3. Outline

This dissertation is divided into four major parts—*Overview*, *Background and Related Work*, *Research Contributions*, and *Conclusions*. The first part is formed by this chapter (Chapter 1), which introduced the audience to the problem addressed in this dissertation, provided an overview of the applications for which solutions to this problem could be adequate, and summarized the contributions of this dissertation.

The second part introduces the literature on the topics related to the problem addressed in this dissertation, namely multiagent systems (Chapter 2), mechanism design (Chapter 3), auctions and combinatorial auctions (Chapter 4), the winner determination problem and the centralized approaches to solve it (Chapter 5), the combinatorial auction test suite CATS (Chapter 6), the details about the PAUSE auction (Chapter 7), and a review related work on distributing the winner determination problem (chapters 8).

The third part includes the results and contributions of this dissertation. Chapter 9 formalizes the problem of bidding in the PAUSE auction. Chapters 10 shows the details of the myopic bidding algorithms including some testing and the empirical results. Similarly, Chapter 11 shows the details of the heuristic-approximate bidding algorithms. Chapter 12 presents a game theoretical analysis that shows that the approximate bidding algorithms represent a dominant strategy for the bidders. The last chapter of this section (Chapter 13) presents an experimental study of the allocations obtained by PAUSE auction, when using the aforementioned bidding algorithms, under different economically motivated problems.

The last part is formed by Chapter 14. This chapter concludes and describes some ideas for future work and extensions.

Part 2

Background and Related Work

Today's computing platforms and information environments are distributed, large, open, and heterogeneous. Computers are no longer stand-alone systems, but have become closely connected both with each other and their users. As mentioned in the introduction chapter, the interaction of these devices leads to several conflicts when trying to access, allocate, or gain some resources and/or perform some tasks. Our research focuses on the study of decentralized incentive compatible mechanisms that can solve these problems (or part of them) based in distributed combinatorial auctions. In this chapter we introduce the concepts and fields needed to understand and formulate the problem and its conceptual solutions. First, we present the concept of agent and, more importantly, the concept of mutiagent systems (MAS) which is, in many cases, the ideal model for understanding, implementing and operating complex socio-technical systems as represented by e-business systems. Second, we introduce the field of mechanism design that deals with the design and characterization of negotiation mechanisms. Then, we introduce the theory behind combinatorial auctions and the winner determination problem, core topics of our research. We also provide a description of the PAUSE mechanism. And finally we describe the related work on distributing combinatorial auctions.

CHAPTER 2

AGENTS AND MULTIAGENT SYSTEMS

The term "agent" is a concept that describes a software abstraction. An agent is an entity that can perceive its environment through sensors and act upon that environment through actuators [**35**]. An agent is defined by its behavior. There are a lot of opinions about what an agent must do, however, there is a common core of concepts that synthesizes these opinions:

> "Agents are active, persistent (software) components that perceive, reason, act, and communicate" [**16**]

Still, the model that has thus far gained most attention, probably due to its flexibility and its well established roots in game theory and artificial intelligence, is that of modeling agents as utility maximizers who inhabit some kind of Markov decision process (MDP). An agent that always tries to optimize an appropriate performance measure is called a **rational agent**. Such a definition of a rational agent is fairly general and can include human agents (having eyes as sensors, hands as actuators), robotic agents (having cameras as sensors, wheels as actuators), or software agents (having a graphical user interface as sensor and as actuator) [**45**]. The modern approach to artificial intelligence (AI) is centered on this concept. **Autonomous agents** are software systems that can independently act on open, unpredictable environments. This concept provides a convenient and powerful way to describe a complex software entity that is capable of acting with a certain degree of autonomy in order to accomplish tasks (involving artificial intelligence techniques and distributed computing), and it is considered by the agents research community as a new paradigm for developing software applications [**48**].

2.1. Multiagent Systems

In the context of concurrent and distributed systems, it becomes obvious that a single agent is insufficient. Many applications, if not most of them, require multiple agents, called also **multiagent systems** (MAS). The goal of multiagent systems research is to find methods that allow us to build complex systems composed of autonomous agents who, while operating on local knowledge and possessing only limited abilities, are nonetheless capable of enacting the desired global behaviors. The main idea is to take a description of what a system of agents should do and break it down into individual agent behaviors.

A common simplifying assumption is that an agent's preferences are captured by a utility function. This function provides a map from the states of the world or outcome of game to a real number (we assume that the agents inhabit some kind of MDP). The bigger the number the more the agent likes that particular state. Specifically, given that S is the set of states in the world the agent can perceive then agent i's utility function is of the form

$$u_i : S \rightarrow \Re. \tag{1}$$

Notice also that the states are defined as those states of the world that the agent can perceive. In practice, agents have sophisticated inputs and it is impractical to define a different output for each input. Thus, most agents also end up mapping their raw inputs to a smaller set of world states. Creating this mapping function can be challenging as it requires a deep understanding of the problem setting. However, given an agent's utility function we can define the agent's preference ordering over the states of the world. By comparing the utility values of two states we can determine which one is preferred by the agent. This ordering has the following properties.

- reflexive: $u_i(s) \geq u_i(s)$
- transitive: If $u_i(a) \geq u_i(b)$ and $u_i(b) \geq u_i(c)$ then $u_i(a) \geq u_i(c)$.
- comparable: either $u_i(a) \geq u_i(b)$ or $u_i(b) \geq u_i(a)$.

We can use utility functions to describe the behavior of almost every rational agent. Utility functions are also useful for capturing the various tradeoffs that an agent must make, along with the value or expected value of its actions. Once we have defined a utility function for all the agents then all they have to do is take actions which maximize their utility. We use the term **selfish agent** to refer to a rational agent that wants to maximize its utility. Utility functions are a succinct way of representing an agent's preferences [**47**].

Multiagent technology is a significant area of interest for such applications as telecommunications, Internet search engines, information retrieval and filtering, computer games, user interface design, information management, electronic commerce, industrial process control, planning, and logistics.

CHAPTER 3

MECHANISM DESIGN

We have talked about self-interested agents in multiagent systems, but how do agents come to an agreement while coordinating their activities? How can agents combine their individual preferences into a common preference? How do we measure the result of the agents' coordination activities?

Negotiation scenarios are governed by a particular mechanism or protocol. The protocol defines the "rules of encounter" between agents [**33**]. When designing a multiagent system, besides the concerns of a traditional communication protocol (free of deadlocks, live lock, etc.), we also need to consider the negotiation protocol's desired properties. **Algorithmic mechanism design** (AMD) is a well-developed subarea of game theory that deals with the design of games with players who have unknown and private utilities. The goal of algorithmic mechanism design research is to design protocols so that any particular negotiation history has certain desirable properties. In [**41**], the following properties are mention as primordial for this kind of protocols:

- **Guaranteed success** – A protocol guarantees success if it ensures that, eventually agreement is certain to be reached.
- **Social welfare** – The sum of all agent's payoffs or utilities in a given solution. Maximum social welfare can be bad for some agents.
- **Pareto efficiency** – A solution is Pareto Efficient (or **Pareto optimal**) if one cannot increase an agent's utility without decreasing another agent's utility. That is, solution x is Pareto efficient if there's no other solution x' such that at least one agent is better off in x' than in x, and no agent is worst off in x' than in x.

- **Individual rationality** – Can participating in the negotiation make things worse? Participation in the negotiations is better than no participation.
- **Stability** – A mechanism should be designed such that the agents' choice of strategy should be stable and non-manipulable (motivate each agent to behave in the desired manner). The best-known kind of stability is **Nash equilibrium**, which is a solution concept of a game involving two or more players, in which no player has anything to gain by changing only his or her own strategy unilaterally. If each player has chosen a strategy and no player can benefit by changing his or her strategy while the other players keep theirs unchanged, then the current set of strategy choices and the corresponding payoffs constitute a Nash equilibrium.
- **Computational efficiency or simplicity** – A mechanisms should be designed so that when agents use them, as little computation is needed as possible.
- **Distribution and communication efficiency** – The mechanism should be designed to ensure that there is no single-point failure (such as a single arbitrator) and minimize the communication overhead.

The use of game-theoretic techniques is very successful in distributed rational decision making or automated negotiations [**19**]. **Distributed algorithmic mechanism design** (DAMD) is field that combines theoretical computer science's traditional focus on computational tractability with its more recent interest in incentive compatibility and distributed computing [**10**].

CHAPTER 4

AUCTIONS

Our research focuses in **combinatorial auctions** (CAs), those auctions in which bidders can place bids on combinations of items, called "packages" rather than just individual items. The study of combinatorial auctions is an important interdisciplinary field combining issues from economics, game theory, optimization, and computer science. Combinatorial auctions are in the first place auctions, a topic extensively studied by economists. Package bidding brings in operations research, especially techniques from combinatorial optimization and mathematical programming. Finally, computer science is concerned with the expressiveness of various bidding languages, and the algorithmic aspects of the combinatorial problem.

To understand the role of combinatorial auctions, it is useful to step back and think about auctions in general. In an auction, agents can express how much they want a particular item via their bid and a central auctioneer can make the allocation based on these bids. Of course, this generally requires the use of a centralized auctioneer but there are techniques for reducing this bottleneck. Still, even centralized auctions can be very complex and produce unexpected results if one does not understand all the details. Abstractly, an auction takes place between an agent known as the auctioneer and a collection of agents known as the bidders. The goal of the auction is for the auctioneer to allocate the good to one of the bidders. Auctions usually end with a deal between two agents (auctioneer and a bidder).

In most settings the auctioneer desires to maximize the price at which the good is allocated or sold—normally, through the design of an appropriate mechanism—while the bidders attempt to minimize that price—by using a strategy that follows the rules of encounter but also delivers optimal result. There are several factors that can affect both the protocol and the strategy that the agents use. The most important of these is the nature of the bidder's valuation.

We have used (in Chapter 2) the notation $u_i(s)$ to refer to the utility that agent i derives from state s. Similarly, if s is instead an item, or set of items, for sale we can say that $v_i(s) \in \Re$ is the valuation that i assigns to s. We furthermore assume that this valuation is expressed in a common currency for all the agents in the auction, thus $v_i(s)$ then becomes the maximum amount of money that agent i is willing to pay for s. When studying auctions we generally assume that all agents have a valuation function over all the items being sold. There exist three types of settings in which the value of an item can be defined:

- **Private value** – In this case the valuation function reflects the agent's utility of owning the given items.
- **Common value** – This is the case when there are items which you cannot consume and gain no direct utility from but which might still have a resale value. For example, in the stock market, when you buy a share in some company you cannot do anything with that share, except sell it. As such, your valuation on that share depends completely on the value that others attribute, and will attribute, to that share.
- **Correlated value** – Most cases, however, lie somewhere in the middle. When you buy a house you take into consideration the value that you will derive from living in that house as well as its appreciation prospects: the price you think others will pay when you finally sell it. Correlated value functions are very common in the real world with durable high priced items.

The type of valuation function that the agents use changes their optimal behavior in an auction. Most multiagent implementations use agents with private value functions as most systems do not want to waste the time required to implement secondary markets for the items being sold. Still, in open multiagent systems it might be impossible to prevent secondary markets from appearing.

4.1. Single Good Auctions

In general, the most well known of auctions are the **single good auctions**, where only one item is being offered for sale. We describe features of the most common auctions of this type.

- **English auction** – Is a first-price, ascending, open-cry auction which dominant strategy is always bid a small amount more than current highest bid, until the bidder's private value is reached or the bidder gets the good paying the price of her bid.
- **Vickrey auction** – Second-price sealed-bid auction, the bidder that bids the highest wins, paying the price of the second highest bid. A bidder dominant strategy in a Vickrey auction is to bid his true valuation. An agent is best off bidding truthfully, no matter what the other bidders are like. The agents reveal their preferences truthfully, this allows globally efficient decisions to be made. No need to waste efforts on gathering information about other agents' preferences.
- **First-price sealed-bid auction** – Each bidder submits one bid, without knowing the other's bids. In in this auction format, it is advantageous for a bidder to gather information about the competing bids before deciding on his own bid; therefore, the "privacy" issue is essential. Bidders bid as a function of their private value and their prior estimates of others' valuations. There is no dominant strategy in general, but is clear that bidders would bid less than the their true valuation.
- **Dutch auction** – Seller continuously lowers the price until one of the bidders takes the item at its current price. Strategically equivalent to the first-price sealed-bid auction. Unlike in the second-price and English auctions, it is not a dominant

strategy in a first-price auction to bid your value. However, the theoretically optimal bidding strategy in both first-price and Dutch auctions is the same for any given bidder.

4.2. COMBINATORIAL AUCTIONS

In a **combinatorial auction** there is a set M of indivisible items that are concurrently auctioned amongst a set of bidders N. The combinatorial aspect of the auction comes from the fact that bidders have preferences regarding subsets of items [**3**].

Formally, every bidder i has a **valuation function** v_i that describes his preferences. Thus, bidder i's value function over a set of items S, the value that i obtains if gets S, is given by $v_i(S) \in \Re$. A valuation function must have **free disposal**—for $S \subseteq T$ we have that $v(S) \leq v(T)$—and it should be **normalized**—$v(\emptyset) = 0$.

There are two different kinds of valuation functions we can consider [**8**]:

- **sub-additive** – A function f is locally sub-additive if for all disjoint sets S_1 and S_2 we have that $f(S_1 \cup S_2) \leq f(S_1) + f(S_2)$. The function expresses substitution effects between goods. This occurs when the bundle is worth less than the valuation for the single goods.
- **super-additive** – A function f is super-additive if for all disjoint sets S_1 and S_2 we have that $f(S_1 \cup S_2) \geq f(S_1) + f(S_2)$. It expresses complementarity between goods. This is the case when the bundle is worth more than the sum of the single goods.

The advantage of combinatorial auctions is that agents can express synergies they have between various items. Using their value functions, bidders can submit bids over bundles. Each bid b is composed of three elements: b^{price}, the value or price of the bid; b^{items}, the set of items the bid is over; and b^{agent}, the agent that placed the bid. Thus, if a bidder i wins a set of items b^{items} paying a price b^{price}, i's **utility** is $u_i(b^{\text{items}}) = v_i(b^{\text{items}}) - b^{\text{price}}$, that is, the difference between its private valuation and the price paid.

An **allocation** of items amongst bidders is a bidset $Y = \{b_1, \ldots, b_n\}$ where $\forall_{b_j, b_k \in Y} b_j^{\text{items}} \cap b_k^{\text{items}} = \emptyset$. The **social welfare** obtained by an allocation Y is $\Sigma_{b \in Y} v_{b^{\text{agent}}}(b^{\text{items}})$. The **social welfare maximizing solution** simply maximizes this equation.

CHAPTER 5

THE WINNER DETERMINATION PROBLEM

The **winner determination problem** (WDP) is a computational problem of how to efficiently determine the allocation of items to biders once the bids have been submitted to the auctioneer. It can be stated as follows: given the set B of bids submitted by all the bidders in a combinatorial auction, find an allocation of items to bidders—including the possibility that the auctioneer retains some items—that maximizes the auctioneer's revenue. More formally, find

$$X^* = \underset{X \subseteq C}{\operatorname{argmax}} \sum_{b \in X} b^{\text{price}}, \tag{2}$$

where C is a set of all bidsets in which none of the bids share an item, that is

$$C = \{Y \subseteq B \,|\, \forall_{a,b \in Y} a^{\text{items}} \cap b^{\text{items}} = \emptyset\}. \tag{3}$$

This problem is difficult for large set of items M. Specifically, it has been proved to be NP-Hard [**34**]. The complexity of the problem resides in the worst case, since there are many possible bundles, specifically if bids exists for all subsets of items. Because of the wide applicability of combinatorial auctions [**8**] one cannot hope for a general-purpose algorithm that can efficiently solve every instance of this problem. Nevertheless, several algorithms that have a satisfactory performance for problem sizes and structures occurring in practice have been developed. However, most of the existing winner determination algorithms for combinatorial auctions are centralized. Examples of these algorithms are CASS [**11**], Bidtree [**38**], and CABOB [**40**]. In the following subsections we describe the existing approaches used by these algorithms.

Using a brute force search of all possible allocations of items to agents is computationally intractable. In fact, as mentioned before, the problem has been proved to be NP-hard

[**34**]. There are three main different approaches that have been used for solving the winner determination problem for combinatorial auctions [**40**]:

- severely **restricting the bundles** on which bids can be submitted so that the remaining problem can be solved optimally and probably fast.
- designing **optimal algorithms** that guarantee to find a solution but are slow on some problem instances.
- designing **approximate algorithms** which are probably fast, but fail to find an optimal solution.

5.1. Restricting the Winner Determination Problem

Even simplifying it, the winner determination problem is not easy to solve. For example, say that instead of trying to find the best allocation we simply want to check if there exists an allocation with total revenue of at least w. We call this the **decision version** of the winner determination problem. Even if it is restricted to instances where every bid has a value equal to 1, every bidder submits only one bid, and every item is contained in exactly two bids, the **decision version** of the winner determination problem in combinatorial auctions has been proved to be NP-complete [**22**].

Because of the NP-hardness of the winner determination problem, attempts to tackle it by restricting the problem to classes of subproblems which can be solved optimally by polynomial time algorithms have been made. The restrictions are put in place by forbidding certain combinations of bids, which narrows the bidders in expressing their real valuations. Disallowing bidding on certain combinations of items, causes that bidders might not be able to bid on all bundles they desire. The research done in this area has focused in reaching two basic goals:

(1) The restriction of bids must be designed in such a way that allows a polynomial algorithm to solve it optimally.

(2) The restricted problems must still be of practical relevance, in other words, scenarios must be found where the bidders are naturally not interested on certain combinations of goods.

Since the pioneer efforts in this area, it has been showed the relevance of this approach in real-world applications. Some polynomial algorithms to solve the resulting winner determination problem optimally are shown in [**34**], where they also consider different structures of permitted bids. For example, in **cardinality-based structures**, bids are only allowed to contain at most two goods. Allowing three goods makes the problems again not solvable in polynomial time. This restriction might be valid for auctions of airport landing slots since flight companies are mostly interested on pairs of landing and take-off slots.

5.2. Optimal Algorithms

There are two approaches used to optimally find the revenue-maximizing bidset X^*. The first one is based on the reformulation of the problem as linear programming problem, which can then be, solved in polynomial time with well-known algorithms. The only restriction is that prices should be attached to single items in the auction. In many cases this requirement can be satisfied by simply adding the missing singleton bids, each with a value of 0 [**29**]. The second approach is to conduct one of the standard artificial intelligence (AI) searches over all possible allocations, given the bids submitted. The advantage of this approach over using a linear programming solver is that the AI search algorithms can be tweaked and be optimized to solve specific problems. That is, we can put some of our domain knowledge into the algorithm to make it run faster, as we shall see.

5.2.1. Linear Programming Approach. As explained before, the winner determination problem is very hard even when we try to limit its complexity. However, the WDP in combinatorial auctions can be reduced to a **linear programming** problem and, therefore, solved in polynomial time with well-known algorithms (due to its rapid development in the last years, the commercial solver ILOG CPLEX has become the state-of-the art algorithm for the WDP using this approach.) but only if prices can be attached to single items in the auction [**29**]. That is, there needs to be a singleton bid for every item. In many cases we can satisfy this requirement by simply adding the missing singleton bids, each with a value of 0. Specifically, the linear program which models the winner determination problem is to find the x that satisfies the following:

Maximize:

$$\sum_{b \in B} x[b] b^{\text{price}}$$

Subject to:

$$\sum_{b \,|\, j \in b^{\text{items}}} x[b] \leq 1, \forall j \in M$$

$$x[b] \in \{0, 1\}, \forall b \in B,$$

, where $x[b]$ is a bit which denotes whether bid b is a winning bid. That is, maximize the sum of the bid values given that each item can be in, at most, one winning bid. It has also been shown that the linear programming problem will solve a combinatorial auction when the bids satisfy any one of the following criteria [**29**]:

(1) When the sets of items can be linearly ordered such that all bids are for consecutive sub-ranges of the items.
(2) The sets of items in all the bids form a nested hierarchy. That is, for every two subsets of items S, S' that appear as part of any bid, we have that either they are disjoint or that one contains the other.
(3) The bids are only OR-of-XORs of singleton bids.
(4) The bids are all singleton bids.
(5) The bids are downward sloping symmetric. That is, each bidder values all items as if they are identical, and has a sequence of valuations $p_1 \geq p_2 \geq \cdots \geq p_m$, where p_j specifies his valuation of the $j'th$ item she wins.

5.2.2. AI-Search Based Approaches. In this class, algorithms for the WDP have concentrated on using branch and bound methods. Branch and bound methods construct decision trees,and try to eliminate certain regions of the solution space by pruning sub-trees.

```
BUILD-BRANCH-ON-ITEMS-SEARCH-TREE
1  Create a singleton bid for any item that does not have one
2  Number items from 1 to $m$
3  Create empty root node
4  for $n \in M$ in order
5      do Add as its children all bids that
6             include the smallest item that is not an ancestor of $n$ but
7             that do not include any item that is an ancestor of $n$.
```

FIGURE 5.1. Algorithm for building a branch on items search tree [**44**]. This algorithms does not find a solution, it only builds a tree for the purpose of illustration.

This is accomplished by finding lower and upper bounds for the unknown optimal solution.

Before we can do search we need to define our search tree. One way we can build a search tree is by having each node be a bid and each path from the root to a leaf correspond to a set of bids where no two bids share an item. In [**44**] an algorithm for building this tree is presented, we show the algorithm in Figure 5.1. We refer to this tree as a **branch on items** search tree. In general, we know that the number of leaves in the tree is bounded. The number of leaves in the tree produced by BUILD-BRANCH-ON-ITEMS-SEARCH-TREE is no greater than $(|B|/|M|)^{|M|}$. The number of nodes is no greater than $|M|$ times the number of leaves plus 1 [**38**].

We can also build a binary tree where, as before, each node is a bid, but this time, each edge represents whether or not that particular bid is in the solution. We refer to this tree as a **branch on bids** search tree. Each edge in the tree indicates whether the parent node (bid) is to be considered as part of the final bidset.

We now have to decide how to search our chosen tree. Since both trees have a number of nodes that is exponential on the number of bids a breadth first search would require too much memory. However, a depth first search is possible, but time consuming. A branch and bound algorithm helps to reduce the search space and speed up computation. In order to implement it we first need a function h which gives us an upper bound on the value of

```
BRANCH-ON-BIDS-CA()
1 $r* \leftarrow 0$    $\triangleright$ Max revenue found. Global variable.
2 $g* \leftarrow \emptyset$    $\triangleright$ Best solution found. Global variable.
3 BRANCH-ON-BIDS-CA-HELPER($\emptyset$)
4 return $g^*$

BRANCH-ON-BIDS-CA-HELPER($g$)
1 if $\bigcup_{b \in g} b^{\text{items}} = M$                  $\triangleright$ $g$ covers all items
2    then if $\sum_{b \in g} b^{\text{price}} > r^*$             $\triangleright$ $g$ has higher revenue than $r^*$
3            then $g^* \leftarrow g$
4                 $r^* \leftarrow \sum_{b \in g} b^{\text{price}}$
5         return
6 for $b \in \{b \in B \mid b^{\text{items}} \cap \bigcup_{b_1 \in g} b_1^{\text{items}} = \emptyset\}$ $\triangleright$ $b$'s items do not overlap $g$
7      do $g' \leftarrow g + b$
8         if $\sum_{b_1 \in g'} b_1^{\text{price}} + h(g') > r^*$
9            then BRANCH-ON-BIDS-CA-HELPER($g'$)
```

FIGURE 5.2. A centralized branch and bound algorithm that searchers a branch on bids tree and finds the revenue maximizing solution given a set B of combinatorial bids over items M [**44**].

allocating all the items that have yet to be allocated. One such function is h

$$h(g) = \sum_{j \in M - \bigcup_{b \in g} b^{\text{items}}} \max_{b | j \in b^{\text{items}}} \frac{b^{\text{price}}}{|b^{\text{items}}|}, \tag{4}$$

where g is the set of bids that have been cleared. The function h simply adds up the maximum possible revenue that each item not in g could contribute by using the bid that pays the most for each item, divided by the number of items on the bid. This function provides an upper bound since no feasible bidset with higher revenue can exist.

Given the upper bound $h(g)$ we can then implement the branch and bound algorithm shown in Figure 5.2. This algorithm searches the branch on bids tree. It maintains a partial solution g to which it adds one bid on each recursive call. Whenever it realizes that partial solution will never be able to achieve revenue that is higher than the best revenue it has already found then it gives up on that subtree, see line 8 of BRANCH-ON-BIDS-CA-HELPER. This algorithm is complete and thus guaranteed to find the revenue maximizing bidset.

We can also use the same heuristic function to do an A^* search. Unfortunately, since A^* acts much like a breadth first search it generally consumes too much memory. A viable solution is to use iterative deepening A^*. IDA^* guesses how much revenue we can expect and runs a depth-first search that prunes nodes that have used more than that. If a solution is not found then the guess is reduced and we try again. IDA^*, with some optimizations, was implemented by the *Bidtree* algorithm [**37**] on the branch on items search tree. In practice, this approach was found to often be slower than a branch and bound search.

The BRANCH-ON-BIDS-CA algorithm is the basic framework for the Combinatorial Auction Branch on Bids (CABOB) algorithm [**40**]. CABOB improves the performance of the basic algorithm in several ways, one of which is by improving the search for new bids to add to the partial solution. Specifically, we note that a naive implementation of line 6 of BRANCH-ON-BIDS-CA-HELPER would mean that we would build this set on each recursive call to the function. That would be very time consuming as there are an exponential number of bids in B. CABOB handles this problem by maintaining graph data structure which has all the bids that can still be used given g. The nodes in the graph are the bids that are still available and the edges connect every pair of bids that share an item. In this way when a new bid is added to g it is removed from the graph as well as all the other bids that are connected to it.

We can also perform the branch and bound search on the branch on items search tree, as shown in Figure 5.3. This algorithm is the basis for the CASS (Combinatorial Auction Structured Search) algorithm which also implements further refinements on the basic algorithm [**11**].

Most algorithms for centralized winner determination in combinatorial auction expand on the basic branch and bound search by using specialized data structures to speed up access to the information need—the viable bids given the current partial solution—and implement heuristics which have been shown to reduce the size of the search space, especially for certain popular bid distributions. In general the best speed attainable by the best algorithms

```
BRANCH-ON-ITEMS-CA()
1  $r* \leftarrow 0$   $\triangleright$ Max revenue found. Global variable.
2  $g* \leftarrow \emptyset$   $\triangleright$ Best solution found. Global variable.
3  BRANCH-ON-ITEMS-CA-HELPER$(1, \emptyset)$
4  return $g^*$

BRANCH-ON-ITEMS-CA-HELPER$(i, g)$
1  if $i = m$                                  $\triangleright$ $g$ covers all items
2     then if $\sum_{b \in g} b^{\text{price}} > r^*$            $\triangleright$ $g$ has higher revenue than $r^*$
3              then $g^* \leftarrow g$
4                   $r^* \leftarrow \sum_{b \in g} b^{\text{price}}$
5          return
6  for $b \in \{b \in B \mid i \in b^{\text{items}} \wedge b^{\text{items}} \cap \bigcup_{b_1 \in g} b_1^{\text{items}} = \emptyset\}$ $\triangleright$ $b$'s items do not overlap $g$
7      do $g' \leftarrow g + b$
8          if $\sum_{b_1 \in g'} b_1^{\text{price}} + h(g') > r^*$
9             then BRANCH-ON-BIDS-CA-HELPER$(g')$
```

FIGURE 5.3. A centralized branch and bound algorithm that searchers a branch on items tree and finds the revenue maximizing solution given a set B of combinatorial bids over items M [**44**].

varies greatly depending on the type of bids submitted. Some optimizations and heuristics that have been found useful include the following [**44**]:

- Keep only the highest bid for any set. That is, if there is a bid of \$10 for items 1,2 and another bid of \$20 for 1,2 then we get rid of the \$10 bid.
- Remove provably noncompetitive bids, that is, those that are dominated by another bid or sets of bids. For example, if there is a bid for \$10 for item 1 and another bid for \$5 for items 1,2 then the \$10 bid dominates the \$5 bid—any situation in which we choose the \$5 bid would be made better if we changed that bid for the \$10 bid.
- Decompose bids into connected sets, each solved independently. If we can separate the set of bids into two or more sets of bids where all bids for any item are to be found in only one of the sets then this set of bids becomes a smaller, and independent, winner determination problem.

- Mark noncompetitive tuple of bids. For example, if there are bids \$1:(1,2), \$1:(3,4), \$10:(1,3), \$10:(2,4) then the pair of \$10 bids dominates the pair of \$1 bids, so we can get rid of them.
- In the branch-on-items tree place the items with the most bids first on the tree. This way the most constrained items are tried first thereby creating fewer leafs.
- If the remaining search subtree is the same for different nodes in the search tree, as can happen when different items are cleared but by different bids, then these subtrees can be cached. The subtree is solved once and the answer, that is, the best set of bids found in it, is saved for possible future use.

5.3. Approximate Algorithms

Approximate (or non-optimal) algorithms are another way to tackle the problem of finding an optimal bidset. These algorithms usually cannot guarantee the quality of the solution but have fast running times. Some of them are also **any-time algorithms**, when a reasonable solution is available at any time but it becomes better with time.

One of the simplest methods is the **greedy algorithm** [**23**]. The basic algorithm can be summarized into two steps:

(1) The bids are sorted by $b^{\text{price}}/|b^{\text{items}}|^c$ for some number c, $0 \leq c \leq 1$. The authors showed that $c = 0.5$ was the approximate best value, it guarantees an approximation ratio of at least $\sqrt{M}$, where M is the number of goods.

(2) Proceed down the sorted list of bids accepting bids if the goods in demand are still unallocated and not conflicted, where bids b_j and b_k conflict if $b_j^{\text{items}} \cap b_k^{\text{items}} \neq \emptyset$.

Several other similar techniques and meta-heuristics have been used, such as stochastic local search [**14**], limited discrepancy search [**36**], a combination of approximated positive linear programming and stepwise random updates of allocations [**49**], genetic algorithms [**2**], and hill climbing and simulated annealing [**12**].

CHAPTER 6

THE COMBINATORIAL AUCTION TEST SUITE (CATS)

CATS (Combinatorial Auction Test Suite) is a generator of combinatorial auction instances for the testing of winner determination algorithms [**24**]. It features five distributions of instances from realistic, economically motivated domains, as well as a collection of artificial distributions that have been used in the literature. The following is the list of bid distributions with a economically motivated domains:

- **Paths in space** – Is based in real-world domains like *railroad network, truck shipping, network bandwidth allocation, natural gas pipeline, etc*. In this domain the goods are edges in a graph, and the bidders acquire a path from a to b by buying a set of edges. CATS generates a random graph, instead of using a real railroad map, to be able to scale in the number of goods. Based in this graph, it generates bids for each bidder.
- **Proximity in space** – The real-world domain that motivates this distribution is *real estate*. Here the goods are nodes in a graph and the edges indicate adjacency between goods, and bidders are interested on buying a set of adjacent nodes according to common and private values.
- **Arbitrary (*arbitrary relationships*)**. The motivation of this distribution is the type of problems where the goods do not give rise to a notion of adjacency, but regularity in complementarity relationships can still exist, for example, physical objects like collectables, semiconductors, etc.
- **Temporal Separation (matching)** – The real-world problems that motivate this distribution are those where *time slices must be secured on multiple resources*. For example, aircraft take-off and landing rights. It is based in the map of airports

for which take-off and landing rights are actually sold, the four busiest airports in the USA (La Guardia, Kennedy, O'Hare, and Reagan). In this airport map, the goods are time slots, not nodes or edges, thus, a random graph is not needed for scalability.

- **Scheduling (*Temporal Adjacency*).** The motivation of this distribution is the problem of *distributed job-shop scheduling with one resource*. Here, the bidders want to use the resource for a given number of time units. They have one or more deadlines with different values for them. It is assumed that all jobs are eligible to start in the first time-slot and each job is allocated continuous time on the resource.

As mentioned before, CATS has also a legacy distributions section, since CA algorithm researchers have compared performance using each other's distributions [**1, 4, 9, 11, 32, 37**]. Despite some drawbacks (e.g. they do not explicitly model bidders and lack a real-world economic motivation [**24**]), these distributions remain important for comparing new work to previously published work.

CHAPTER 7
THE PAUSE AUCTION

The PAUSE auction is an increasing price combinatorial auction that naturally distributes the problem of winner determination among the bidders. In the PAUSE auction, the bidders have an incentive to perform the needed computation, as shown below. A PAUSE auction for m items has m stages:

(1) Stage 1 consists of having simultaneous ascending price open-cry auctions. During this stage the bidders can only place bids on individual items. At the end of this state we will know what the highest bid for each individual item is and who placed that bid.

(2) Each successive stage $k = 2, 3, \ldots, m$ consists of an ascending price auction. In these stages the bidders must submit bidsets that cover all items but each one of the bids must be for k items or less.

In stages $k \geq 2$ the bidders are allowed to use bids that other agents have placed in previous rounds when building their bidsets, thus allowing them to find better solutions (remember that their bidsets have to cover all items). Also, any new bidset has to have a sum of bid prices which is bigger than that of the currently winning bidset. At the end of each stage k all agents know the best bid for every subset of size k or less. Also, at any point in time after stage 1 has ended there is a standing bidset whose value increases monotonically as new bidsets are submitted.

Since in the final round all agents consider all possible bidsets, we know that the final winning bidset will be one such that no agent can propose a better bidset. Note, however, that this bidset is not guaranteed to be the one that maximizes revenue since we are using an ascending price auction so the winning bid for each set will be only slightly bigger than the second highest bid for the particular set of items. That is, the final prices will not be the same as the prices in a traditional combinatorial auction where all the bidders bid their true valuation; which represent an incentive for the bidders to participate in this type of auction. However, there remains the open question of whether the final distribution of items to bidders found in a PAUSE auction is the same as the revenue maximizing solution. Chapters 10, 11, and 12 provide answers to this question by presenting the tests and results of the set of bidding strategies we have developed.

7.1. The Job of the Auctioneer in the PAUSE Auction

The PAUSE auction makes the job of the auctioneer very easy. All it has to do is to make sure that each new bidset has a revenue bigger than the current winning bidset, as well as make sure that every bid in an agent's bidset that is not his does indeed correspond to some other agents' previous bid (Figure 1.3). The computational problem shifts from one of winner determination to one of bid generation. Each agent must search over the space of all bidsets which contain at least one of its bids. The search is made easier by the fact that the agent needs to consider only the current best bids and only wants bidsets where its own utility is higher than in the current winning bidset. Each agent also has a clear incentive for performing this computation, namely, its utility only increases with each bidset it proposes (of course, it might decrease with the bidsets that others propose). Finally, the PAUSE auction has been shown to be envy-free in that at the conclusion of the auction no bidder would prefer to exchange his allocation with that of any other bidder [**8**].

We can even envision completely eliminating the auctioneer and, instead, have every agent perform the task of the auctioneer (Figure 1.4). That is, all bids are broadcast and when an agent receives a bid from another agent it updates the set of best bids and determines if the new bid is indeed better than the current winning bid. The agents would have an incentive to perform their computation as it will increase their expected utility. Also, any lies about other agents' bids are easily found out by keeping track of the bids sent out by every agent (the set of best bids). Namely, the only one that can increase an agent's bid value is the agent itself. Anyone claiming a higher value for some other agent is lying. The only thing missing is an algorithm that calculates the utility-maximizing bidset for each agent.

CHAPTER 8
RELATED WORK

Although the research of various aspects of combinatorial auctions is vast (we recommend [**8**] for a good review), the study of distributed winner determination algorithms for combinatorial auctions is still relatively new (besides the PAUSE auction there are few other attempts to do this). As mentioned in the introduction chapter, a simple way to distribute the winner determination problem is by using sequential or simultaneous single item auctions. However, this approach exposes the bidders to the possibility that they will win some, but not all the items they desire [**7**]. Another way of distributing the WDP among the bidders is provided by the Virtual Simultaneous Auction (VSA) [**11**] which is based on market-oriented programming ideas [**46**]. It is an iterative algorithm where successive auctions for the items are held and the bidders change their bids based on the last auction's outcome. The auction is guaranteed to find the optimal solution when the bidding terminates. Unfortunately, there is no guarantee that bidding will terminate and experimental results show that in most cases bidding appears to go on forever. Another approach consists of the algorithms for distributing the WDP in combinatorial auctions presented in [**28**], but these algorithms assume the computational entities are the items being sold and thus end up with a different type of distribution. In [**31**] the authors present a distributed mechanism for calculating VCG payments in a mechanism design problem. Their mechanism roughly amounts to having each agent calculate the payments for two other agents and give these to a secure central server which then checks to make sure results from all pairs agree, otherwise a re-calculation is ordered. This general idea, which they call the redundancy principle, could also be applied to our problem but it requires the existence of a secure center agent that everyone trusts. Another interesting approach is given in [**30**] where the bidding agents prioritize their bids, thus reducing the set of bids that the centralized winner determination algorithm must consider, making the problem easier. Finally, in the computation procuring clock auction [**5**] the agents are given an ever-increasing percentage of the surplus achieved by their proposed solution over the current best. As such, it assumes the agents are impartial computational entities, not the set of possible buyers as assumed by the PAUSE auction.

Part 3

Research Contributions of this Dissertation

CHAPTER 9

BIDDING IN THE PAUSE AUCTION: PROBLEM FORMULATION

In the PAUSE auction the bidders are the ones who do the computation to find the allocation. Their job is to find the set of bids that form a valid solution (bidset) and that maximizes their utility. The agents maintain a set B of the current best bids, one for each set of items of size $\leq k$, where k is the current stage. At any point in the auction, after the first round, there will also be a set $W \subseteq B$ of currently winning bids. This is the set of bids that covers all the items and currently maximizes the revenue, where the revenue (r) of W is given by

$$r(W) = \sum_{b \in W} b^{\text{price}}. \tag{5}$$

Agent i's value function is given by $v_i(S) \in \Re$ where S is a set of items. Given an agent's value function and the current winning bidset W we can calculate the agent's utility from W as

$$u_i(W) = \sum_{b \in W \,|\, b^{\text{agent}} = i} v_i(b^{\text{items}}) - b^{\text{price}}. \tag{6}$$

That is, the agent's utility for a bidset W is the value it receives for the items it wins in W minus the price it must pay for those items. If the agent is not winning any items then its utility is zero.

The goal of the bidding agents in the PAUSE auction is to maximize their utility, subject to the constraint that their next set of bids must have a total revenue that is at least ϵ bigger than the current revenue, where ϵ is the smallest increment allowed in the auction. Formally, given that W is the current winning bidset, agent i must find a g_i^* such that

$$g_i^* = \arg\max_{g \subseteq 2^B} u_i(g_i), \tag{7}$$

where each g_i is a set of bids that covers all items, $r(g_i) \geq r(W) + \epsilon$, and $\forall_{b \in g}$ $(b \in B)$ or $(b^{\text{agent}} = i$ and $b^{\text{price}} > B(b^{\text{items}})$ and $|b^{\text{items}}|) \leq k)$, and $B(items)$ is the value of the bid in B for the set $items$ (if there is no bid for those items it returns zero). That is, each bid b in g must satisfy at least one of the two following conditions:

(1) b is already in B,

(2) b is a bid of size less than or equal to k in which the agent i bids higher than the price for the same items in B.

If we have an algorithm that can find that g_i^*, we would be interested in analyzing the solutions that this algorithm and the PAUSE auction generate. We would be interested on determining how long it would take for populations of agents using this algorithm to arrive at a solution, as well as quantifying the bidders expected utility of this solution. Also we would be interested in knowing if the agents in a PAUSE auction arrive at the revenue-maximizing solution. In the following section we define some metrics to analyze a bidding algorithm for the PAUSE auction.

9.1. METRICS

As designers we would be interested in the *(allocative) efficiency* of the solutions found by the PAUSE auction. One of the most widely used criteria for comparing alternative mechanism by comparing the solutions that the mechanism lead to, is the *social welfare*. The social welfare is the sum of all agents' payoffs or utilities in a given solution. It measures the global good of the agents. Thus, an **optimal solution** would be one that maximizes the social welfare (as explained in Section 4.2).

Additionally, knowing the optimal solution of an auction, we could compare the efficiency of the PAUSE auction as ratio to the optimal solution. This would be useful for knowing how different is a solution obtained by PAUSE from the optimal solution, in terms of global good of the agents. We define the **efficiency ratio** to be the ratio of the sum of the valuations of the winning bidders for the bids they win to the optimal solution.

$$efficiencyRatio(W_s) = \frac{\sum_{b \in W_s} v_{b^{\text{agent}}}(b^{\text{items}})}{r*}, \tag{8}$$

where $v_{b^{\text{agent}}}(b^{\text{items}}) \in \Re$ is the valuation that the bidder b^{agent}, the winner of bid b, has for items b^{items}. It gives us the efficiency of the allocation W_s found by PAUSE using the bidding algorithm s as compared to $r*$, the social welfare of the optimal solution. A ratio of 1 means that the PAUSE solution has allocated the items to the same buyers that they are allocated to in the optimal allocation—or at least to a set of buyers with the same sum of valuations.

In an auction the seller wants to maximize the *revenue*. We can compare the revenue generated by the PAUSE auction to the maximum possible revenue for that auction, which would be obtained if the bidder agents bid their private valuation [1]. We can do this by calculating a **revenue ratio**.

$$revenueRatio(W_s) = \frac{r(W_s)}{r*}, \tag{9}$$

where $r*$ is the social welfare of the optimal solution—the maximum revenue assuming that bidders bid truthfully—and $r(W_s)$ is the revenue generated by the allocation W_s found by the PAUSE auction using bidding algorithm s. As mentioned before, we know beforehand that the prices paid in PAUSE are lower than those that would be paid in a in a centralized one-shot first price sealed-bid combinatorial auction assuming truthfully bidding, because PAUSE is an English auction. Thus, the prices paid are roughly the second highest price plus some ϵ.

In a one-shot first price sealed-bid combinatorial auction, assuming that bidder pay their true valuation, the bidders' utility is zero. In the PAUSE auction, the bidders' utility can be greater than zero, since as mentioned before, the winners end up paying less than their

[1]although this is not a realistic scenario, since in practice bidders tend to bid lower than their true valuation, unless the mechanism is incentive compatible and strategyproof like the Vickrey-Clarke-Groves (VCG) [**17**]

private valuations; given that PAUSE is an increasing price auction. When considering whether or not to participate in a PAUSE auction, bidders would be more interested in knowing the *expected utility* from switching to the PAUSE auction or, more precisely, the expected utility of choosing amongst the different bidding algorithms. We calculate the **bidders' expected utility ratio** by dividing the sum of the bidders utility by $r*$.

$$expectedUtilityRatio(W_s) = \frac{\sum_{b \in W_s} u_{b^{\text{agent}}}(b)}{r*}, \qquad (10)$$

where $u_{b^{\text{agent}}}(b) = v_{b^{\text{agent}}}(b^{\text{items}}) - b^{\text{price}}$ is the utility obtained by bidder b^{agent}, who is wining bid b; and as before, W_s is the allocation found by PAUSE using the bidding algorithm s and $r*$ is the social welfare of the optimal solution.

In order to determine the scalability of the PAUSE auction we can analyze the *running time* required to finish an auction. In general, the time required to find the winning bidset is tied to the number of goods and bids in the auction. In the PAUSE auction there is one more factor involved, the number of bidders, since they are the ones that actually solve the problem. Knowing how the PAUSE auction scales based on the bidding algorithm is of interest not only to the seller and the buyer, but also the designer.

CHAPTER 10

MYOPICALLY-OPTIMAL BIDDING ALGORITHMS FOR A DISTRIBUTED COMBINATORIAL AUCTION

According to the PAUSE auction, during the first stage we have only several English auctions, with the bidders submitting bids on individual items. In this case, an agent's dominant strategy is to bid ϵ higher than the current winning bid until it reaches its valuation for that particular item. Our algorithms focus on the subsequent stages: $k > 1$. When $k > 1$, agents have to find g_i^*. This can be done by performing a complete search on B. However, this approach is computationally expensive since it produces a large search tree. Our algorithms represent alternative approaches to overcome this expensive search.

10.1. THE PAUSEBID ALGORITHM

In the PAUSEBID algorithm (shown in Figure 10.1) we use branch and bound to prune the search tree. Given that bidders want to maximize their utility and that at any given point there are likely only a few bids within B which the agent can dominate, we start by defining $my\text{-}bids$ to be the list of bids for which the agent's valuation is higher than the current best bid, as given in B. We set the value of these bids to be the agent's true valuation—but we won't necessarily be bidding true valuation, as we explain later. Similarly, we set $their\text{-}bids$ to be the rest of the bids from B. Finally, the agent's search list is simply the concatenation of $my\text{-}bids$ and $their\text{-}bids$. Note that the agent's own bids are placed first on the search list as this will enable us to do more pruning (PAUSEBID lines 3 to 9). The agent can now perform a branch and bound search on the branch-on-bids tree produced by these bids, implemented by PBSEARCH (Figure 10.3). Our algorithm not only implements the

```
PAUSEBID$(i, k)$
1  $my\text{-}bids \leftarrow \emptyset$
2  $their\text{-}bids \leftarrow \emptyset$
3  for $b \in B$
4      do if $b^{\text{agent}} = i$ or $v_i(b^{\text{items}}) > b^{\text{price}}$
5          then $my\text{-}bids \leftarrow my\text{-}bids +$ new Bid$(b^{\text{items}}, i, v_i(b^{\text{items}}))$
6          else $their\text{-}bids \leftarrow their\text{-}bids + b$
7  for $S \in$ subsets of $k$ or fewer items such that $v_i(S) > 0$ and $\neg\exists_{b \in B} b^{\text{items}} = S$
8      do $my\text{-}bids \leftarrow my\text{-}bids +$ new Bid$(S, i, v_i(S))$
9  $bids \leftarrow my\text{-}bids + their\text{-}bids$
10 $g^* \leftarrow \emptyset$            $\triangleright$ Global variable
11 $u^* \leftarrow u_i(W)$        $\triangleright$ Global variable
12 PBSEARCH$(bids, \emptyset)$
13 $g^* \leftarrow$ DISTRIBUTEPAYMENTS$(i, g^*)$
14 return $g^*$
```

FIGURE 10.1. The PAUSEBID algorithm which implements a branch and bound search. i is the agent and k is the current stage of the auction, for $k \geq 2$.

```
DISTRIBUTEPAYMENTS$(i, g)$
1  $surplus \leftarrow \sum_{b \in g \mid b^{\text{agent}} = i} b^{\text{price}} - B(b^{\text{items}})$
2  if $surplus > 0$
3     then $min\text{-}payment \leftarrow \max(0, r(W) + \epsilon - (r(g) - r_i(g)), \sum_{b \in g \mid b^{\text{agent}} = i} B(b^{\text{items}}))$
4          for $b \in g \mid b^{\text{agent}} = i$
5              do if $min\text{-}payment \leq 0$
6                    then $b^{\text{price}} \leftarrow B(b^{\text{items}})$
7                    else $b^{\text{price}} \leftarrow B(b^{\text{items}}) + min\text{-}payment \cdot \frac{b^{\text{price}} - B(b^{\text{items}})}{surplus}$
8  return $g$
```

FIGURE 10.2. The DISTRIBUTEPAYMENTS function distributes the payments of the bids agent i, included in g, proportionally to the agent's true valuation for each set of items.

standard bound but it also implements other pruning techniques in order to further reduce the size of the search tree.

The bound we use is the maximum utility that the agent can expect to receive from a given set of bids. We call it u^*. Initially, u^* is set to $u_i(W)$ (PAUSEBID line 11), where W is the current winning bidset, since that is the utility the agent currently receives and any solution he proposes should give him more utility. If PBSEARCH ever comes across a partial solution where the maximum utility the agent can expect to receive is less than u^* then that subtree is pruned (PBSEARCH line 22). We can determine the maximum utility only after the algorithm has searched over all of the agent's own bids (which are first on the list) because after that we know that the solution will not include any more bids where the agent is the winner thus the agent's utility will no longer increase. For example, if an agent has only one bid in $my\text{-}bids$ then the maximum utility he can expect is equal to his value for the items in that bid minus the minimum possible payment we can make for those items and still come up with a set of bids that has revenue greater than $r(W)$. The calculation of the minimum payment is shown in line 19 for the partial solution case and line 9 for the case where we have a complete solution in PBSEARCH. Note that in order to calculate the $min\text{-}payment$ for the partial solution case we need an upper bound on the payments that we must make for each item. This upper bound is provided by

$$h(S) = \sum_{s \in S} \max_{\{b \in B \mid s \in b^{\text{items}}\}} \frac{b^{\text{price}}}{|b^{\text{items}}|}. \tag{11}$$

This function produces a bound identical to (4), the one used by the Bidtree algorithm—it merely assigns to each individual item in S a value equal to the maximum bid in B divided by the number of items in that bid.

To prune the branches that cannot lead to a solution with revenue greater than the current W, the algorithm considers both the values of the bids in B and the valuations of the agent. Similarly to (11) we define

$$h_i(S, k) = \sum_{s \in S} \max_{\{S' \mid s \in S' \wedge v_i(S') > 0 \wedge |S'| \leq k\}} \frac{v_i(S')}{|S'|} \tag{12}$$

which assigns to each individual item $s \in S$ the maximum value produced by the valuation of S' divided by the size of S', where S' is a set of items for which the agent has a valuation greater than zero, contains s, and its size is less or equal than k. The algorithm uses the heuristics h and h_i (lines 15 and 19 of PBSEARCH), to prune the just mentioned branches. A final pruning technique implemented by the algorithm is ignoring any branches where

```
PBSEARCH($bids, g$)
 1  if $bids = \emptyset$ then return
 2  $b \leftarrow$ first($bids$)
 3  $bids \leftarrow bids - b$
 4  $g \leftarrow g + b$
 5  $\bar{I}_g \leftarrow$ items not in $g$
 6  if $g$ does not contain a bid from $i$
 7     then return
 8  if $g$ includes all items
 9     then $min\text{-}payment \leftarrow \max(0, r(W) + \epsilon - (r(g) - r_i(g)), \sum_{b \in g \,|\, b^{\text{agent}}=i} B(b^{\text{items}}))$
10          $max\text{-}utility \leftarrow v_i(g) - min\text{-}payment$
11          if $r(g) > r(W)$ and $max\text{-}utility \geq u^*$
12             then $g^* \leftarrow g$
13                  $u^* \leftarrow max\text{-}utility$
14          PBSEARCH($bids, g - b$) $\triangleright$ $b$ is Out
15     else $max\text{-}revenue \leftarrow r(g) + \max(h(\bar{I}_g), h_i(\bar{I}_g))$
16          if $max\text{-}revenue \leq r(W)$
17             then PBSEARCH($bids, g - b$) $\triangleright$ $b$ is Out
18          elseif $b^{\text{agent}} \neq i$
19             then $min\text{-}payment \leftarrow (r(W) + \epsilon) - (r(g) - r_i(g)) - h(\bar{I}_g)$
20                  $max\text{-}utility \leftarrow v_i(g) - min\text{-}payment$
21                  if $max\text{-}utility > u^*$
22                     then PBSEARCH($\{x \in bids \mid x^{\text{items}} \cap b^{\text{items}} = \emptyset\}, g$) $\triangleright$ $b$ is In
23                  PBSEARCH($bids, g - b$) $\triangleright$ $b$ is Out
24          else
25                  PBSEARCH($\{x \in bids \mid x^{\text{items}} \cap b^{\text{items}} = \emptyset\}, g$) $\triangleright$ $b$ is In
26                  PBSEARCH($bids, g - b$) $\triangleright$ $b$ is Out
27  return
```

FIGURE 10.3. The PBSEARCH recursive procedure where *bids* is the set of available bids and g is the current partial solution.

the agent has no bids in the current answer g and no more of the agent's bids are in the list (PBSEARCH lines 6 and 7).

The resulting g^* found by PBSEARCH is thus the set of bids that has revenue bigger than $r(W)$ and maximizes agent i's utility. However, agent i's bids in g^* are still set to its own valuation and not to the lowest possible price. The function DISTRIBUTEPAYMENTS shown in Figure 10.2 is responsible for setting the agent's payments so that it can achieve its maximum utility u^*. If the agent has only one bid in g^* then it is simply a matter of reducing the payment of that bid by u^* from the current maximum of the agent's true valuation.

However, if the agent has more than one bid then we face the problem of how to distribute the agent's payments amongst these bids. There are many ways of distributing the payments and there does not appear to be a dominant strategy for performing this distribution. We have chosen to distribute the payments in proportion to the agent's true valuation for each set of items.

10.2. The cachedpausebid Algorithm

pausebid repeats the whole search from the scratch at every time it is invoked. We can minimize this problem by caching the result of previous searches. The cachedpausebid algorithm (shown in Figure 10.4) is our second approach to solve the bidding problem in the PAUSE auction. It utilizes a cache table where for storing some solutions to avoid doing a complete search every time. The problem is the same; the agent i has to find g_i^*. We note that g_i^* is a bidset that contains at least one bid of the agent i. Let S be a set of items for which the agent i has a valuation such that $v_i(S) \geq B(S) > 0$, let g_i^S be a bidset over S such that $r(g_i^S) \geq r(W) + \epsilon$ and

$$g_i^S = \arg\max_{g \subseteq 2^B} u_i(g), \tag{13}$$

where each g is a set of bids that covers all items and $\forall_{b \in g}\, (b \in B)$ or ($b^{\text{agent}} = i$ and $b^{\text{price}} > B(b^{\text{items}})$) and ($\exists_{b \in g} b^{\text{items}} = S$ and $b^{\text{agent}} = i$). That is, g_i^S is i's best valid bidset (contains all the items) that includes a bid from i for the set of items S. In the PAUSE auction we cannot bid for sets of items with size greater than k. So, if we have the corresponding g_i^S for each set of items S for which $v_i(S) > 0$ and $|S| \leq k$ then g_i^* is the g_i^S that maximizes the agent's utility. That is

$$g_i^* = \arg \max_{\{S \,|\, v_i(S) > 0 \wedge |S| \leq k\}} u_i(g_i^S). \tag{14}$$

Each agent i implements a hash table $C\text{-}Table$ such that $C\text{-}Table[S] = g^S$ for all S with $v_i(S) \geq B(S) > 0$. We can then find g^* by searching for the g^S, stored in $C\text{-}Table[S]$, that maximizes the agent's utility, considering only the set of items S with $|S| \leq k$. The

```
CACHEDPAUSEBID($i, k, k\text{-}changed$)
for each $S$ in $C\text{-}Table$
    do if $v_i(S) < B(S)$
        then remove $S$ from $C\text{-}Table$
        else if $k\text{-}changed$ and $|S| = k$
            then $B' \leftarrow B' +$ new Bid$(i, S, v_i(S))$
$g^* \leftarrow \emptyset$
$u^* \leftarrow u_i(W)$
for each $S$ with $|S| \leq k$ in $C\text{-}Table$
    do $\bar{S} \leftarrow Items - S$
       $g^S \leftarrow C\text{-}Table[S]$                 $\triangleright$ Global variable
       $min\text{-}payment \leftarrow \max(r(W) + \epsilon, \sum_{b \in g^S} B(b^{\text{items}}))$
       $u^S \leftarrow r(g^S) - min\text{-}payment$ $\triangleright$ Global variable
       if $(k\text{-}changed$ and $|S| = k) or (\exists_{b \in B'} b^{\text{items}} \subseteq \bar{S}$ and $b^{\text{agent}} \neq i)$
         then $B'' \leftarrow \{b \in B' | b^{\text{items}} \subseteq \bar{S}\}$
              $bids \leftarrow B'' + \{b \in B | b^{\text{items}} \subseteq \bar{S}$ and $b \notin B''\}$
              for $b \in bids$
                  do if $v_i(b^{\text{items}}) > b^{\text{price}}$
                      then $b^{\text{agent}} \leftarrow i$
                           $b^{\text{price}} \leftarrow v_i(b^{\text{items}})$
              if $k\text{-}changed$ and $|S| = k$
                then $n \leftarrow |\, bids\, |$
                     $u^S \leftarrow 0$
                else $n \leftarrow |B''|$
              $g \leftarrow \emptyset +$ new Bid$(S, i, v_i(S))$
              CPBSEARCH$(bids, g, n)$
              $C\text{-}Table[S] \leftarrow g^S$
       if $u^S > u^*$ and $r(g^S) \geq r(W) + \epsilon$
         then $g^* \leftarrow$ DISTRIBUTEPAYMENTS$(i, g^S)$
              $u^* \leftarrow u_i(g^*)$
         else if $u^S \leq 0$ and $v_i(S) < B(S)$
                then remove $S$ from $C\text{-}Table$
return $g^*$
```

FIGURE 10.4. The CACHEDPAUSEBID algorithm that implements a caching based search to find a bidset that maximizes the utility for the agent i. k is the current stage of the auction (for $k \geq 2$), and $k\text{-}changed$ is a boolean that is true right after the auction moved to the next stage.

remaining problem is maintaining the $C\text{-}Table$ updated and avoiding the search for every g^S every time. CACHEDPAUSEBID deals with this and other details.

Let B' be the set of bids that contains the new best bids, that is, B' contains the bids recently added to B and the bids that have changed price (always higher), bidder, or both and were already in B. Let $\bar{S} = Items - S$ be the complement of S (the set of items not included in S). CACHEDPAUSEBID takes three parameters: i the agent, k the current stage of the auction, and $k\text{-}changed$ a boolean that is true right after the auction moved to the next stage. Initially $C\text{-}Table$ has one row or entry for each set S for which $v_i(S) > 0$. We start by eliminating, from $C\text{-}Table$ (line 3), the entries corresponding to each set S for which $v_i(S) < B(S)$. Then, in the case that $k\text{-}changed$ is true, for each set S with $|S| = k$, we add to B' a bid for that set with value equal to $v_i(S)$ and bidder agent i (line 5); this is a bid that the agent is now allowed to consider. We then search for g^* amongst the g^S stored in $C\text{-}Table$, for which we only need to consider the sets with $|S| \leq k$ (line 8). But how do we know that the g^S in $C\text{-}Table[S]$ is still the best solution for S? There are only two situations when we are not sure about that and we need to do a search to update $C\text{-}Table[S]$; i) when $k\text{-}changed$ is true and $|S| \leq k$, since there was no g^S stored in $C\text{-}Table$ for this S; ii) when there exists at least one bid in B' for the set of items $\bar{S}$ or a subset of it submitted by an agent different than i, since it is probable that this new bid can produce a solution better than the one stored in $C\text{-}Table[S]$.

We handle the two cases mentioned above in lines 13 to 26 of CACHEDPAUSEBID. In both of these cases, since g^S must contain a bid for S we need to find a bidset that covers the missing items, that is $\bar{S}$. Thus, our search space consists of all the bids on B for the set of items $\bar{S}$ or for a subset of it. We build the list $bids$ that contains only those bids. However, we put the bids from B' at the beginning of $bids$ (lines 14 and 15 of CACHEDPAUSEBID) since they are the ones that have changed. Then, we replace the bids in $bids$ that have a price lower than the valuation the agent i has for those same items with a bid from agent i for those items and value equal to the agent's valuation (lines 16–19).

The recursive procedure CPBSEARCH, called in line 25 of CACHEDPAUSEBID and shown in Figure 10.5, is the one that finds the new g^S. CPBSEARCH is a slightly modified version of our branch and bound search implemented in PBSEARCH. The first modification is that it has a third parameter n that indicates how deep on the list $bids$ we want to search, since it stops searching when n is less or equal to zero and not only when the list $bids$ is

```
CPBSEARCH($bids, g, n$)
 1  if $bids = \emptyset$ or $n \leq 0$ then return
 2  $b \leftarrow$ first($bids$)
 3  $bids \leftarrow bids - b$
 4  $g \leftarrow g + b$
 5  $\bar{I}_g \leftarrow$ items not in $g$
 6  if $g$ includes all items
 7     then $min\text{-}payment \leftarrow \max(0, r(W) + \epsilon - (r(g) - r_i(g)), \sum_{b \in g \mid b^{\text{agent}} = i} B(b^{\text{items}}))$
 8          $max\text{-}utility \leftarrow v_i(g) - min\text{-}payment$
 9          if $r(g) > r(W)$ and $max\text{-}utility \geq u^S$
10             then $g^S \leftarrow g$
11                  $u^S \leftarrow max\text{-}utility$
12          CPBSEARCH($bids, g - b, n - 1$) $\triangleright$ $b$ is Out
13     else $max\text{-}revenue \leftarrow r(g) + \max(h(\bar{I}_g), h_i(\bar{I}_g))$
14          if $max\text{-}revenue \leq r(W)$
15             then CPBSEARCH($bids, g - b, n - 1$) $\triangleright$ $b$ is Out
16          elseif $b^{\text{agent}} \neq i$
17             then $min\text{-}payment \leftarrow (r(W) + \epsilon) - (r(g) - r_i(g)) - h(\bar{I}_g)$
18                  $max\text{-}utility \leftarrow v_i(g) - min\text{-}payment$
19                  if $max\text{-}utility > u^S$
20                     then CPBSEARCH($\{x \in bids \mid x^{\text{items}} \cap b^{\text{items}} = \emptyset\}, g, n + 1$) $\triangleright$ $b$ is In
21                  CPBSEARCH($bids, g - b, n - 1$) $\triangleright$ $b$ is Out
22          else
23                  CPBSEARCH($\{x \in bids \mid x^{\text{items}} \cap b^{\text{items}} = \emptyset\}, g, n + 1$) $\triangleright$ $b$ is In
24                  CPBSEARCH($bids, g - b, n - 1$) $\triangleright$ $b$ is Out
25  return
```

FIGURE 10.5. The CPBSEARCH recursive procedure where *bids* is the set of available bids, g is the current partial solution and n is a value that indicates how deep in the list *bids* the algorithm has to search.

empty (line 1). Each time that there is a recursive call of CPBSEARCH n is decreased by one when a bid from *bids* is discarded or out (lines 12, 15, 21, and 24) and n remains the same otherwise (lines 20 and 23). We set the value of n before calling CPBSEARCH, to be the size of the list bids (CACHEDPAUSEBID line 21) in case i), since we want CPBSEARCH to search over all *bids*; and we set n to be the number of bids from B' included in bids (CACHEDPAUSEBID line 23) in case ii), since we know that only the first n bids in *bids* changed and can affect our current g^S.

Another difference with PBSEARCH is that the bound in CPBSEARCH is u^S which we set to be 0 (CACHEDPAUSEBID line 22) when in case i) and $r(g^S) - min\text{-}payment$

(CACHEDPAUSEBID line 12) when in case ii). We call CPBSEARCH with g already containing a bid for S. After CPBSEARCH is executed we are sure that we have the right g^S, so we store it in the corresponding $C\text{-}\mathit{Table}[S]$ (CACHEDPAUSEBID line 26).

When we reach line 27 in CACHEDPAUSEBID, we are sure that we have the right g^S. However, agent i's bids in g^S are still set to its own valuation and not to the lowest possible price. If u^S is greater than the current u^*, as in PAUSEBID, we have chosen to distribute the payments in proportion to the agent's true valuation for each set of items (using the function DISTRIBUTEPAYMENTS) to achieve maximum utility. Otherwise, when $u^S \leq 0$ and the valuation that the agent i has for the set of items S is lower than the current value of the bid in B for the same set of items, we remove the corresponding $C\text{-}\mathit{Table}[S]$ since we know that is not worthwhile to keep it in the cache table (CACHEDPAUSEBID line 31).

10.3. Test and Comparison: Myopic-Optimal Algorithms

We have implemented both algorithms and performed a series of experiments in order to determine how their solution compares to the revenue-maximizing solution and how their times compare with each other. In order to do our tests we had to generate value functions for the agents. The algorithm we used is shown in Figure 10.6. The type of valuations it generates correspond to domains where a set of agents must perform a set of tasks but there are cost savings for particular agents if they can bundle together certain subsets of tasks. For example, imagine a set of robots which must pick up and deliver items to different locations. Since each robot is at a different location and has different abilities, each one will have different preferences over how to bundle. Their costs for the item bundles are subadditive, which means that their preferences are superadditive.

We fixed the number of agents to be 5. We experimented with different number of items m, namely from 2 to 10. We created a dataset consisting of 900 different sets of agents (set of valuations), each one representing a different auction; 100 auctions for each number of items. In the dataset, each bidder has different private valuations for approximately $2m$ set of items. For these experiments, the minimum increment allowed in the auction was set

```
GENERATEVALUES($i$, $items$)
1  for $x \in items$
2      do $v_i(x) = $ EXPD$(.01)$
3  for $n \leftarrow 1 \ldots (num\text{-}bids - items)$
4      do $s_1, s_2 \leftarrow$ Two random sets of items with values.
5          $v_i(s_1 \cup s_2) = v_i(s_1) + v_i(s_2) + $ EXPD$(.01)$
```

FIGURE 10.6. Algorithm for the generation of random value functions. EXPD(x) returns a random number taken from an exponential distribution with mean $1/x$.

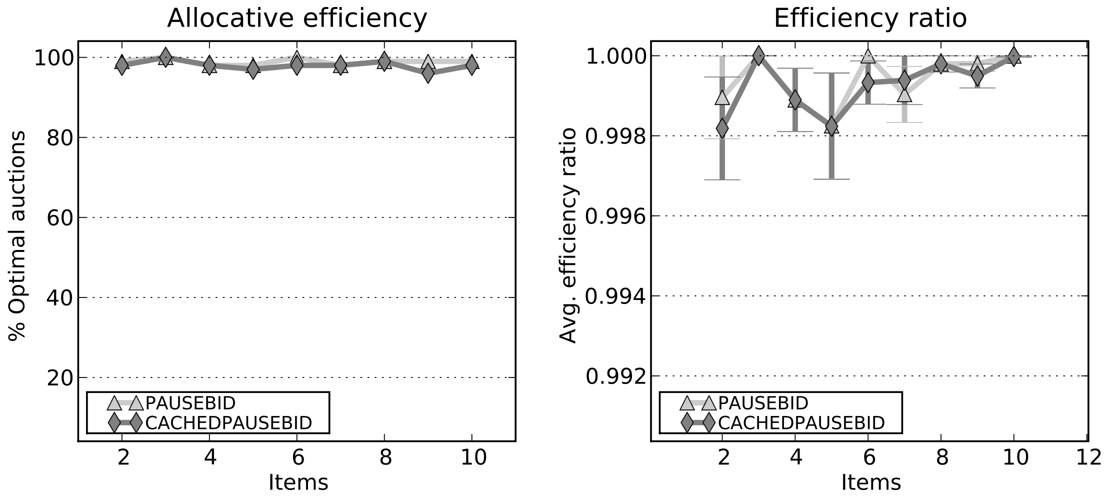


FIGURE 10.7. Efficiency of the myopic-optimal algorithms as a function of the number of items in the auction. The plot on the left shows the percentage of auctions that arrive to the *optimal allocation*, the social welfare maximizing solution. The plot on the right shows the average *efficiency ratio* (8) of the auctions.

to 1, that is $\epsilon = 1$. We use the metrics defined in Section 9.1. We calculated the optimal solution and the social welfare ($r*$) using CASS [**11**]. That is, for each agent i and each set of items S for which $v_i(S) > 0$ we generated a bid. This set of bids was fed to CASS which implements a centralized winner determination algorithm to find the solution which maximizes revenue, which, in this case, is also the social welfare maximizing solution.

The first question we have was, does the auction allocate items to the bidders who value them most highly? So the first experiment we performed simply ensured the proper functioning of our algorithms by analyzing the allocative *efficiency*. Thus, first we calculated the percentage of auctions in which the solution was the social welfare maximizing solution. When analyzing the *efficiency*, we realized that they do not always find the same distribution of items as the optimal solution (as shown in the corresponding plot in Figure 10.7). The cases where our algorithms failed to arrive at the distribution of the social welfare maximizing solution, which in this case is also the revenue-maximizing solution, are those where there was a large gap between the first and second valuation for a set (or sets) of items. If the revenue-maximizing solution contains the bid (or bids) using these higher valuation then it is impossible for the PAUSE auction to find this solution because that bid (those bids) is never placed. For example, if agent i has $v_i(1) = 1000$ and the second highest valuation for (1) is only 10 then i only needs to place a bid of 11 in order to win that item. If the optimal solution requires that 1 be sold for 1000 then that solution will never be found because that bid will never be placed. However, in these experiments, only few auctions finished with an allocation different from the optimal one; up to 2% PAUSEBID and up to 4% for CACHEDPAUSEBID. Another important observation is that the our myopic-optimal algorithms do not always arrive to the same solution. We have to remember that the PAUSE auction is multi-stages and in each stage bidders have many chances to formulate bids. This small difference is the result of the no determinism implicit when ordering the $bids$ list to create the search tree, which make them search in different order.

We also found that average *efficiency ratio* increases non-monotonically as the number of items increases for both algorithms (as shown in the corresponding plot in Figure 10.7). For 2 items this average is higher than 0.998 and it reaches 1 for 10 items. This means that the social welfare of the solutions that did not converged to the optimal solution (the social welfare maximizing solution) where very close to it. On average the allocations found by our algorithms are highly efficient, for both of the algorithms.

We know that the revenue generated by the PAUSE auction is generally lower than the revenue of the revenue-maximizing solution, when assuming truthful bidding and first price payments, but how much lower? To answer this question we analyzed the average *revenue ratio* (9). We found that the average *revenue ratio* of our algorithms increases

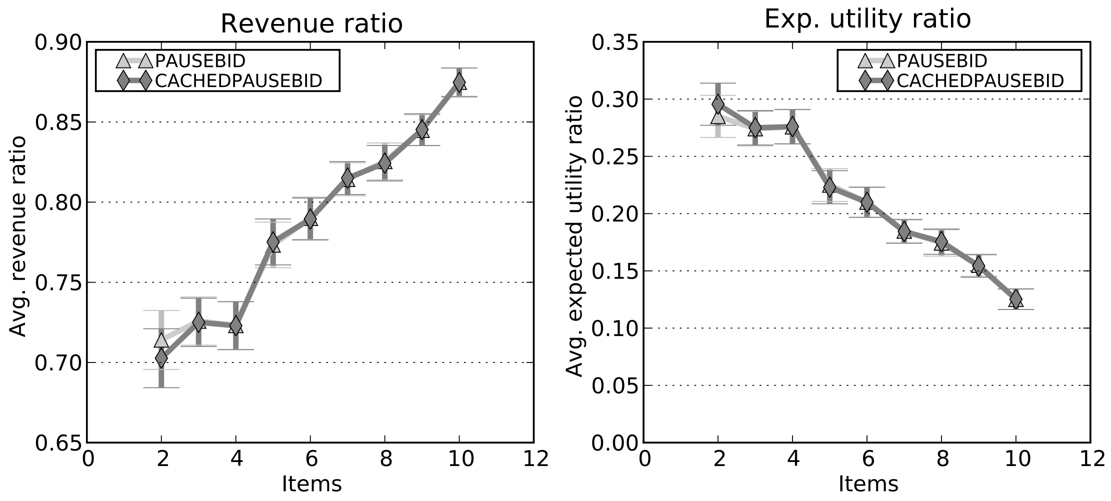


FIGURE 10.8. Revenue and utility obtained by the myopic-optimal algorithms as a function of the number of items in the auction. The plot on the left shows the average *revenue ratio* (9). The plot on the right shows the average bidders' *expected utility ratio* (10).

as the number of items increases, as shown in Figure 10.8. Both algorithms generate, on average, the same *revenue ratio*. In the case of 2 items, the average *revenue ratio* is 0.702 for CACHEDPAUSEBID and 0.714 for PAUSEBID, while in the other extreme (10 items), both algorithms have on average a *revenue ratio* of 0.874. Thus, this shows that on average the revenue generated by the PAUSE auction when using our algorithms is between 13% and 28% less than the optimal revenue.

To analyze the bidders' expected utility we calculated the average *bidders' expected utility ratio*. As shown in the corresponding plot of Figure 10.8, the average *bidders' expected utility ratio* decreases as a function of the number of items in the auction. The curve of the plot is inverse of that of the average *revenue ratio*, and in general, is similar for both algorithms. For the case of 2 items, the average *bidders' expected utility ratio* is 0.284 for PAUSEBID and 0.295 for CACHEDPAUSEBID. For the case of 10 items, the average *bidders' expected utility ratio* is the same for both algorithms. about 0.125.

Finally, we analyzed the running time of our algorithms. The experiments where carried on in an SGI Altix 4700 with 128 Itanium Cores @ 1.6 GHz/8MB Cache and 256 GB of RAM (shared-memory system). As shown in Figure 10.9, CACHEDPAUSEBID is faster

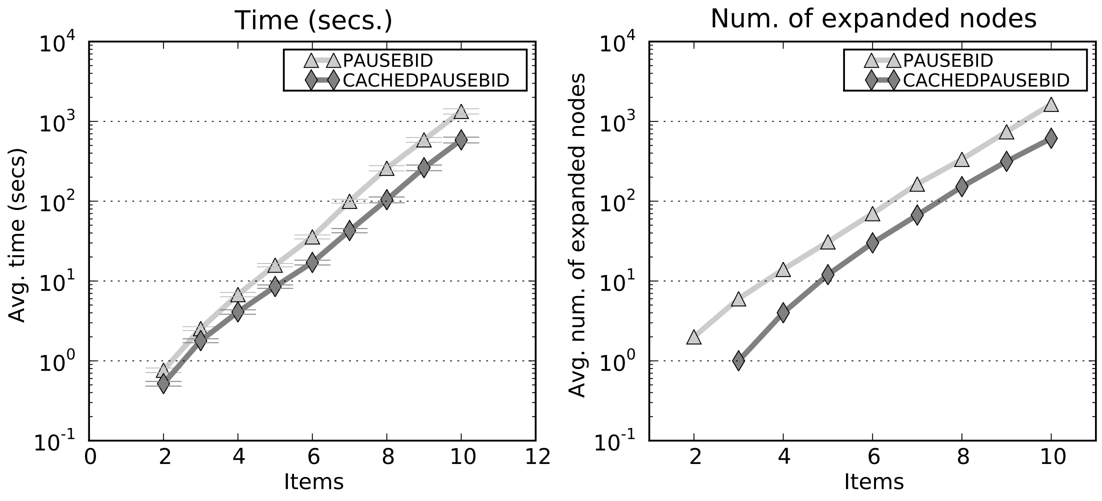


FIGURE 10.9. Computational efficiency and scalability of the myopic-optimal algorithms as a function of the number of items in the auction. The plot on the left shows the average *running time* per auction. The plot on the right shows the average *number of expanded nodes* per auction.

than PAUSEBID, the difference in execution speed is even more clear as the number of items increases. As expected since this is an NP-Hard problem, the running time grows exponentially with the number of items. The average running time of CACHEDPAUSEBID is about half the running time of PAUSEBID.

The scalability of our algorithms (or computational efficiency) can also be determined by counting the number of nodes expanded in the search tree. For this we count the number of times that PBSEARCH gets invoked for each time that PAUSEBID is called and the number of times that CPBSEARCH gets invoked for each time that CACHEDPAUSEBID, respectively for each of our algorithms. As expected since this is an NP-Hard problem, the number of expanded nodes does grow exponentially with the number of items (as shown in the corresponding plot in Figure 10.9). However, we found that CACHEDPAUSEBID outperforms PAUSEBID, since it expands on average less than half the number of nodes. For example, the average number of nodes expanded when 2 items is zero for CACHEDPAUSEBID while for PAUSEBID is 2; and in the other extreme (10 items) CACHEDPAUSEBID expands on average only 633 nodes while PAUSEBID expands on average 1672 nodes, a difference of more than 1000 nodes. Although the number of nodes expanded by our algorithms increases as function of the number of items, the actual number of nodes is a much smaller than the worst-case scenario of n^n where n is the number of items. For example, for 10 items we expand slightly more than 10^3 nodes for the case of PAUSEBID and less than that for the case of CACHEDPAUSEBID which are much smaller numbers than 10^{10}. Notice also that our value generation algorithm (Figure 10.6) generates a number of bids that is exponential on the number of items, as might be expected in many situations. As such, these results do not support the conclusion that time grows exponentially with the number of items when the number of bids is independent of the number of items. We would expect that both algorithms will grow exponentially as a function the number of *bids*, but stay roughly constant as the number of items grows.

10.4. Conclusions: Myopic-Optimal Algorithms

We have presented two algorithms, PAUSEBID and CACHEDPAUSEBID, that bidder agents can use to engage in a PAUSE auction. Both algorithms implement a myopic utility maximizing strategy that is guaranteed to find the bidset that maximizes the agent's utility given the set of outstanding best bids at any given time, without considering possible future bids. Both algorithms find, most of the time, the same distribution of items as the revenue-maximizing solution. The cases where our algorithms failed to arrive at that distribution are those where there was a large gap between the first and second valuation for a set (or sets) of items. As it is an NP-Hard problem, the running time of our algorithms remains exponential but it is significantly better than a full search. PAUSEBID performs a branch and bound search completely from scratch each time it is invoked. CACHEDPAUSEBID caches partial solutions and performs a branch and bound search only on the few portions affected by the changes on the bids between consecutive times. CACHEDPAUSEBID has a better performance since it explores fewer nodes (less than half) and it is faster. As expected the revenue generated by a PAUSE auction is lower than the revenue of a revenue-maximizing solution found by a centralized winner determination algorithm. We also found that the revenue generated by our algorithms increases as function of the number of items in the auction.

Our algorithms have shown that it is feasible to implement the complex coordination constraints supported by combinatorial auctions without having to resort to a centralized winner determination algorithm. Moreover, because of the design of the PAUSE auction, the agents in the auction also have an incentive to perform the required computation. Our bidding algorithms can be used by any multiagent system that would use combinatorial auctions for coordination but would rather not implement a centralized auctioneer. However, a system using theses algorithms would have poor scalability. Thus, our next step is to develop faster algorithms that can scale for more complex problems involving a bigger number of items and bidder agents.

CHAPTER 11

APPROXIMATE BIDDING ALGORITHMS FOR A DISTRIBUTED COMBINATORIAL AUCTION

The algorithms previously presented implement a complete search over the set of current best bids B in order to find the new bidset which is myopic-optimal. Approximate bidding algorithms forgo optimality in favor of heuristics and simple local searches which deliver a solution very quickly. The approximate algorithms we have developed are based in the *greedy algorithm* introduced in Section 5.3 and the well-known *hill climbing* algorithm.

11.1. THE GREEDYPAUSEBID ALGORITHM

The GREEDYPAUSEBID algorithm (Figure 11.1) is based in the greedy strategy discussed in Section 5.3. However, it maximizes the bidder's utility, instead of maximizing the seller's revenue, under the condition that the resulting revenue has to be greater or equal than $r(W) + \epsilon$. We start by defining $my\text{-}bids$ to be the list of bids for which the agent's valuation is higher than the current best bid, as given in B. We set the value of these bids to be the agent's true valuation (but we won't necessarily be bidding true valuation, as we explain later). Similarly, we set $their\text{-}bids$ to be the rest of the bids from B. If $my\text{-}bids$ is empty, there is no bid that the agent can dominate at this time and the algorithm ends. The function SORTFORGREEDY (called in lines 12 and 16) sorts the list of bids received as first parameter by $b^{\text{price}}/|b^{\text{items}}|^c$. After $my\text{-}bids$ is sorted, we take the first bid and add it to the initial bidset g, to make sure that the solution includes the bid from $my\text{-}bids$ with the highest rank. Finally, to complete the allocation containing all the items, the agent's search list is simply the concatenation of $their\text{-}bids$ and the rest of $my\text{-}bids$ sorted again by the

```
GREEDYPAUSEBID$(i, k, c)$
 1  $my\text{-}bids \leftarrow \emptyset$
 2  $their\text{-}bids \leftarrow \emptyset$
 3  for $b \in B$
 4      do if $b^{\text{agent}} = i$ or $v_i(b^{\text{items}}) > b^{\text{price}}$
 5            then $my\text{-}bids \leftarrow my\text{-}bids$ +new Bid$(b^{\text{items}}, i, v_i(b^{\text{items}}))$
 6            else $their\text{-}bids \leftarrow their\text{-}bids + b$
 7  for $S \in$ subsets of $k$ or fewer items such that $v_i(S) > 0$ and $\neg\exists_{b \in B} b^{\text{items}} = S$
 8      do $my\text{-}bids \leftarrow my\text{-}bids$ +new Bid$(S, i, v_i(S))$
 9  $g \leftarrow \emptyset$
10  if $my\text{-}bids = \emptyset$
11     then return $g$
12  $my\text{-}bids \leftarrow$ SORTFORGREEDY$(my\text{-}bids, c)$
13  $b \leftarrow$ first$(my\text{-}bids)$
14  $g \leftarrow g + b$
15  $bids \leftarrow their\text{-}bids$ +rest$(my\text{-}bids)$
16  $bids \leftarrow$ SORTFORGREEDY$(bids, c)$
17  while $bids \neq \emptyset$
18      do $b \leftarrow$ first$(bids)$
19         $bids \leftarrow$ rest$(bids)$
20         $I_g \leftarrow$ items in $g$
21         if $b^{\text{items}} \cap I_g = \emptyset$
22            then $g \leftarrow g + b$
23                 $bids \leftarrow \{x \in bids \mid x^{\text{items}} \cap b^{\text{items}} = \emptyset\}$
24  if $r(g) > r(W) + \epsilon$
25     then $g \leftarrow$ DISTRIBUTEPAYMENTS$(i, g)$
26          if $u_i(g) \leq u_i(W)$
27             then $g \leftarrow \emptyset$
28     else $g \leftarrow \emptyset$
29  return $g$
```

FIGURE 11.1. Our GREEDYPAUSEBID algorithm returns an empty bidset if the solution it finds does not improve the utility of bidder i. c is the bids sorting factor and k is the current stage of the auction, for $k \geq 2$.

same criteria (lines 16 and 15 respectively). After we finish walking down the *bids* list, we have an allocation g. However, agent i's bids in g are still set to his own valuation and not to the lowest possible price. If $r(g) \leq r(W) + \epsilon$, then the algorithm ends. Otherwise (when $r(g) > r(W) + \epsilon$), we call the procedure DISTRIBUTEPAYMENTS with g as parameter (line 25), the same method used by PAUSEBID. After distributing the payments of g

```
HILLCLIMBING$(i, g, bids)$
1  $remain\text{-}bids \leftarrow \{x \in bids \mid x \notin g\}$
2  for $b \in remain\text{-}bids$
3      do $g' \leftarrow \{x \in g \mid x^{\text{items}} \cap b^{\text{items}} = \emptyset\}$
4         $g' \leftarrow$ CONSISTENTBIDS$(g', remain\text{-}bids)$
5         if $g' = \emptyset$
6            then continue
7         if $u_i(g') > u_i(g)$
8            then $g \leftarrow g'$
9                 return HILLCLIMBING$(i, g, bids)$
10 return $g$
```

FIGURE 11.2. The HILLCLIMBING is a local search algorithm that explores the neighborhood's bidsets or allocations until the point where there are no more bidsets that produce higher utility for the agent i. $bids$ is the bid list that contains $my\text{-}bids + their\text{-}bids$. g is the initial greedy solution as found by GREEDYPAUSEBID.

the algorithm ends by returning g if the utility that the agent receives from g is greater than that it gets from W, otherwise it returns an empty bidset.

11.2. THE GREEDYPAUSEBID+HILL ALGORITHM

A simple extension to the greedy approach consists of using a local search algorithm that continuously updates the initial allocation found by the greedy algorithm [**12**], thus locally searching in the remaining bids to improve the solution. We implement this idea in the GREEDYPAUSEBID+HILL algorithm. This algorithm starts with the solution provided by GREEDYPAUSEBID and then explores the neighborhood of that solution, using a simple *hill climbing*, looking for allocations that generate a higher utility for the bidder. It consist of two main steps:

(1) Call GREEDYPAUSEPID algorithm with appropriate input and $c = 0.5$.
(2) If the solution g returned by GREEDYPAUSEBID is not empty then call the procedure HILLCLIMBING with $bids$ = $my\text{-}bids$ + $their\text{-}bids$ sorted by c.

The HILLCLIMBING is a local search algorithm that explores the neighborhood's bidsets or allocations until the point where there are no more bidsets that produce higher utility for the agent i. The function CONSISTENTBIDS (called in line 4) finds consistent bids for the bidset g' by walking down the list $remain\text{-}bids$. g' consists of all the bids in g that are not conflicted with b (line 3), so there will be free items to allocate after the insertion of b. The function CONSISTENTBIDS tries to insert other bids to complete the bidset since a complete bidset must contain all the items. CONSISTENTBIDS returns and empty bidset when it cannot complete a feasible bidset.

11.3. Test and Comparison: Myopic-optimal and Approximate Algorithms

Our approximate bidding algorithms allow an agent to place good, but not necessarily myopic-optimal, bids in a PAUSE auction. But the use of these new strategies raises several questions. Does a system of approximate bidders arrive at a different solution than the optimal biding agents? Do the agents lose utility by switching to these faster algorithms? Is there any difference in the revenue generated by a system of approximate bidders compared with the revenue generated by a system of optimal bidders? How faster are our new algorithms as compared to PAUSEBID and CACHEDPAUSEBID?

We carried out an experimental simulation with the same dataset used in the experiment presented in Section 10.3. We calculated the metrics presented in Section 9.1 to compare the solutions found by both optimal and approximate algorithms to the social welfare maximizing solution. As in the previous chapter, in these experiments we set ϵ (the minimum bid increment) to be 1 and the whole population of bidders is homogeneous: all bidders use the same bidding strategy.

As shown in Figure 11.3, we found that the running time required to clear an auction grows exponentially as a function of the number of items when using PAUSEBID, but remains linear when using any of the other algorithms. GREEDYPAUSEBID is up to 99 times faster than PAUSEBID and 10 times faster than GREEDYPAUSEBID+HILL (for the case of 10 items). Something important to notice is that, different from the myopic-optimal algorithms

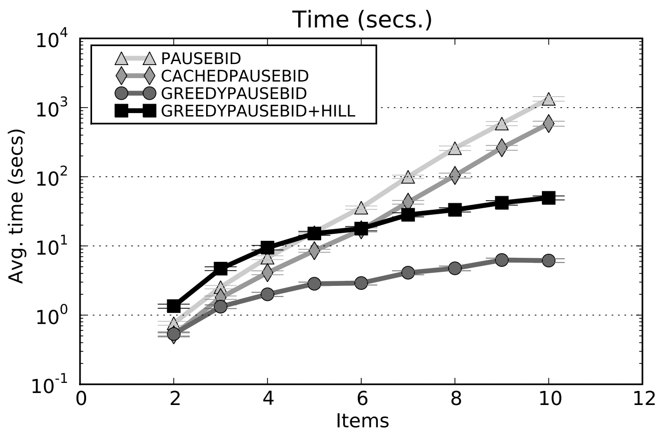


FIGURE 11.3. Computational efficiency of the myopic-optimal and approximate algorithms. The plot shows the average *running time* per auction as a function of the number of items in the auction.

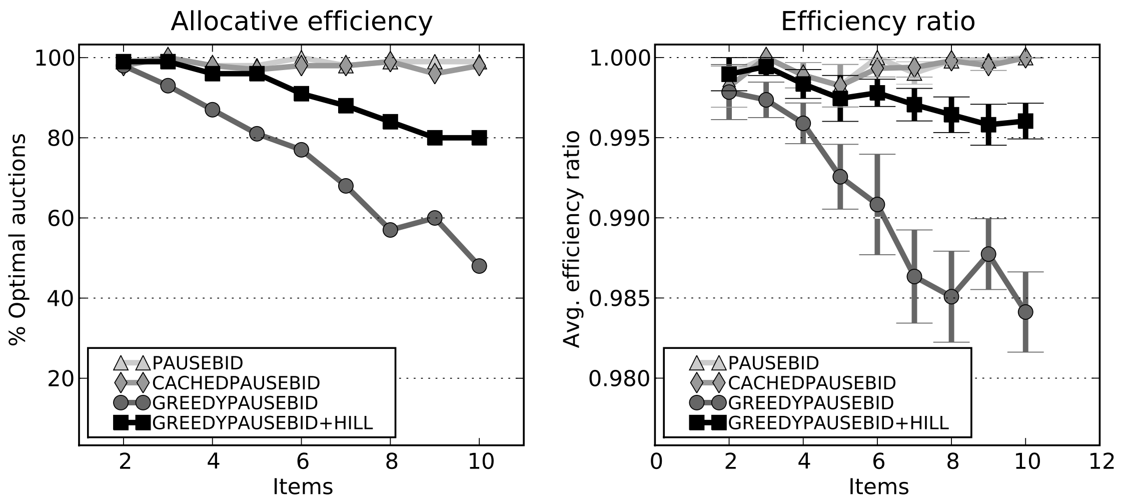


FIGURE 11.4. Allocative Efficiency of the myopic-optimal and approximate algorithms. The plot on the left shows the percentage of auctions that arrive to the *optimal allocation*, the social welfare maximizing solution. The plot on the right shows the average *efficiency ratio* (8) of the auctions.

which curve grows exponential as a function of then number of items, the approximate algorithms curve remains linear.

In the previous chapter, we showed that the myopic-optimal algorithms converge to the same distribution of items to bidders as the social welfare maximizing solution more than 95 percent of the time. As show in Figure 11.4, the percentage of auctions that end up with an optimal solution is lower for all the approximate algorithms. When using GREEDY-PAUSEBID, the percentage of auctions that convergence to the optimal solution, drops monotonically as function of the number of items, going from 98% with 2 items down to 48% with 10 items. The case of the GREEDYPAUSEBID+HILL algorithm is much better, although not as efficient as the case of the myopic-optimal, the percentage of auctions that convergence to the optimal solution drops as function of the number of items, going from 99% with 2 items down to 80% with 10 items. This experiment shows that a system with approximate bidders does arrive at a different solutions than a system with myopic-optimal bidders.

As expected the average *efficiency ratio* (8) decreases as a function of the number of items too (as shown in the corresponding plot in Figure 11.4). Although the average efficiency ratio of GREEDYPAUSEBID+HILL decreases as a function of the number of items, it is maintained above 0.995 when 10 items. In terms of *efficiency*, GREEDYPAUSEBID+HILL is much better than GREEDYPAUSEBID, but it is not better than the myopic-optimal algorithms. However, the (allocative) efficiency of the solutions found by all our algorithms is high.

We calculate the revenue ratio of the solutions found by our algorithms to the optimal revenue. As expected, since PAUSE is an increasing price combinatorial auction, the winners end up paying less than in a centralized one-shot first price sealed-bid combinatorial auction. However, the *revenue* increases as a function of the number of items. For 10 items, the revenue generated by the myopic-optimal algorithms is about 4% higher than the one generated by GREEDYPAUSEBID+HILL and 6% higher than that of GREEDYPAUSEBID.

The low payments obtained in the PAUSE auction are translated in higher bidder's utility. In the corresponding *utility* plot in Figure 11.5 we show the *bidders' expected utility ratio*. The plot shows that the approximate bidding algorithms, on average, provide higher bidder's utility than the myopic-optimal ones (up to 0.05 higher). It might seem counter-intuitive that bidders receive a higher utility when using an approximate algorithm than when using an algorithm that guarantees a bid that gives them the highest possible

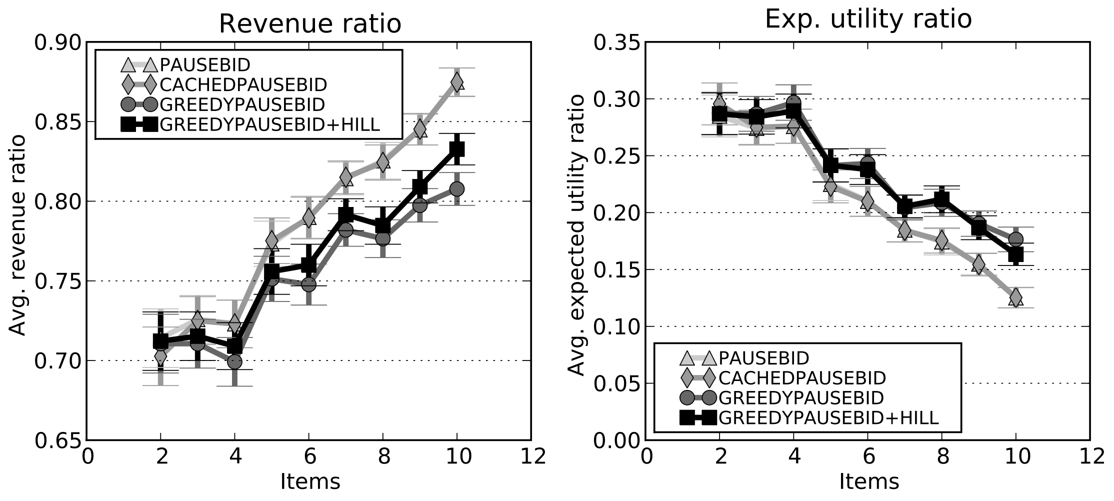


FIGURE 11.5. Revenue and utility obtained by the myopic-optimal and approximate algorithms as a function of the number of items in the auction. The plot on the left shows the average *revenue ratio* (9). The plot on the right shows the average bidders' *expected utility ratio* (10).

utility. However, we must consider that the agents are engaged in a PAUSE auction and that their utility depends, not on the intermediate bids placed by the agents, but only on the result of the auction. It seems that having all agents place less than optimal bids makes the competition for the items end sooner, dominated bidders give up, and the result is that they all, on average, get higher utility.

11.4. Conclusions: Myopic-optimal and Approximate Algorithms

Our tests have shown that our heuristic greedy algorithms offer higher bidders utility and high speed as a second incentive. We have also shown that the allocations obtained with our algorithms are very efficient, although the ones from the myopic-optimal are in average better. In general, the revenue generated by a PAUSE auction is lower than the optimal, and the greedy approximate algorithms generate about 5% less revenue than the other algorithms; nevertheless, we found that it increases as the number of items in the auction increases. However this is translated into more utility for the bidders. Thus, the bidders's benefits provided by PAUSE will attract more bidders, which would result in more benefits for the seller in addition of the savings from eliminating the tax paid to the third party. From these experiments we could say that the approximate algorithms are the best option for the bidders. However, we need other type of experiments to make that conclusion. The next section presents an analysis to find out if this is true or not.

CHAPTER 12

A GAME THEORETICAL ANALYSIS OF BIDDING STRATEGIES FOR A DISTRIBUTED COMBINATORIAL AUCTION

Our algorithms show that it is feasible to implement distributed combinatorial auctions without having to resort to a central auctioneer (Section 11.3). Furthermore, the allocations obtained with the PAUSE auction, when using our algorithms, are very efficient. Thus, a rational bidder agent that aims to maximize her utility would prefer joining to a PAUSE auction over joining to a centralized auction. First, because she does not have to reveal her private valuations, and second, because of the opportunity of obtaining the items at a lower price that in a one-shot first price combinatorial auction; although she has to perform some computational work. However, since the bidder does not know neither the other bidders' bidding strategies nor their valuations functions, she faces the predicament of choosing a bidding strategy. Which bidding algorithm should she use? Is it better for her to spend more time in computation and bid optimally, or save computational time and bid with some approximate strategy? Should her decision depend on the algorithms the other bidders in the auction are using? In this section we present an experimental game theoretical analysis to answer this questions.

12.1. EXPERIMENTAL MODEL

Although we know that there might be a number of possible bidding algorithms or strategies, in this experiment we consider only three of our algorithms, which represent the extremes of the possible bidding algorithms. In one hand, we have one of our myopic-optimal algorithms, PAUSEBID, representing the extreme where a bidder uses all her resources to bid with total optimality in each round, which we consider it is the best the bidders can do; the only strategy that could be better than this would be a look-ahead algorithm, which would be very difficult to design and implement. In the other hand, we have the GREEDYPAUSEBID algorithm, representing the extreme where a bidder follows a greedy strategy that is very simple and fast but is not optimal; we consider this the simplest approach which in theory has very good results. We also have the GREEDYPAUSEBID+HILL algorithm, an strategy that combines the greedy approach enhanced with local search, this approach would be in between the other two approaches.

In Section 11.3 we showed that the approximate bidding algorithms, on average, provide higher expected utility for all the bidders. However, a rational selfish agent does not care about others' utility, she cares only about her own utility. Thus, the predicament the bidder faces is in choosing the bidding strategy. What strategy should I use? What if all the other bidders are using PAUSEBID, should I still join? What if they are all using GREEDYPAUSEBID? We modeled the problem as a *strategic game*. In the game *Me* is the bidder deciding to join to the auction, and *Others* represent the rest of the bidders in the auction. Their possible strategies are the 3 bidding algorithms: PAUSEBID, GREEDYPAUSEBID, and GREEDYPAUSEBID+HILL. Notice that, for simplifications purposes, we did not include CHACHEDPAUSEBID in the game because it generates about the same amount of bidders' expected utility than PAUSEBID. In this game, we assume that each bidder chooses her bidding strategy once and for all, and does not change it during the time the auction last. We used the same data set used in Sections 10.3 and 11.3. That is, we fixed the number of agents to be 5. We experimented with different number of items m, namely from 2 to 10. The dataset consists of 900 different sets of agents (set of valuations), each one representing a different auction; 100 auctions for each number of items. In the dataset, each bidder has different private valuations for approximately $2m$ set of items.

		Others		
		PB	GP	GH
	PB	78, 78	86, 84	84, **87**
Me	GP	85, 80	86, 86	**87**, **88**
	GH	**88, 79**	**96, 85**	**87**, **87**

FIGURE 12.1. Payoff matrix. The payoffs correspond to the average utility obtained from all the auctions. Each auction has 5 bidders and each one gets to play the *Me* role. The *others* payoff is the average of the average sum of the utilities of the other 4 bidders. Here we use PB for PAUSEBID, GP GREEDYPAUSEBID, and GH for GREEDYPAUSEBID+HILL.

We obtained their ordinal preferences, shown in the payoff matrix in Figure 12.1, by an experimental simulation in which we let each bidder play the *Me* role for each pair of bidding strategies, *Others* payoff is average of the sum of the other bidders utility divided by 4 (the number of the other bidders). Notice that in cases where *Me* and *Others* have homogeneous strategies, we only need to run the auction once, since all possible combinations will lead to the exact same final allocation. The payoff matrix shows that the social welfare solution is the case when bidder uses GREEDYPAUSEBID+HILL and the others use GREEDYPAUSEBID. However, if everybody thinks as *Me*, everybody would be using GREEDYPAUSEBID+HILL. Thus, the game has two pure Nash equilibria, GP/GH (*Me* GREEDYPAUSEBID, *Others* GREEDYPAUSEBID+HILL) and GH/GH (*Me* GREEDYPAUSEBID+HILL, *Others* GREEDYPAUSEBID+HILL). The payoffs for *Me* in the outcomes of GP/GH and GH/GH are the same, which makes GH/GH a *non-strict Nash equilibrium*; thus the game has an equilibria in which a player is indifferent between her equilibrium action and some other action, given the other players' actions. That is, if *Others* are playing GH then *Me* will be ambivalent between GP and GH, so she might choose GP since that uses a bit less computation.

However, the most important point, confirmed by this game model, is that all the bidders get more utility by moving away from PAUSEBID. That is, the heuristic greedy algorithms strictly dominate PAUSEBID.

12.2. Conclusions: Game-Theoretical Analysis

Based on the results of our experiments, we believe that the approximate bidding algorithms are a realistic tool for the development of large-scale distributed combinatorial auctions, with no need for a centralized auctioneer. Furthermore, they are dominant strategies over the myopic-optimal ones, and bidders have double incentive to use them: they are faster and they provide the bidders higher utility. Specifically, as shown by our game theoretical model, the case where everybody bids using GREEDYPAUSEBID+HILL algorithm is a non-strict Nash equilibrium. Although these strategies reduce total revenue and thus, the seller's utility; even the seller can benefit from them, since she does not have to pay a commission for the auctioneer's job (a third party agency or the maintenance of system) and they might attract more bidders. We envision a future where complex sourcing problems are solved by millions of automated agents bidding and negotiating in distributed combinatorial auctions and even more complex negotiation networks. For our future work we are looking at how to further increase the scalability of the PAUSE auction by allowing agents to place incomplete bid sets and only communicate their bids to a subset of agents. Our goal is to develop new auctions which can scale to any number of bidders while distributing the computational cost evenly amongst them.

Now, we are interested in knowing about the performance of the PAUSE auction and our algorithms in other problem domains. In the next Chapter, we present an study for this purpose, which includes different type of problems.

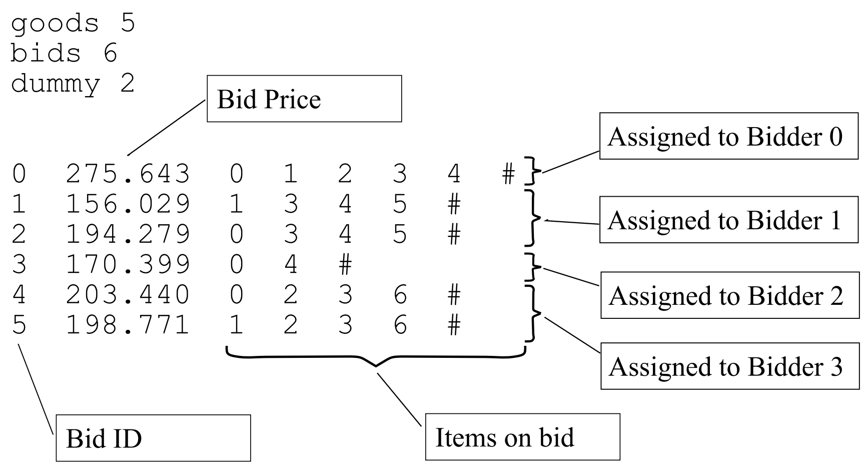
CHAPTER 13

THE ECONOMIC AND COMPUTATIONAL EFFICIENCY OF PAUSE AND OUR ALGORITHMS UNDER DIFFERENT BID DISTRIBUTIONS

Having myopic-optimal and heuristic greedy algorithms for bidding in the PAUSE auction, we were next interested in determining how long it would take for populations of agents using these algorithms to arrive at a solution, as well as quantifying the expected utility of this solution, under different problems or scenarios. Since the efficiency of a winner determination algorithm varies greatly depending on the input bid distribution we decided to use four different problem settings from the *Combinatorial Auctions Test Suite* [**24**], known as CATS.

13.1. CATS DISTRIBUTIONS

As mentioned in Chapter 6, CATS is a generator of combinatorial auction instances for the testing of winner determination algorithms. It features five distributions of instances from realistic, economically motivated domains, as well as a collection of artificial distributions that have been used in the literature. A classification by *gross hardness* of ten of this distributions is presented by [**25**]. This classification is based on an experiment that shows what distributions are harder to solve, thus the corresponding *gross hardness* corresponds to time requirements. In order to observe the PAUSE auction and our bidding algorithms in different economic scenarios we selected four of those ten CATS distributions, covering the whole hardness range.



```
goods 5
bids 6
dummy 2

0  275.643  0  1  2  3  4  #
1  156.029  1  3  4  5  #
2  194.279  0  3  4  5  #
3  170.399  0  4  #
4  203.440  0  2  3  6  #
5  198.771  1  2  3  6  #
```

FIGURE 13.1. An example of the content of a CATS file and how we use it. It contains six bids enumerated from 0 to 5; five items, from 0 to 4; and two dummies, 5 and 6.

- *Scheduling*. This distribution is motivated by the distributed job shop scheduling problem. It is the second easiest amongst all the CATS distributions. It allows XOR bids.
- *L2 (weighted random)*. An artificial distribution with no economic motivation. Based on its gross hardness, this distribution is in the middle.
- *Arbitrary*. Based on scenarios where some goods do not give rise to a notion of adjacency but regularity in complementarity relationships can still exist (e.g., physical objects like collectables, semiconductors). This distribution is the third hardest one. It allows XOR bids.
- *L3 (uniform)*. This is another artificial distribution, which is the hardest amongst all the CATS distributions.

13.1.1. Using CATS files to feed the decentralized combinatorial auction. CATS was created to test centralized algorithms that find winning bidset in combinatorial auctions. These algorithms ignore the identity of the bidders, focusing instead on the bids. Their goal is to find the allocation that maximizes revenue regardless of who placed those bids. For this reason, CATS does not identify the bidder that placed the bid, nor does it

	Bidders	Bids	Goods
for Bidders$*$	(3 to 11)	20	10
for Bids	5	(12-20)	10
for Goods	5	20	(10 to 17)

TABLE 13.1. Combination of parameters used to create the data set for each one of the CATS distributions described in the previous section. $*$The number of bidder cannot be manipulated in the L2 and L3 distributions, in these cases the number of bidders is equal to the number of bids.

provides us with the bidder's utility function, it only includes a bid *id*, the price of the bid, and the set of items in the bid. To implement a PAUSE auction we need to know every bidder's valuations. In order to use a CATS file in the PAUSE auction without changing the economic distribution, we had to assign each bid from the file bid to a different bidder, except when the bids are XOR bids. The XOR bids in the CATS file include an easily identifiable dummy item. All bids with the same dummy item are assigned to one bidder. In addition, CATS does not offer a direct way to control the number of bidders. The only way to do this is by generating a file and then check how many bidders can be obtained. Figure 13.1 shows an example of a CATS file and how we use it.

13.2. EXPERIMENTAL SETTINGS

Our experiments consist of combinations of the parameters mentioned in previous subsections. We created a dataset for each one of the CATS distribution mentioned above. Each data set contains 100 bid files or auctions for some combinations of number of bidders, bids, and goods. Table Table 13.1 shows the combinations used in the experiments. To analyze the effect of one variable we fix the value of the other two.

In this experiment we try our four bidding algorithms (strategies): PAUSEBID, CACHEDPAUSEBID, GREEDYPAUSEBID, and GREEDYPAUSEBID+HILL. For each bidding strategy we carried out a PAUSE auction over each distribution and calculate the metrics described in Section 9.1. It is important to remember that in each auction all the bidders use the same bidding algorithm. The experiments where carried on in an SGI Altix 4700 with 128 Itanium Cores @ 1.6 GHz/8MB Cache and 256 GB of RAM (shared-memory system). We

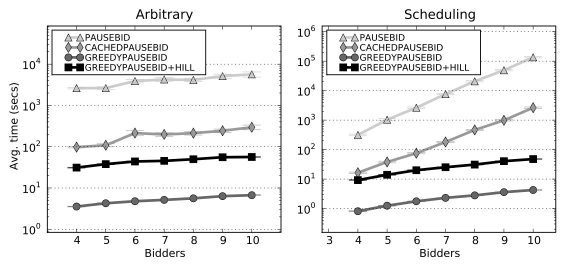


FIGURE 13.2. *Average time* as a function of the number of *bidders* in the auction. The number of goods and bids is fixed at 10 and 20 respectively.

calculated the revenue-maximizing solution ($r*$) using CASS [**11**]. To maintain a good estimation of the time, each auction was executed in a single core.

13.3. TIME

In our first test we look at the time the system takes to clear an auction as a function of the number of bidders. Remember that when using CATS the number of bidders can be (indirectly) controlled only in distributions that allow XOR bids, that is, only in the arbitrary and the scheduling distributions. Figure 13.2 shows the *average time* for these two distributions as function of the number of bidders.

In the arbitrary distribution, the time for all the bidding algorithms grows linearly as a function of the number of bidders. As expected, PAUSEBID has greatest average of time, its curve has a slope of 514.7. As also expected, CACHEDPAUSEBID is second in time requirements, its curve has a slope of 30.8. It is important to notice that the difference on average time between these two myopic-optimal algorithms is huge. CACHEDPAUSEBID is about 95% faster than PAUSEBID, for example, when the number of bidders is 10, the average time for PAUSEBID is 5676.94 seconds while for CACHEDPAUSEBID it is only 295.93 seconds. The approximate algorithms are much faster, GREEDYPAUSEBID+HILL is about 99% faster than PAUSEBID and GREEDYPAUSEBID is even faster (about 99.88%); for example, when the number of bidders is 10, their corresponding time averages are 56.4 and 6.7 seconds respectively.

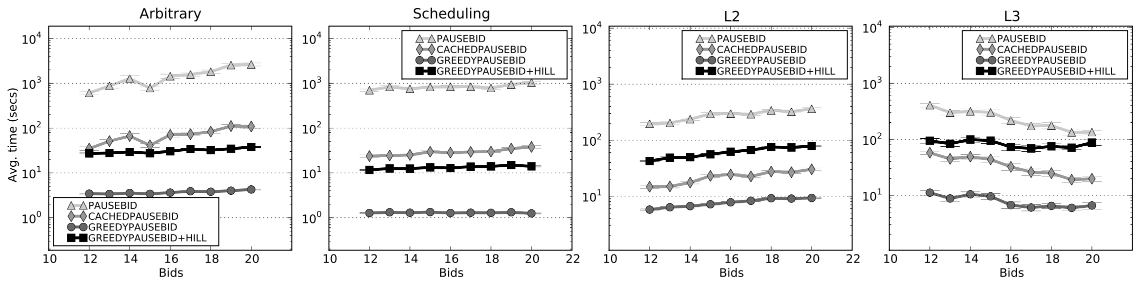


FIGURE 13.3. *Average time* as a function of the number of *bids* in the auction. The number of goods is fixed at 10, and the number of bidders at 5 (for L2 and L3 the number of bidders is the same as the number of bids).

In the scheduling distribution, the average time of PAUSEBID grows exponentially with the number of bidders. CACHEDPAUSEBID is about 98.05% faster than PAUSEBID. For example, when the number of bidders is 10, the average time for PAUSEBID is 135450.76 seconds while for CACHEDPAUSEBID it is 2637.38 seconds. The average time for the approximate bidding algorithms is, as in the arbitrary distribution, linear as a function of the number of bidders, although here their corresponding slopes are steeper.

Figure 13.3 shows the average running time as a function of the number of bids. This figure shows that the number of bids does not significantly affect the average time. In all the distributions, except L3, the time grows linearly as a function of the number of bids, of course the slopes of the curves of the myopic-optimal algorithms are bigger, and the increment is not monotonic. We make two interesting observations. The first one is that in L3 the average time decreases as function of the number of bids, and the second is that CACHEDPAUSEBID is even faster than GREDYPAUSEBID+HILL in the L2 and L3 distributions. For the myopic-optimal bidding algorithms the most difficult distribution (highest average time) is the arbitrary one, followed by scheduling, L2, and L3; while for the greedy strategies, the hardest one is L3 followed by L2, arbitrary and scheduling.

When analyzing the average running time as a function of the number of goods, we find that results are very similar to the results we found earlier using different distributions [**26, 27**] (the time is analyzed as function number of items too, using an original value function or distribution), where the myopic-optimal biding strategies show an exponential

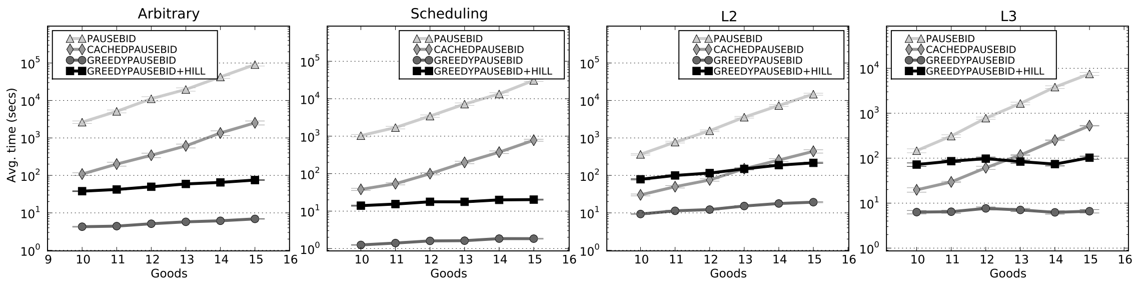


FIGURE 13.4. *Average time* as a function of the number of *goods* in the auction. The number of bids is fixed at 20, and the number of bidders at 5 (for L2 and L3 the number of bidders is the same as the number of bids).

growth rate and the approximate ones show a linear growth rate (Figure 13.4). The only difference is that in these experiments the CACHEDPAUSEBID is much faster than PAUSEBID, here CACHEDPAUSEBID is more that 95% faster in the arbitrary, scheduling and L2 distributions, and 85% in the L3 distribution; while in the previously mentioned experiments CACHEDPAUSEBID was only 50% faster than PAUSEBID.

Something important to notice here is that according to the study done in [**25**], the hardest CATS distribution is L3, closely followed by the arbitrary, L2, and scheduling. However, in the PAUSE auction, L3 is the easiest one for the myopic-optimal bidding strategies and, it gets easier as the number of goods and the number of bids increases [1]. Another interesting observation is that for the approximate bidding strategies, L2 is usually the hardest, followed by the L3 and the arbitrary.

13.4. REVENUE

The *average revenue ratio* (9) increases as a function of the number of bidders in the auction, as shown in Figure 13.5, for all the bidding algorithms. It seems that as the competence increases, the prices that the winning bidders pay also increase. Also in general, the myopic-optimal algorithms have a higher revenue ratio than the greedy strategies, however the ratios become very close to each other as the number of bidders increases. In the arbitrary distribution, the average revenue ratio curve starts at .76 for the greedy algorithms

[1]We have carried out few experiments, not shown here, with much bigger values for all the variables where the greedy strategies are even faster than CASS under this distribution.

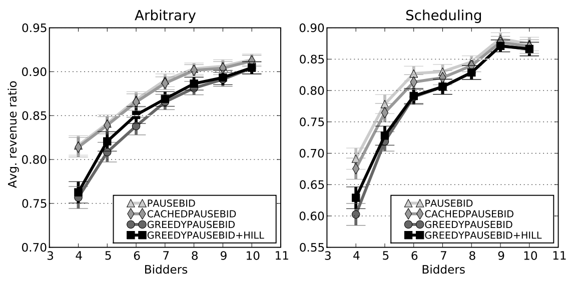


FIGURE 13.5. *Average revenue ratio* (9) as a function of the number of *bidders* in the auction. The number of goods and bids is fixed at 10 and 20 respectively.

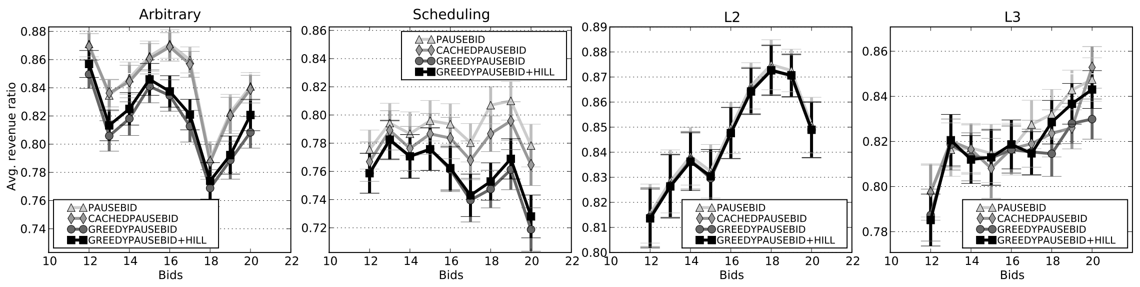


FIGURE 13.6. *Average revenue ratio* (9) as function of the number of *bids* in the auction. The number of goods and bidders is fixed at 10 and 5 respectively. Notice that all the bidding algorithms overlap under the L2 distribution.

and $.81$ for the myopic-optimal ones, and ends in $.90$ and $.91$ respectively with 10 bidders. In this distribution, there is a small difference of about $.1$ between GREEDYPAUSEBID and GREEDYPAUSEBID+HILL when the number of bidders is less than 7, but it disappears afterwards. In the scheduling distribution, the curves start a little bit lower at around $.61$ for the greedy strategies and $.68$ for the myopic-optimal ones, and also end a little bit below $.86$ and $.87$ with 10 bidders. In this distribution, there is a small difference of about $.05$ between PAUSEBID and CACHEDPAUSEBID when the number of bidders is less than 8, but it disappear afterwards.

Figure 13.6 shows a comparison of the average revenue ratio as a function of the number of bids. In the arbitrary distribution both PAUSEBID and CACHEDPAUSEBID have the same average revenue ratio. In this distribution, the revenue ratio decreases non-monotonically, as a function of the number of bids. Although the myopic-optimal algorithms generate

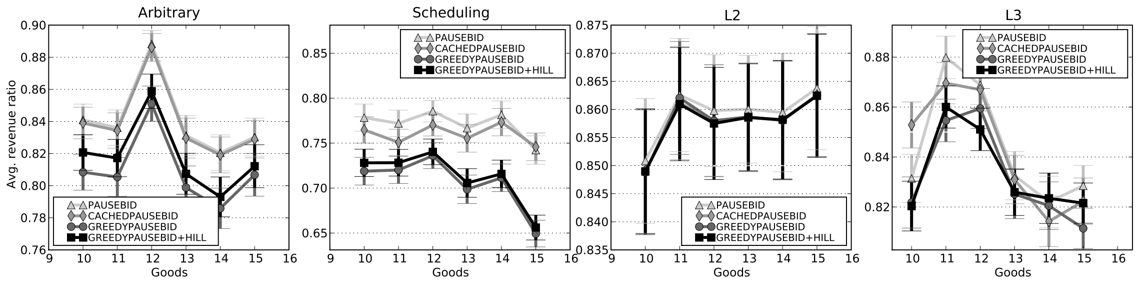


FIGURE 13.7. The *Average revenue ratio* (9) as a function of the number of *goods* in the auction. The number of bids and bidders is fixed at 20 and 5 respectively.

higher revenue than GREEDYPAUSEBID and GREEDYPAUSEBID+HILL, the difference is only .02 and .01 respectively and these differences are very uniform over the number of bids. The maximum value obtained by the myopic-optimal algorithms is .87 and the minimum is .79. In the scheduling distribution there is a difference (about .01) between the average ratio of the two myopic-optimal algorithms, in favor of PAUSEBID. The average revenue ratio for these algorithms tends to increase non-monotonically as function of the number of bids, while it decreases non-monotonically for the approximate strategies, getting more distant from each other as the number of bids increases. Although at the beginning both greedy strategies curves are identical, after 17 bids, the curve of GREEDYPAUSEBID starts going lower, however the difference at 20 bids is still less than .01. In the L2 distribution, all the bidding strategies have the same average revenue ratio and it increases non-monotonically as function of the number of bids; reaching up to .87 with 18 bids. Although in L3 the average revenue ratio also increases non-monotonically as function of the number of bids, although it does not go as high as L2 (the maximum is .87). Although the curves are not identical under this distribution, they are very close to each other. When the number of bids is 20, the CACHEDPAUSEBID shows higher revenue ratio (.85) followed by PAUSEBID and GREEDYPAUSEBID+HILL (both .84), where GREEDYPAUSEBID has the lowest ratio (.83). Amongst the four distributions, L2 is the one in which PAUSE generates higher revenue, followed by arbitrary, L3, and scheduling.

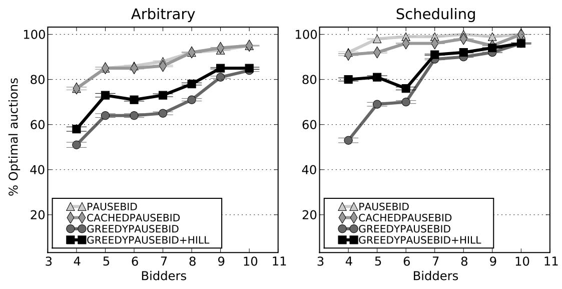


FIGURE 13.8. Percentage of *optimal auctions* as function of the number of *bidders* in the auction. The number of goods and bids is fixed at 10 and 20 respectively.

Figure 13.7 shows the average revenue ratio as a function of the number of goods. The distribution where, on average, higher revenue is generated is L2. In this distribution all the bidding strategies have about the same average revenue ratio which remains between .85 and .86 regardless of the number of goods. In the arbitrary distribution the revenue ratio decreases non-monotonically, the corresponding curves of PAUSEBID and CACHEDPAUSEBID are the same and they are .03 above GREEDYPAUSEBID and .02 above GREEDYPAUSEBID+HILL. In the L3 distribution the revenue ratio decreases non-monotonically as a function of the number of goods. In general, under this distribution the myopic-optimal biding algorithms have higher revenue ratio. The L3 distribution has the third highest average revenue. The lowest revenue is achieved in the scheduling distribution. In this distribution the curves tend to decrease non-monotonically as the number of goods increases. PAUSEBID has higher average revenue ratio than all the other bidding algorithms, although CACHEDPAUSEBID has the same values after 14 goods. Here, GREEDYPAUSEBID+HILL is .05 below and GREEDYPAUSEBID is .06 below PAUSEBID.

13.5. ALLOCATIVE EFFICIENCY

When analyzing the efficiency of the solutions obtained by our algorithms, we found that the number of auctions finishing with an optimal allocation increases as a function of the number of bidders. For example, in the arbitrary distribution, the curve of the myopic-optimal algorithms (shown in Figure 13.8) starts at 76% with 4 items and ends in 95% with

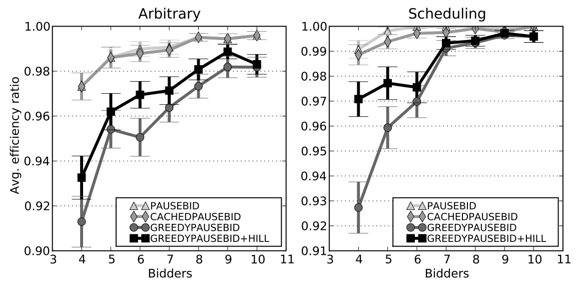


FIGURE 13.9. *Average efficiency ratio* (8) as function of the number of *bidders* in the auction. The number of goods and bids is fixed at 10 and 20 respectively.

10 items. That is, in average, 87.8% of all the auctions in this experiment (615 out of 700) ended up with an optimal allocation; when using any of the myopic-optimal algorithms. Meanwhile, the curve of GREEDYPAUSEBID starts with 51% at 4 items, and increases to 84% at 10 items; only 68.8% of all the auctions in this experiment ended up with an optimal solution when using this algorithm. The GREEDYPAUSEBID+HILL produces better solutions than GREEDYPAUSEBID; in average, the percentage of the total auctions arriving to the optimal solution is 6% higher. However, this percentage is about 13% lower than that of the myopic-optimal algorithms.

Although the curves of the plots for the two distributions presented in Figure 13.8 are very similar, the percentage of total auctions ending up with an optimal allocation is about 11% higher in the scheduling distribution than in the arbitrary distribution, for all the algorithms. This shows that the bid distribution or problem scenario has in impact on the efficiency of the mechanism and the algorithms.

Figure 13.9 shows the *allocative efficiency ratio* (8) in the arbitrary and scheduling distributions as a function of the number of bidders. In general, the efficiency increases monotonically as function of the number of bidders. The fact that the efficiency ratio is not 1 (except for both PAUSEBID and CACHEDPAUSEBID in the scheduling distribution with more than 6 bidders) indicates that the allocation of goods to bidders is different in PAUSE than when using a centralized solution. In general, the myopic-optimal algorithms have an average efficiency ratio higher than the other two and GREEDYPAUSEBID+HILL has a

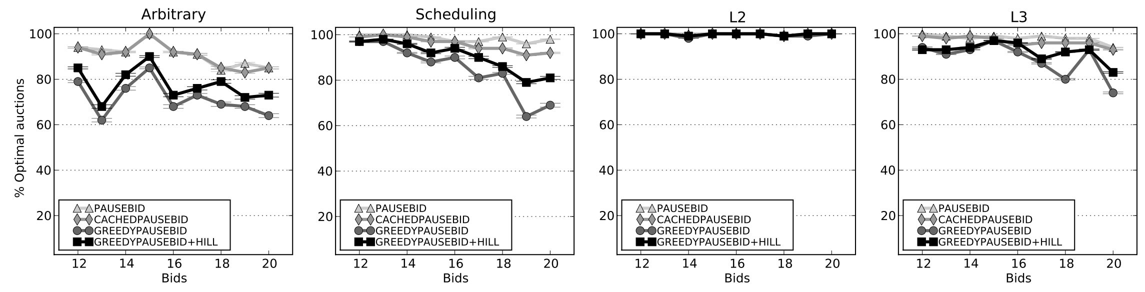


FIGURE 13.10. Percentage of *optimal auctions* as a function of the number of *bids* in the auction. The number of goods and bidders is fixed at 10 and 5 respectively.

higher ratio than GREEDYPAUSEBID. However, their differences decrease as the number of bidders increases.

Figure 13.10 shows that the number of auctions finishing with an optimal allocation decreases as a function of the number of bids, in three of the four bid distributions in the experiment: arbitrary, scheduling, and L3. Only in the L2 distribution the number of bids did not affect the efficiency of the solutions; most of the auctions ended up in an optimal allocation. As in the previous experiment, a higher percent of the auctions ended up in an optimal allocation when using the myopic-optimal algorithms. In the arbitrary distribution, 90% of all the auctions where optimal with both PAUSEBID and CACHEDPAUSEBID. While only 71% of all the auctions were optimal when using GREEDYPAUSEBID and 77% when using GREEDYPAUSEBID+HILL. In the scheduling distribution, the percentages were 98%, 95%, 84% and 90% for PAUSEBID, CACHEDPAUSEBID, GREEDYPAUSEBID, and GREEDY-PAUSEBID+HILL respectively. In the L3 distribution, the corresponding percentage are 98%, 96%, 89%, and 92%. Finally, the percentage of optimal auctions is the same for all the algorithms under the L2 distribution, 99%.

When analyzing how the number of bids affects the *efficiency ratio* (Figure 13.11), we notice that the L2 distribution is not affected at all and that in this distribution all the biding strategies reach the highest efficiency ratio. The efficiency in L3 is also very high (more than .98) and although all the curves are very close to each other, we notice that PAUSE-BID has a higher efficiency. In general, the efficiency ratio decreases a small amount as a

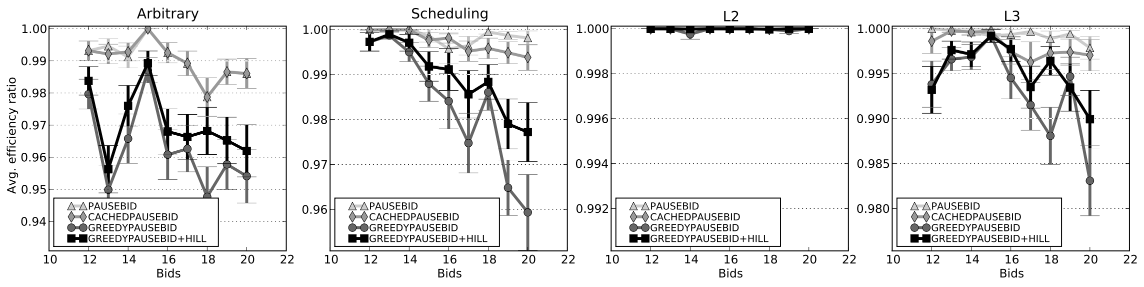


FIGURE 13.11. *Average efficiency ratio* (9) as a function of the number of *bids* in the auction. The number of goods and bidders is fixed at 10 and 5 respectively.

function of the number of bids, with the greedy algorithms affected the most. In the scheduling distribution the efficiency remains high (.998 on average) when using the myopic-optimal bidding algorithms. Both greedy algorithms have an efficiency ratio of .997 at 12 bids, which is almost as high as that of the myopic-optimal algorithms. However, as the number of bids increases, it decreases non-monotonically and their corresponding curves separate from each other. The efficiency ratio reaches .977, for GREEDYPAUSEBID+HILL, and 0.959, for GREEDYPAUSEBID, at 20 bids. The arbitrary distribution has the lowest efficiency among all the distributions studied here. However, it is still high (above .945). Under this distribution we notice that the efficiency ratio curves of all the bidding algorithms tend to decrease non-monotonically. Both, PAUSEBID and CACHEDPAUSEBID, start at .993 with 12 bids and drop to .978 at 18 bids, although they increase again to .986. Here, GREEDYPAUSEBID and GREEDYPAUSEBID+HILL are, on average, .03 and .02 below PAUSEBID respectively.

The effect of the number of goods on efficiency is very similar to the effect of the number of bids. As shown in Figure 13.12, the percentage of optimal auctions decreases as a function of the number of bids, this is for all the distributions except L2. In the arbitrary distribution, the average of auctions finishing in an optimal allocation is 88% for both, PAUSEBID and CACHEDPAUSEBID; and 70% and 77% for GREEDYPAUSEBID and GREEDYPAUSEBID+HILL respectively. The corresponding percentages of auctions finishing in a optimal allocation for PAUSEBID, CACHEDPAUSEBID, GREEDYPAUSEBID, and

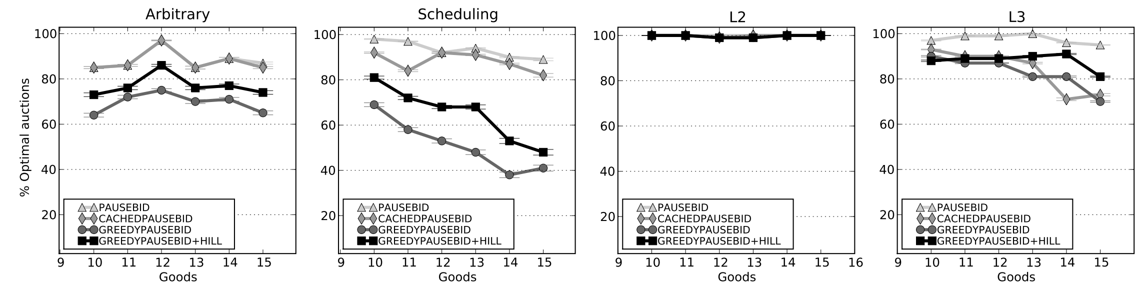


FIGURE 13.12. Percentage of *optimal auctions* as a function of the number of *goods* in the auction. The number of bids and bidders is fixed at 20 and 5 respectively.

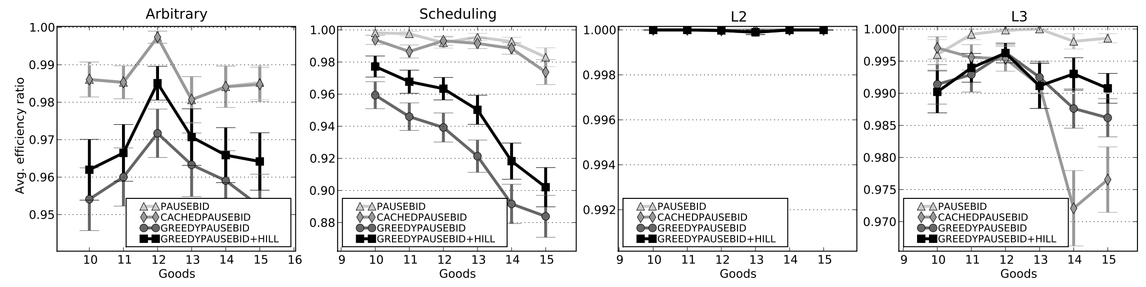


FIGURE 13.13. *Average efficiency ratio* (9) as a function of the number of *goods* in the auction. The number of bids and bidders is fixed at 20 and 5 respectively.

GREEDYPAUSEBID+HILL are: 93%, 88%, 51%, and 65% in the scheduling distribution; 98%, 84%, 83%, and 88% in the L3 distribution; and 100% for all the algorithms in the L2 distribution.

From the point of the *average efficiency ratio*, the allocative efficiency of all the algorithms is high in all the distributions and is not dramatically affected by any of the factors. However, the myopic-optimal bidding algorithms are more efficient than the approximate algorithms. All the PAUSE bidding algorithms have higher efficiency in the L2 and L3 distributions than in the other two distributions.

13.6. BIDDERS' EXPECTED UTILITY

The average *bidders' expected utility ratio* (10) produce curves (shown in Figures Figure 13.14, Figure 13.15, and Figure 13.16) with values roughly inverse to that of the average

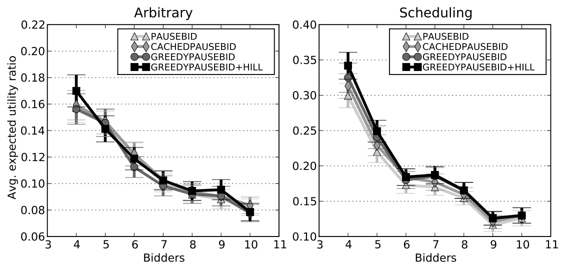


FIGURE 13.14. *Average expected utility ratio* (10) as a function of the number of *bidders* in the auction. The number of goods and bids is fixed at 10 and 20 respectively.

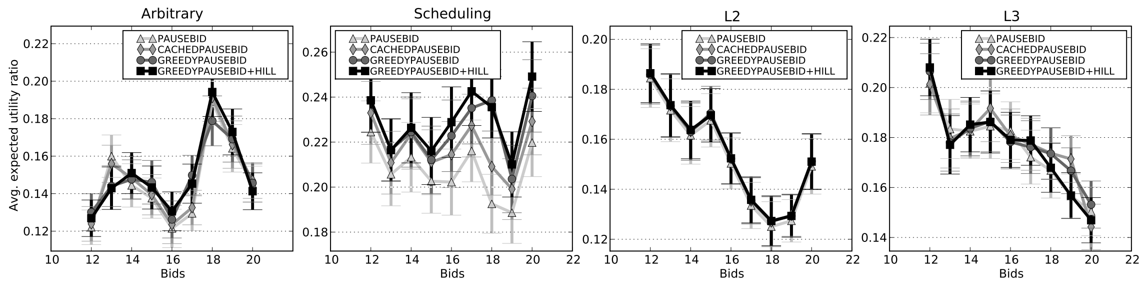


FIGURE 13.15. *Average expected utility ratio* (10) as a function of the number of *bids* in the auction. The number of goods and bidders is fixed at 10 and 5 respectively.

revenue ratio (Figures Figure 13.5, Figure 13.6, Figure 13.7). This relation was expected, since the higher the payments (the revenue) the lower the possible expected utility. That means that in the L2 and L3 distributions, where the revenue of all the bidding algorithms is very similar, the bidding strategy does not have any effect over the bidders' expected utility. The only incentive to choose one over the other is speed. Since, in general, GREEDYPAUSEBID is the fastest, it is the best choice.

In the arbitrary and scheduling distributions, where the revenue ratio is different amongst the bidding strategies, the approximate bidding strategies result in higher bidders' expected utility—which provides prospective bidders with an incentive to join a PAUSE auction. However, GREEDYPAUSEBID+HILL is the one that offers the highest expected utility and not GREEDYPAUSEBID, which is the one that most often generates lower revenue; although, as shown in Figures Figure 13.14, Figure 13.15, Figure 13.16, the error bars overlap most of the time. GREEDYPAUSEBID+HILL is usually more allocative efficient, as shown before.

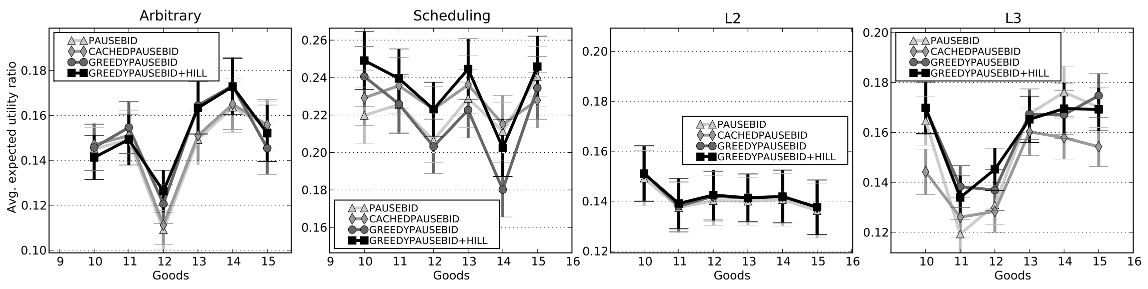


FIGURE 13.16. *Average expected utility ratio* (10) as a function of the number of *goods* in the auction. The number of bids and bidders is fixed at 20 and 5 respectively.

Thus, in the allocations obtained by GREEDYPAUSEBID+HILL the winners are bidders that have higher valuations than those in the allocations obtained by GREEDYPAUSEBID, and yet they do not pay as much as the bidders in the allocations obtained by PAUSEBID. The previous example shows that there is a relation between efficiency and bidders' expected utility, and the latter does not depend solely on lower revenue (lower prices).

13.7. RESULTS SUMMARY

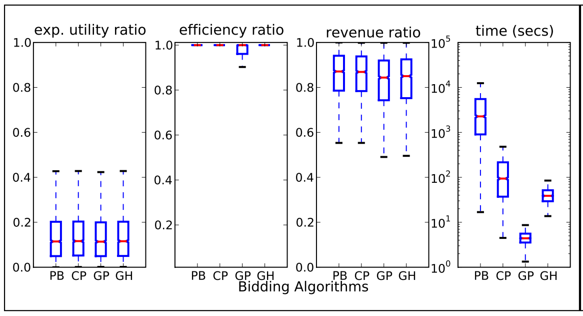
As a summary of the results of our study, Table Table 13.2 contains the average and standard deviation and Figure 13.17 shows the dispersion measures from all the auctions carried out. Besides the time, the metrics used for this study are the ratios of the bidders' expected utility, allocative efficiency, and revenue. The corresponding values of these metrics for the revenue-maximizing solution are: zero for the first one and 1 the other two. The results of our study show that all our algorithms offer about the same bidder's expected utility for switching from a centralized auction to a PAUSE auction, although the greedy algorithms are a little bit better. On average, GREEDYPAUSEBID+HILL has higher bidder's expected utility ratio than all the other algorithms, its advantage is between 0.002 to 0.004 in the arbitrary distribution, 0.008 to 0.018 in the scheduling distribution, and 0.001 to 0.002 in the L2 distribution (Table Table 13.2). However, in the L3, GREEDYPAUSEBID has an advantage between 0.001 and 0.005 over the other algorithms. In the scheduling distribution almost 50% of the auctions had a bidder's expected utility ratio grater than or

Distri-bution	Bidding Algorithm	exp. utility ratio		efficiency ratio		revenue ratio		time (secs)	
		average	stdev	average	stdev	average	stdev	average	stdev
Arbitrary	PB	0.135	0.104	0.989	0.040	0.854	0.106	9713	27550
	CP	0.137	0.103	0.989	0.040	0.852	0.106	325	982
	GP	0.137	0.107	0.961	0.077	0.824	0.118	5	2
	GH	**0.139**	0.108	0.970	0.068	0.831	0.116	42	17
Scheduling	PB	0.203	0.144	0.997	0.022	0.794	0.144	12824	33185
	CP	0.211	0.144	0.994	0.029	0.784	0.144	282	685
	GP	0.213	0.153	0.965	0.079	0.751	0.155	2	1
	GH	**0.221**	0.152	0.977	0.065	0.756	0.153	19	11
L2	PB	0.148	0.107	1.000	0.000	0.852	0.107	2057	4695
	CP	0.149	0.107	1.000	0.000	0.851	0.107	80	178
	GP	0.149	0.107	1.000	0.001	0.851	0.107	10	5
	GH	**0.150**	0.107	1.000	0.000	0.850	0.107	93	61
L3	PB	0.167	0.105	0.999	0.009	0.832	0.105	1091	2586
	CP	0.165	0.103	0.994	0.027	0.829	0.105	88	137
	GP	**0.170**	0.103	0.992	0.026	0.822	0.101	7	8
	GH	0.169	0.103	0.994	0.023	0.825	0.101	84	83

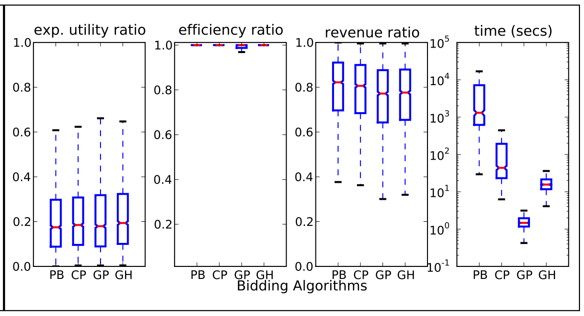
TABLE 13.2. Average and standard deviation of the metrics used in our experiments for each distribution and bidding algorithm pair. For the arbitrary and scheduling distributions the values correspond to the results of 2200 auctions. For the L2 and L3 distributions the values correspond to the result of 1500 different auctions. We define *exp. utility ratio* in (10), *efficiency ratio* in (8), and *revenue ratio* in (9). In the table we use PB for PAUSEBID, CP for CACHEDPAUSEBID, GP for GREEDYPAUSEBID, and GH for GREEDYPAUSEBID+HILL.

equal to 0.2 (Figure 13.17(b)); on average, in this distribution the highest bidder's utility was reached. In the arbitrary distribution about 75% of the auctions had a bidders' expected utility ratio less than 0.2 (Figure 13.17(a)). In this distribution the bidders might got, on average, the lowest utility.

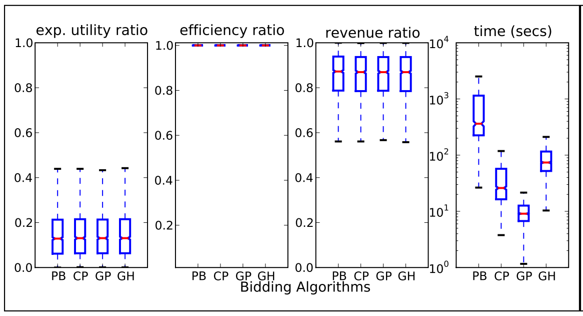
Our test results also show that our algorithms have a high efficiency ratio, this means that they frequently arrive at the same allocation of items to bidders as the revenue-maximizing solution. Most of the cases where our algorithms failed to arrive at that allocation are those where there was a large gap between the first and second valuation for a set (or sets) of items. In the L2 and L3 distributions all the algorithms have the same allocative efficiency ratio. However, in the other two distributions there is small difference in favor of the myopic-optimal algorithms. GREEDYPAUSEBID is the one with the lowest allocative



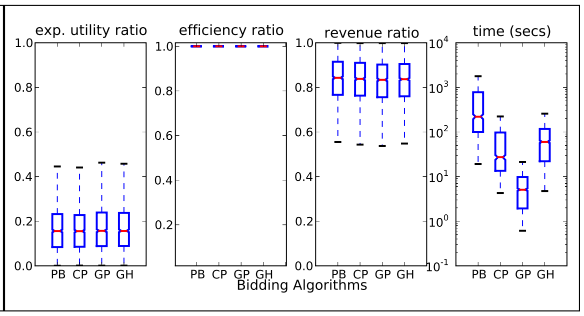
(A) Arbitrary



(B) Scheduling



(C) L2



(D) L3

FIGURE 13.17. Median and upper and lower quartiles of the different metrics used in our experiments for each distribution and bidding algorithm pair. The plots contain the results of 2200 auctions, for the arbitrary and scheduling distribution, and 1500 auctions, for the L2 and L3 distributions. We define *exp. utility ratio* in (10), *efficiency ratio* in (8), and *revenue ratio* in (9). In these plots we use PB for PAUSEBID, CP for CACHEDPAUSEBID, GP for GREEDYPAUSEBID, and GH for GREEDYPAUSEBID+HILL. The plots do not contain outliers.

efficiency. The plots related to the revenue ratio in Figure 13.17(a) show that 50% of the auctions carried out in the arbitrary distribution had an efficiency ratio lower than one when using this algorithm; although only 25% of the auctions had an allocative efficiency ratio between 0.9 and 0.95, the rest had an allocative efficiency ratio grater than .95. It is also possible to appreciate, in Figure 13.17(b), that in the scheduling distribution 50% of the auctions had also an efficiency ratio lower than when using GREEDYPAUSEBID; nevertheless, the lowest 25% of the auctions are between .97 and .98. Thus, we show that even the algorithm with the worse allocative efficiency ratio is highly allocative efficient.

The running time of PAUSEBID and CACHEDPAUSEBID remains exponential in the number of bidders, goods, and bids, although the last one does not affect the time as much as the others. However, both algorithms are significantly better than a full search, and CACHEDPAUSEBID is more than 95 times faster than PAUSEBID. The running time of the greedy algorithms remains linear in the number of bidders, goods, and bids, and they are up to 99 times faster than PAUSEBID; with GREEDYPAUSEBID being the fastest one. Table Table 13.2 shows that the average time of the greedy algorithms is only a small fraction of that of the myopic-optimal ones. Figures Figure 13.17(a) and Figure 13.17(b) show that, under the arbitrary and scheduling distributions, all the auctions were cleared much faster when using GREEDYPAUSEBID than when using the other algorithms. Those figures also show that GREEDYPAUSEBID+HILL is, in general, much faster than the myopic-optimal algorithms; although there are few cases where CACHEDPAUSEBID was faster. An interesting observation is that under the L2 distribution CACHEDPAUSEBID is generally faster than GREEDYPAUSEBID+HILL, Figure 13.17(c), and under L3 distribution their average time and dispersion measurements are about the same, Figure 13.17(d).

Thus, the greedy algorithms offer a double incentive to the bidders, higher expected utility and the speed. Rational bidders should use GREEDYPAUSEBID+HILL in the arbitrary, scheduling, and L2 distributions when time is not an issue. When more speed is crucial, they should use GREEDYPAUSEBID. On average the difference on bidder's expected utility ratio between GREEDYPAUSEBID+HILL and GREEDYPAUSEBID might not be very significant (GREEDYPAUSEBID is only 0.002 below in the arbitrary and scheduling distributions and 0.001 below in the L3 distribution), however their difference in time is more significant (GREEDYPAUSEBID is more than 10 times faster).

As expected, since PAUSE is an ascending price auction, the revenue generated by the PAUSE auction is lower than the revenue of a revenue-maximizing solution, most of the time. Under the arbitrary distribution, the average revenue ratio of PAUSEBID and CACHEDPAUSEBID is around 0.85, while for GREEDYPAUSEBID and GREEDYPAUSEBID+HILL is around 0.83. In the scheduling distribution their average revenue is 0.79 and 0.75 respectively. In the other two distributions there is almost no difference amongst the revenue generated by all the algorithms, 0.85 in L2 and 0.83 in L3.

Part 4

Conclusions

CHAPTER 14

CONCLUSIONS

In recent years, companies and governments around the world have left behind administered allocation systems, single item auctions, and other ad hoc mechanisms; traditionally used to sell valuable commodities, solve sourcing problems, or allocate scarce public resources; to take advantage of the power of combinatorial auction based market mechanisms, maximizing revenue and minimizing cost of sales. The "package" bidding enabled by these mechanisms allows bidders to benefit from combining the complementarities of the items being auctioned and to better express the value of any synergies. Different from consumer auctions platforms like eBay, combinatorial auctions platforms are used in high stakes business-to-business environments. Thus, combinatorial auctions have been fundamentally changing the way of selling valuable resources, allowing the creation of electronic markets to supplement traditional sales channels like bilateral negotiated contracts, and creating markets where in the past there were none or where the market is not exactly a glowing example of great commercial outcomes.

However, combinatorial auctions mechanisms require a central auctioneer, a third party that stays between the seller and the buyer or the provider and the consumer; which receives the bids from the bidders and carries out the computation to find the best allocation—an NP-complete problem—and, of course, it receives a fee for its service. Unfortunately, these types of centralized auctions are not a good fit for some scenarios; as when for example, the resources are owned by different entities and each entity has localized information, when the bidders do not want to reveal their valuations (some times their bids represent valuable information like production cost), or when it is difficult or unaffordable to establish a trusted auctioneer. These cases are the motivation of our research into distributed combinatorial auctions, peer-to-peer incentive compatible mechanisms for solving the allocation problem. An approach were the computational problem of finding an allocation is distributed amongst the bidders, who have a clear financial incentive to perform this computation.

The PAUSE auction is one of a few approaches to decentralize the winner determination problem in combinatorial auctions. With this auction, we can even envision completely eliminating the auctioneer and, instead, have every agent perform the task of the auctioneer. However, while PAUSE establishes the rules the bidders must obey, it does not tell us how the bidders should calculate their bids. We have presented two myopic-optimal (Chapters 10) and two heuristic-approximate 11 bidding algorithms. We implemented our algorithms and carried out an experimental study over different type of problems.

Our algorithms have shown that it is feasible to implement distributed combinatorial auctions without having to resort to a centralized winner determination algorithm (Sections 10.3 and 11.3). Bidder agents that are not currently part of a PAUSE auction have an incentive to join a PAUSE auction and, because of the design of the PAUSE auction, perform the required computation. We have shown that their utility will increase if they abandon their centralized combinatorial auction in favor of PAUSE and use our algorithms. We found that our heuristic greedy algorithms offer high speed as a second incentive (Section 11.3). We have also shown that the allocations obtained with our algorithms are very efficient. Moreover, we have shown, through a game theoretical analysis, that the case where everybody bids using GREEDYPAUSEBID+HILL algorithm is a non-strict Nash equilibrium (Chapter 12). The revenue generated by a PAUSE auction is lower than the optimal, nevertheless, we found that it increases as the number of bidders in the auction increases because of the increased competition (Chapter 13). Thus, the bidders's benefits provided by PAUSE will attract more bidders, which would result in more benefits for the seller in addition of the savings from eliminating the tax paid to the third party.

Thus, our experiments have shown that, over a representative set of problems, the PAUSE auction is both efficient in terms of the solution it finds as well as the time it takes to find it (Chapter 13). We have shown that the PAUSE auction along with the heuristic bidding algorithms is a realistic method for solving combinatorial allocation problems without the use a centralized auctioneer. Moreover, because of the computational tractability of this mechanism our algorithms are suitable for problems where agents with small computational power are involved such as electronic commerce and automated negotiation. Another possible application is in multirobot environments where robots are responsible for coordinating complex tasks such as transporting equipments within a manufacturing plant, delivering packages in an office, rescuing victims in situations inaccessible by humans, or tracking enemy targets in battlefields. We have also shown that the nature of bidding distributions related to the problem being solved do have a significant effect on the run time of the algorithm, so this should be taken into consideration before deploying a PAUSE-based system.

We are looking at how to further increase the scalability of the PAUSE auction to millions of agents by allowing agents to place incomplete bidsets and only communicate their bids to a subset of agents. Our goal is to develop new auctions which can scale to any number of bidders while distributing the computational cost evenly amongst them.

BIBLIOGRAPHY

1. Martin R. Andersson and Tuomas W. Sandholm, *Contract types for satisficing task allocation: Ii experimental results*, AAAI Spring Symposium: Satisficing Models, 1998.

2. JianCong Bai, HuiYou Chang, and Yang Yi, *An immune partheno-genetic algorithm for winner determination in combinatorial auctions*, ICNC (3), 2005, pp. 74–85.

3. Liad Blumrosen and Noam Nisan, *Combinatorial auctions*, Algorithmic Game Theory (Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani, eds.), Cambridge University Press, New York, NY, USA, 2007, pp. 267–299.

4. Craig Boutilier and Holger H. Hoos, *Bidding languages for combinatorial auctions*, Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, 2001, pp. 1211–1217.

5. Paul J. Brewer, *Decentralized computation procurement and computational robustness in a smart market*, Economic Theory **13** (1999), no. 1, 41–92.

6. Harumi Kuno Claudio Bartolini, Chris Preist, *Requirements for automated negotiation*, W3C workshop on Web services, W3C, 2001.

7. Peter Cramton, *Simultaneous ascending auctions*, in Cramton et al. [**8**], pp. 99–114.

8. Peter Cramton, Yoav Shoham, and Richard Steinberg (eds.), *Combinatorial auctions*, MIT Press, 2006.

9. Sven de Vries and Rakesh V. Vohra, *Combinatorial auctions: A survey*, INFORMS Journal on Computing **15** (2003), no. 3, 284–309.

10. Joan Feigenbaum and Scott Shenker, *Distributed algorithmic mechanism design: Recent results and future directions*, Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, ACM Press, New York, 2002, pp. 1–13.

11. Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham, *Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches*, Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers Inc., 1999, pp. 548–553.

12. Naoki Fukuta and Takayuki Ito, *Towards better approximation of winner determination for combinatorial auctions with large number of bids*, Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology, 2006, pp. 618–621.

13. Peter Gradwell and Julian Padget, *Markets vs auctions: Approaches to distributed combinatorial resource scheduling*, Multiagent and Grid Systems **1** (2005), no. 4, 251 – 262.

14. Holger H. Hoos and Craig Boutilier, *Solving combinatorial auctions using stochastic local search*, Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, AAAI Press / The MIT Press, 2000, pp. 22–29.

15. Brahim Chaib-draa Houssein Ben-Ameur and Peter Kropf, *Multi-item auctions for automatic negotiation*, Information and Software Technology **44** (2002), no. 5, 291–301.

16. Michael N. Huhns and Munindar P. Singh (eds.), *Readings in agents*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

17. Matthew O. Jackson, *Mechanism theory*, The Encyclopedia of Life Support Systems, EOLSS Publishers, 2000.

18. Frank Kelly and Richard Stenberg, *A combinatorial auction with multiple winners for universal service*, Management Science **46** (2000), no. 4, 586–596.

19. Sarit Kraus, *Beliefs, time and incomplete information in multiple encounter negotiations among autonomous agents*, Annals of Mathematics and Artificial Intelligence **20** (1997), no. 1–4, 111–159.

20. ______, *Automated negotiation and decision making in multiagent environments*, Mutli-agents systems and applications (Jaime G. Carbonell and Jorg Siekmann, eds.), Springer-Verlag, 2001, pp. 150–172.

21. Alisa Land, Susan Powell, and Richard Steinberg, *PAUSE: A computationally tractable combinatorial auction*, in Cramton et al. [**8**], pp. 139–157.

22. Daniel Lehmann, Rudolf Muller, and Tuomas Sandholm, *The winner determination problem*, in Cramton et al. [**8**], pp. 297–317.

23. Daniel Lehmann, Liadan Ita Oćallaghan, and Yoav Shoham, *Truth revelation in approximately efficient combinatorial auctions*, Journal of the ACM **49** (2002), no. 5, 577–602.

24. Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham, *Towards a universal test suite for combinatorial auction algorithms*, Proceedings of the 2nd ACM conference on Electronic commerce, ACM Press, 2000, pp. 66–76.

25. Kevin Leyton-Brown and Yoav Shoham, *A test suite for combinatorial auctions*, in Cramton et al. [**8**], pp. 451–478.

26. Benito Mendoza and José M. Vidal, *Bidding algorithms for a distributed combinatorial auction*, Proceedings of the Autonomous Agents and Multi-Agent Systems Conference, 2007.

27. ______, *Approximate bidding algorithms for a distributed combinatorial auction (short paper)*, Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (Estoril, Portugal) (Padgham, Parkes, Müller, and Parsons, eds.), May 2008.

28. Muralidhar V. Narumanchi and José M. Vidal, *Algorithms for distributed winner determination in combinatorial auctions*, LNAI volume of AMEC/TADA, Springer, 2006.

29. Noam Nisan, *Bidding and allocation in combinatorial auctions*, Proceedings of the ACM Conference on Electronic Commerce, 2000, pp. 1–12.

30. Sunju Park and Michael H. Rothkopf, *Auctions with endogenously determined allowable combinations*, Tech. report, Rutgets Center for Operations Research, January 2001, RRR 3-2001.

31. David C. Parkes and Jeffrey Shneidman, *Distributed implementations of vickrey-clarke-groves auctions*, Proceedings of the Third International Joint Conference on Autonomous Agents and MultiAgent Systems, ACM, 2004, pp. 261–268.

32. David C. Parkes and Lyle H. Ungar, *Iterative combinatorial auctions: Theory and practice*, Procedings of the 17th National Conference on Artificial Intelligence (AAAI-00), 2000, pp. 74–81.

33. Jeffrey S. Rosenschein and Gilad Zlotkin, *Rules of encounter*, The MIT Press, Cambridge, MA, 1994.

34. Michael H. Rothkopf, Aleksandar Pekec, and Ronald M. Harstad, *Computationally manageable combinational auctions*, Management Science **44** (1998), no. 8, 1131–1147.

35. Stuart Russell and Peter Norvig, *Artificial intelligence: A modern approach*, second ed., Prentice Hall, 2003.

36. Yuko Sakurai, Makoto Yokoo, and Koji Kamei, *An efficient approximate algorithm for winner determination in combinatorial auctions*, EC '00: Proceedings of the 2nd ACM conference on Electronic commerce (New York, NY, USA), ACM, 2000, pp. 30–37.

37. Tuomas Sandholm, *An algorithm for winner determination in combinatorial auctions*, Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, 1999, pp. 542–547.

38. ———, *An algorithm for winner determination in combinatorial auctions*, Artificial Intelligence **135** (2002), no. 1-2, 1–54.

39. ———, *Expressive commerce and its application to sourcing: How we conducted $35 billion of generalized combinatorial auctions*, AI Magazine **28** (2007), no. 3, 45–58.

40. Tuomas Sandholm, Subhash Suri, Andrew Gilpin, and David Levine, *CABOB: a fast optimal algorithm for winner determination in combinatorial auctions*, Management Science **51** (2005), no. 3, 374–391.

41. Tuomas W. Sandholm, *Distributed rational decision making*, Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence (Gerhard Weiss, ed.), The MIT Press, Cambridge, MA, USA, 1999, pp. 201–258.

42. Katia Sycara, *Multiagent systems*, AI Magazine **10** (1998), no. 2, 79–93.

43. Jay M. Tenenbaum, *AI meets web 2.0: Building the web of tomorrow, today*, AI Magazine **27** (2006), no. 4.

44. Jose M. Vidal, *Fundamentals of multiagent systems with netlogo examples*, 2007.

45. Nikos Vlassis, *A concise introduction to multiagent systems and distributed artificial intelligence*, Synthesis Lectures on Artificial Intelligence and Machine Learning **1** (2007), no. 1, 1–71.

46. Gerhard Weiss (ed.), *Multiagent systems: A modern approach to distributed artificial intelligence*, MIT Press, 1999.

47. Michael Wooldridge, *Introduction to multiagent systems*, John Wiley and Sons, 2002.

48. Michael Wooldridge and Nicholas R. Jennings, *Intelligent agents: Theory and practice*, The Knowledge Engineering Review **10** (1995), no. 2, 115–152.

49. Edo Zurel and Noam Nisan, *An efficient approximate allocation algorithm for combinatorial auctions*, Proceedings of the ACM Conference on Electronic Commerce, 2001.