A STUDY IN COMBINATORIAL AUCTIONS

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF INFORMATICS

OF

THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

BETÜL BİLGE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF INFORMATION SYSTEMS

JULY 2004

Approval of the Graduate School of Informatics

_____

Prof. Dr. Neşe YALABIK
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Assoc. Prof.Dr. Onur DEMİRÖRS
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Assoc. Prof. Ferda Nur ALPASLAN
Supervisor

Examining Committee Members

Prof. Dr. Semih BİLGEN _____

Prof. Dr. Faruk POLAT _____

Assoc. Prof. Dr. Ferda Nur ALPASLAN _____

Assist. Prof. Dr. Erkan MUMCUOĞLU _____

Dr. Altan KOÇYİĞİT _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this wok.

Betül Bilge

ABSTRACT

A STUDY IN COMBINATORIAL AUCTIONS

Bilge, Betül
M.S., Department of Information Systems
Supervisor: Assoc. Prof. Dr. Ferda Nur Alpaslan

July 2004, 101 pages

By the emergence of electronic commerce and low transaction costs on the Internet, an interest in the design of new auction mechanisms has been arisen. Recently many researchers in computer science, economics, business, and game theory have presented many valuable studies on the subject of online auctions, and auctions theory.

When faced from a computational perspective, combinatorial auctions are perhaps the most challenging ones. Combinatorial auctions, that is, auctions where bidders can bid on combinations of items, tend to lead to more efficient allocations than traditional auction mechanisms in multi-item multi-unit situations where the agents' valuations of the items are not additive. However, determining the winners to maximize the revenue is NP-complete.

In this study, we first analyze the existing approaches for combinatorial auction problem. Based on this analysis, we then choose three different approaches, which are search approach, descending simultaneous auctions approach, and IP (Integer Programming) formulation approach to build our models. The performances of the models are compared using computer simulations, where we model bandwidth allocation system. Finally a combinatorial auction tool is built which can be used for online auctions and e-procurement systems.

Keywords: Auctions, Combinatorial Auctions, Multi-item multi-unit auctions, Simultaneous auctions, Integer Programming, Linear Programming, e-commerce, e-procurement, Winner determination, Simulation, Bandwidth allocation

ÖZ

TÜMLEŞİK AÇIK ARTIRMALAR ÜZERİNE BİR ÇALIŞMA

Bilge, Betül
Master, Bilişim Sistemleri Bölümü
Tez yöneticisi: Doç. Dr. Ferda Nur Alpaslan

Temmuz 2004,101 sayfa

Elektronik işin ilerlemesi ve işlem ücretlerinin İnternette düşük olması sebebiyle yeni açık artırma mekanizmaları tasarlamak için ilgi artmıştır. Yakın zamanda bilgisayar bilimleri, ekonomi, iş, ve oyun teorisi üzeride çalışan pek çok bilim adamı açık artırma teorileri üzerinde çok değerli çalışmalar yapmışlardır.

İşlemsel açıdan bakıldığında, tümleşik açık artırmalar en göze çarpan açık artırma türü olarak karşımıza çıkmaktadır. Tümleşik açık artırmalarda, teklif verenler satılan parçaların bileşimine bir teklif verebilir. Bu tür açık artırmalar, teklif verenlerin satılan parçalara biçtikleri değerin toplamsal olmadığı, çok-parça çok-birim olan açık artırma modellerinde geleneksel açık artırmalara gore çok daha verimli paylaştırma sağlar. Ama tümleşik açık artırmalarda kazancı azami hadde çıkaracak, kazanaları belirlemenin NP-Complete olduğu görülmüstür.

Bu çalışmada, önce tümleşik açık artırma problemi için yapılmış olan yaklaşımlar incelenmiştir. Bu inceleme sonunda üç farklı yaklaşım seçilmiş ve bu yaklaşımlar modellenmiştir. Bu yaklaşımlar : arama yolu, eşzamanlı azalan acık artırma yolu, ve sayısal programlama yoludur. Bu modellerin performansı yapılan simulasyonla karşılaştırılmıştır. Bu simulasyonda bant genişliği paylaştırımı modellenmiştir. Son olarak bir tümleşik açık artırma aracı yapılmıştır.

Anahtar Kelimeler : Açık Artırma, Tümleşik açık artırma, Cok-parça çok-birim açık artırmaları, Eşzamanlı açık artırmalar, Sayısal programlama, Lineer programlama, e-iş, e-tedarik, Kazanan belirleme, Simulasyon, Bant genişliği paylaştırımı

Aileme,

Teman, Hayrettin, Buğra ve Burak Bilge.

Rüzgarda bir yapraktım,
Dalınıza tutundum.
Yağmurda bir damlaydım,
Çiçeğinize çiğ oldum.
Sizsiz bu kocaman ormanda kaybolurdum.
Herşey için
TEŞEKKÜRLER…

ACKNOWLEDGEMENTS

I express sincere thanks to my advisor Assoc. Prof. Dr. Ferda Nur Alpaslan for providing insight and guidance as well as encouragement and inspiration throughout this research.

Special thanks go to my friends Gökhan Erbay, Beyza Özkan, Fatma Cemile Serçe and Nigar Şen Köktaş for their valuable comments, suggestions, and supports.

Finally, my deepest thanks are to my parents and brothers who supported and motivated me with their never ending patience, tolerance and understanding throughout the study. Thanks are also to everyone who helped me directly or indirectly in the development and writing of this thesis by providing guidance and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application Programming Interface
AUSM	Adaptive User Selection Mechanism
BOB	Branch on Bids
CABOB	Combinatorial Auction Branch on Bids
CAP	Combinatorial Auction Problem
CATool	Combinatorial Auction Tool
DFS	Depth First Search
DSAM	Descending Simultaneous Auctions Model
FCC	Federal Communication Commission
GLPK	GNU Linear Programming Kit
GVA	Generalized Vickrey Auction
GUI	Graphical User Interface
IDA*	Iterative deepening A* Search
IP	Integer Programming
IPFM	IP Formulation Model
LDS	Limited Discrepancy Search
LP	Linear Programming
MIP	Mixed Integer Programming
OR	Operations Research
SM	Search Model
SPP	Set Packing Problem

VCG Vickrey Clarke Groves

VCL Visual Component Library

CHAPTER 1

Introduction

As the Internet has grown and become widely used, commerce has been shifted to this world. Recently many researchers in computer science, economics, business and game theory are interested in e-commerce, and a very rapid progress can be seen in the area.

According to the Object Management Group (OMG, http://www.omg.org), commerce is at its heart an exchange of information. Electronic marketplaces are places where buyers and sellers can come together, exchange information, negotiate and transact as in traditional marketplaces. In the recent years, agent technology has also been applied to e-commerce. By addition of the software agent technologies several time consuming stages of commerce can be automated, since unlike traditional software, software agents are personalized, continuously running, and semi-autonomous [18]. As a result, many agent-oriented software systems have been developed that model different commerce areas, for example Fox et al. [8] demonstrated agent-based supply-chain architectures and Arpınar et al. [1] proposed an electronic marketplace called MOPPET, where the commerce process in the marketplace was modeled as agent-based workflows.

It is seen that auctions are one of the most successful negotiation protocol and account for an enormous volume of transactions on the Internet. In an auction, a seller sells goods to several potential buyers. Determining the auction's winner and its payment is trivial in single-item auctions. The problem is also computationally tractable in multi-item (homogeneous or heterogeneous) auctions when agents' valuations for different

items are additive, meaning that total valuation can be determined in an additive manner by their valuations for single items. But the problem starts when bidder (buyer) agents have preferences over bundles, i.e. a bidder's valuation for the bundle need not to be equal to the sum of his valuations of the individual items in the bundle, that is valuation need not to be additive. This problem is referred to as the *combinatorial auction problem*. In a combinatorial auction, a seller is faced with a set of price offers for various bundles of goods, and his aim is to allocate the goods in a way that maximizes his revenue.

In this study, we focus on combinatorial auctions problem; especially multi-item multi-unit auctions in which bidders have preferences over bundles and their valuation for the bundle is not additive. After reviewing the previous works, we decide to build three different models using different approaches of the previous works. The performance of each model is compared and analyzed using computer simulation where we model bandwidth allocation among multiple users. Based on the performance of the models and some other situations, one of the models is chosen and a simple web server is built which has the ability of performing combinatorial auction and can be used for online auctions, and e-procurement systems and can be used as a test bed for combinatorial auctions

1.1 Background

E-marketplaces which employ multi-agent technology can be analyzed as marketplaces that is entirely open and decentralized, i.e. no single site controls the market, and each participant initializes an agent that will act in the interest of its owner, or as marketplaces where the participants agree upon certain set of rules about what can be bought and sold in this e-marketplace and how this can be done, i.e. they negotiate. Negotiation is a crucial part of commercial activities in physical as well as in electronic markets.

Guttman et al. described negotiation as a form of decision-making where two or more parties jointly search a space of possible solutions in order to reach a consensus [12]. Game Theory and economics describe such an interaction in terms of protocols and strategies, where the protocols include the rules (i.e. legal actions) of the game [12].

Guttman et al. [12] analyzed several electronic markets and their corresponding negotiation protocols from economic, game theoretic and business perspectives. Three different possible competitive negotiation models in an e-marketplace can be seen in Figure 1 which is taken from [12].

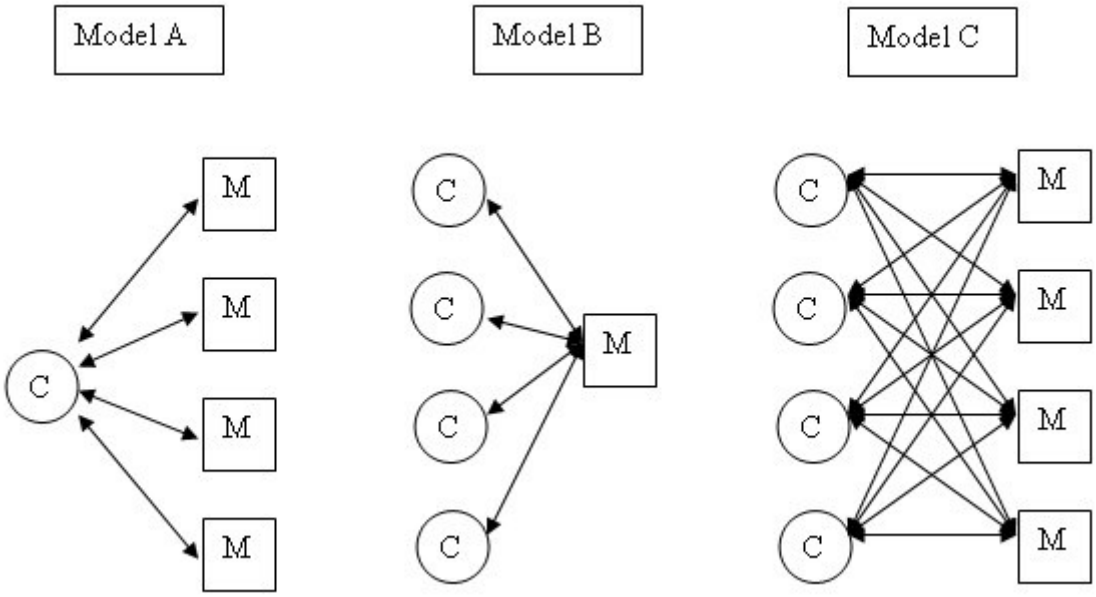


Figure 1 three different possible competitive negotiation models in an e-marketplace {seller is denoted by M ("merchant") and the buyer by C ("consumer")}(from [12])

1.1.1 Negotiation Models

The previous existing marketplaces are categorized according to the classification schema given in Figure 1. These are as follows:

Negotiation Model A

Comparison-shopping agents (also known as shopping bots) fit in this category. Shopping agent research dates to the Web's early years. Some comparison – shopping agents are given below.

BargainFinder, which was developed in 1995 by Andersen Consulting, was one of the first shopping agent. It was implemented for getting and comparing prices for CD's from virtual retailers. However, because that some retailers blocked access since they did not want to complete on price, BargainFinder stopped operation [7].

Ringo, which was one of the earliest commercial software-agent technologies, makes personalized recommendations for music albums and artists on the basis of collaborative filtering by using options of like-minded users. [29].

Menczer at al. [7] claimed that most of the comparison-shopping agents that are available to consumers were biased, i.e. presenting results only from partner companies that pay fees to participate. In order to make user-centered business viable, they developed IntelliShopper, which learns shoppers' individual preferences and autonomously monitors vendor sites for items matching with those preferences.

Negotiation Model B

Online auctions fit in this category. Online auctions represent a model for the way the Internet is shaping the new economy. Online auctions are a very hot topic which many researches are interested in. Some online auction agents are given below.

eBay [6] supports, among other features, an English auction with reserve price and proxy bidding for the bidders. eBay's over 10 million monthly visitors provide the necessary critical mass of buyers (bidders) and sellers to set market prices for their goods. The more bids that come in, the more competition there is, and higher prices are more possible. On eBay, the seller can set an open minimum bid that serves as the starting price for the auction. Additionally, the seller also has the option of setting a

hidden reserve price below which she is not obligated to sell. Once the bidding level exceeds the reserve, bidder is indicated.

Another interesting system is eAuctionHouse, which supports several important features including bidding via software agents and combinatorial auctions that explained below. Sandholm et al. [25] states that eAuctionHouse is, to their knowledge, the first and currently only Internet auction site that supports combinatorial auctions [22, 23, 24, 26, 27, 31]. They also build another system called Nomad, which is a mobile agent system, and integrate it with eAuctionHouse. With the Nomad system, mobile agents travel to the eAuctionHouse site and practice in auctions on the user's behalf [25].

Negotiation Model C

In this category the negotiation is bilateral. There are fewer existing systems in this category. Below is a list of such systems.

Kasbah is one of the well known web-based multi-agent consumer-to-consumer transactions system, which has buyer (bidder) and seller agents [12, 18]. Negotiations between Kasbah buying and selling agents are bilateral, competitive and straightforward. A user who wants to buy or sell a good creates an agent. To initialize an agent, a user inputs to the system what s/he wants to buy or sell, the desired price, the highest acceptable price for buying agents, the lowest acceptable price for selling agents, and the date s/he wants the transaction to be completed. The user then may choose for his agent one of several predefined negotiation strategies, gives it some strategic direction, and sends it off into a centralized agent marketplace. Kasbah agents seek out potential buyers or sellers and negotiate with them on behalf of their owners [12, 18].

Tete-a-Tete is another important system that provides a unique negotiation approach to retail sales [18]. Tete-a-Tete's consumer owned shopping agents merchant owned sales agents cooperatively negotiate across multiple terms of a transaction like warranties,

delivery times, services contacts, return policies, gift services, and other merchant value added services, which is different many of the other online negotiation systems that competitively negotiate over price. Tete-a-Tete's negotiations include an exchange of XML-based proposals, critiques, and counter proposals. A shopping agent may receive proposals from multiple sales agents. The shopping agent evaluates and orders them based on how they fit their owner's preferences. [18]

Lee et al. propose an auction agent system that is called MoCASS (Mobile collaborative auction agent system) [17]. MoCAAS uses a collaborative mobile agent and brokering mechanism, which mediates between the buyer and seller and executes bidding asynchronously and autonomously. It consists of five main components, which are buyer agent, broker agent, bid agent, and auctioneer agent. In MoCASS when a new auctioneer agent is created, it registers itself with a broker agent. The buyer submits the item's identity and reserve-price to the buyer agent. The buyer agent submits the data received to the broker agent. The broker agent searches for a recommendable auctioneer agent and computes the expected price, then returns the results to the buyer agent. The buyer selects an auctioneer agent among the auctioneer agents recommended by the broker agent. The buyer agent creates the bid agent, and then dispatches it to the selected auctioneer agent. The dispatched bid agent registers itself with the auctioneer-agent, reads the auction information from the auctioneer agent's blackboard, and executes the bidding autonomously. The buyer agent receives the intermediate auction information from the bid agent, and controls the bid agent. When the auction ends, the auctioneer agent unregisters itself with the broker agent, and the bid agent sends a 'win' or 'fail' message to the buyer agent [17].

1.2 Problem Statement

After the review of negotiation models and some of the previous work done on e-marketplaces, it is seen that auctions are one of the most successful negotiation protocol and account for an enormous volume of transactions on the Internet. In fact it should be noted that fixed-price sale is a special case of negotiation. Benyoucef et al. describe the

most basic form of negotiation as no negotiation at all i.e. fixed-price sale [2]. Most e-marketplaces operate using this principle. Auctions are a bit more complex, but auctioning is the most popular negotiation strategy at present. Auctions are also a critical enabler for negotiations in agent-based electronic commerce environments. Some agent environments focus on the intelligence of agents and concentrate on teaching the agents effective strategies for negotiation. Sandholm et al. [25] also state that electronic auctions are emerging as one of the most successful e-commerce technologies among the current business models. Courcoubetis et al. [4] describe the advantages of the auctions as, simplicity in determining market-based prices, and efficiency, since as stated if the auction is properly designed, goods can be acquired by those that value them most. Auctions may lead to higher revenues compared to traditional methods of selling goods.

There are many possible ways to classify auctions such as ascending or descending, single-item or multi-item, single or double, sealed-bid or open-cry, sequential or parallel, and etc [3,5,19,21,22,23,24,26,27,31,32]. More information about auctions can be seen in chapter 2. But as Tennenholtz [31] points out, the design of auctions introduces deep problems and challenges both from game-theoretic and computational perspectives.

In an auction, a seller sells goods to several potential buyers. Determining the auction's winner and its payment is trivial in single-item (i.e. single-object) auctions. The problem is also computationally tractable in multi-item (homogeneous or heterogeneous) auctions when agents' valuations for different items are additive, meaning that total valuation can be determined in an additive manner by their valuations for single items. But the problem starts when bidder (buyer) agents have preferences over bundles, i.e. a bidder's valuation for the bundle need not to be equal to the sum of his valuations of the individual items in the bundle, that is valuation need not to be additive. This problem is referred to as the *combinatorial auction problem* and it is generally intractable. Determining the winners in order to maximize revenue is NP-complete [3, 19, 23, 24, 26, 31, 32].

Examples of goods which are thought to exhibit such a property including procurement of indirect materials (like PC, printer, UPS), logistics marketplaces, airport landing slots, land parcels, oil leases, shipping space, markets for trucking services, electricity market, electromagnetic spectrum licenses, bandwidth allocations [4,30], and etc. In fact there are many other practical conditions where we need to use combinatorial auctions e.g. when buying interrelated items. For example, a vacation package can be considered as consisting of three items: a transportation ticket, a hotel room, and a ski trip. The three items are obviously interrelated since the consumer would have to travel to the location where the ski trip journey initiates on the date of the trip [2]. Another interesting example is the study of Hunsberger et al. [14]. In their study Hunsberger et al. present a mechanism based on a combinatorial auction that a group of agents may use to solve the initial-commitment decision problem (ICDP). ICDP occurs when rational, autonomous agents encounter an opportunity to collaborate on some group activity and as a result they must decide whether to commit to doing that activity [14].

When we survey some of the previous works that are done on the multi-item auctions in which bidders have preferences over bundles (i.e. combinatorial auction problem) considering the computational aspects, we see that the previous works can be categorized into three parts. One part tries to solve the unrestricted problem using search algorithms [23, 24, 26], second part tries to solve the problem using OR (Operation Research) methodologies and/or deals with the identification of tractable cases of the combinatorial auctions problem [31, 32] and the last part deals with the usage of sequential or simultaneous auctions in order to solve the combinatorial auctions problem [3, 4, 19].

In this study, we focus on combinatorial auctions problem; especially multi-item multi-unit auctions in which bidders have preferences over bundles and their valuation for the bundle is not additive. After reviewing the previous works, we decided to build three different models using different approaches of the previous works. The performance of each model is compared and analyzed using computer simulation. We model bandwidth

allocation among multiple users, since bandwidth allocation is one of the well known examples where preferences of bundles are involved. It can be modeled as multi-item multi-unit auctions which is one of the most complex situations that occur in combinatorial auction problem. Based on the performance of the models and some other situations, one of the models is chosen and a simple web server is built which has the ability of performing combinatorial auction, and can be used for online auctions and e-procurement systems. The details can be found in chapter 3.

1.3 Road Map

Chapter 2 provides a brief overview of the general concepts used throughout the thesis. Also in chapter 2 the review of literature can be found.

Chapter 3 describes the models that are built, gives information about the computer simulation, and the simple web service tool that is built. Also in this chapter, the improvements of the thesis will be provided in detail.

Chapter 4 gives information about the simulation results.

Chapter 5 presents discussions, future work and conclusion.

CHAPTER 2

Review of Literature

2.1 Introduction to Auctions

A negotiation mechanism is mainly a protocol within which agents interact to determine a contact. Auctions form a general class of such protocols. McAfee and McMillan defined auctions as: “An auction is a market institution with an explicit set of rules determining resource allocation and prices on the basis of bids from the market participants.” [34].

Auctions are popular, distributed and autonomy-preserving ways of allocating items such as goods, resources, services, etc. among agents. Auctions are relatively efficient in terms of process and outcome [24]. Klemperer [16] points out that, auctions provide a very valuable testing-ground for economic theory -especially for game theory with incomplete information- since auctions are simple and well-defined economic environments. Auctions also have many practical computer science applications. By the emergence of electronic commerce and low transaction costs on the Internet, an interest in the design of new auction mechanisms has been seen. Several successful web sites exist for buying and selling items using auction protocols like eBay. Also recently many researchers in computer science, economics, business and game theory have presented many valuable studies on the subject of e-commerce, e-negotiations, online auctions, and auction theory.

In the following sections auction theory, different types of auctions and basic terminology will be explained in more detail.

2.2 Auction Settings

Sandholm describes three qualitatively different auction settings that depend on how an agent's value of the item is formed. They are private value, common value, and correlated value auctions [9].

In private value auctions, the value of the good depends only on the agent's own preferences. As Sandholm explains the key point is that the winning bidder will not resell the item or get utility from showing it off to others, since in such cases the value would depend on other agents' valuations (valuation is the monetary equivalent of expected utility).

In common value auctions, an agent's value of an item depends entirely on other agents' values of it, i.e. the agents' value is affected by learning any other agent's preferences or information contrast to the private value auctions. For example, auctioning of treasury bills fit in this category [9,16].

In correlated value auctions, an agent's value depends partly on its own preferences and partly on others' values. For example, a negotiation within a contracting setting best fits in this category.

2.3 The Standard Auction Types

Four basic types of auctions that are generally used and analyzed are the English (first-price open-cry, or ascending) auction, Dutch (descending) auction, first-price sealed-bid auction, and Vickrey (second-price sealed-bid) auction. In this section while describing the rules of the standard auction types, single-item auctions are given. Multi-

item auctions and differences from single-item auctions can be seen in the next sections.

In the English (first-price open-cry) auction, each bidder is free to raise his bid. This auction can also be called “ascending auction”. When no bidder is willing to raise anymore, the auction ends, and the highest bidder wins the item at the price of his bid. Sandholm [9] describes an agent’s strategy as a series of bids as a function of his private value, his prior estimates of other bidder’s valuations, and the past bids of others. In private value English auctions, a bidder agent’s dominant strategy is to always bid a small amount more than the current highest bid, and stop when his private value price is reached. In correlated value auctions, the rules are often varied to make the auctioneer increase the price at a constant rate or at a rate he thinks appropriate. Also, sometimes open-exit is used where a bidder has to openly declare exiting without a re-entering possibility which provides the other bidders more information regarding the agent’s valuation. As can be seen English auctions i.e. ascending auctions can occur in multi rounds, i.e. involves iterations. Also in English auctions, especially in correlated or common value auctions, the problem of waiting other’s bids to see variation in price, can occur [23].

In the Dutch (can also be called as descending) auction, the seller begins at a high price and incrementally lowers until some bidder signals acceptance. The Dutch auction is strategically equivalent to the first-price sealed-bid auction. Because in both games, an agent’s bid matters only if it is the highest, and no relevant information is revealed during the auction process. Like English auctions, Dutch auctions can also occur in multi rounds, i.e. involve iterations. The Dutch auction got its name from the Dutch flower auction, used in the sale of flowers in the Netherlands. Dutch auction has been used traditionally for selling single objects such as works of art or single lots of a good such as cut flowers, fish, etc. Dutch auctions are efficient in terms of real time since usually the auctions are very short so that a lot of merchandise can be sold.

In the first-price sealed-bid auction, each bidder submits one bid without knowing the others' bids. The highest bidder wins the item and pays the amount of his bid. An agent's strategy is his bid as a function of his private value and prior beliefs of others' valuations. In general there is no dominant strategy for bidding in this auction. An agent's best strategy is to bid less than his true valuation, but how much less depends on what the others bid. The agent would want to bid the lowest amount that still wins the auction (this amount should not exceed his valuation). This auction type does not involve iterations.

In the Vickrey (also called second-price sealed-bid) auction, each bidder submits one bid without knowing the others' bids. The highest bidder wins, but at the price of the second highest bid. An agent's strategy is his bid as a function of his private value and prior beliefs of others' valuations. Vickrey proves that a bidder's dominant strategy in a private value auction is to bid his true valuation [9]. This auction also involves no iteration as first-price sealed-bid auction.

One problem with all four of the auction protocols is that they are not collusion proof, i.e. the bidders may coordinate their bid prices so that the bids stay artificially low. In this time, the bidders get the item at a lower price than they normally would. Collusion agreements may be done more easily in the English and the Vickrey auctions, so the first-price sealed-bid and the Dutch auctions are more preferable considering collusion agreement. Rasmusen shows [9] this in the following example:. Let bidder Smith have value 20, and every other bidder have value 18 for the auctioned item._ Say that the bidders collude by deciding that Smith will bid 6 and everyone else will bid 5. In an English auction this is self-enforcing, because if one of the other agents exceeds the agreed price 5, Smith will observe this and will increase his bid up to 20, and the cheater will not gain anything from breaking the coalition agreement. In the Vickrey auction, the collusion agreement can just as well be that Smith bids 20, and others bid 5, because Smith will get the item for 5 anyway (second-price). Bidding 20 removes the incentive from any bidder to break the coalition agreement. On the other hand, in a first-price sealed-bid auction, if Smith bids anything below 20, the other agents have an

incentive to bid higher than Smith's bid because that would cause them to win the auction. The same holds for the Dutch auction, meaning that breaking coalition agreement in Dutch and first-price sealed-bid auctions are easier. However, for collusion to occur under the Vickrey auction, the first-price sealed-bid auction, or the Dutch auction, the bidders need to identify each other before the submission of bids. On the other hand, in the English auction this is not necessary, because the bidders identify themselves by shouting bids. In this work it is assumed that no coalition agreement will occur.

A second problem that can occur in the auctions is the insincerity of the auctioneer, i.e. seller. This can be a problem in the Vickrey auction. The auctioneer may overstate the second highest bid to the highest bidder unless that bidder can verify it. An overstated second offer would give the highest bidder a higher bill than he would receive if the auctioneer were truthful. Rothkopf et al. states that [9] cheating by the auctioneer has been suggested to be one of the main reasons why the Vickrey auction protocol has not been widely adopted in auctions among humans. To solve the problem, cryptographic electronic signatures could be used by the bidders so that the auctioneer could actually present the second best bid to the winning bidder, and would not be able to alter it. The other three auction protocols English, Dutch, and first-price sealed-bid, do not suffer from lying by the auctioneer because the highest bidder gets the item at the price of his bid. In this work, it is assumed that there is no lying auctioneer.

2.4 Evaluation Criteria

Sandholm [9] lists many types of criteria that can be used to evaluate negotiation protocols. Auctions can also be evaluated using the criteria below. Takahashi et al. [30] define the three desirable conditions for the auctions to satisfy as Individual Rationality, Pareto Efficiency, and Incentive Compatibility.

2.4.1 Social Welfare

Social welfare is the sum of all agents' payoffs or utilities in a given solution. It measures the global good of the agents. It can be used as a criterion for comparing alternative mechanisms by comparing the solutions that the mechanisms lead to. In fact when measured in terms of utilities, this criterion is a bit arbitrary, since each agent's utility function depends on the agent itself.

2.4.2 Pareto Efficiency

Pareto efficiency is another solution evaluation criterion. An allocation is Pareto efficient if there is no other allocation in which some other individual is better off and no individual is worse off. So Pareto efficiency measures global good, and it does not require a utility comparison that depends on each agent as social welfare does. Social welfare maximizing solutions are a subset of Pareto efficient ones.

2.4.3 Individual Rationality

Participation in a negotiation is individually rational to an agent, if the utility to the agent when the resource is allocated to him is higher than the utility when he is not allocated the resource. A mechanism is individually rational if participation is individually rational for all agents. Only individually rational mechanisms are favorable i.e. if the negotiated solution is not individually rational for some agent, that self-interested agent should not participate in that negotiation.

2.4.4 Stability

If a self-interested agent is better off behaving in some other manner than desired, it will behave in that manner so, among self-interested agents, mechanism should be designed to be stable. The mechanism should motivate each agent to behave in the desired manner. Sometimes it is possible to design mechanisms with dominant

strategies. This means that an agent is best off by using a specific strategy no matter what strategies the other agents use. However, often an agent's best strategy depends on what strategies other agents choose. In such settings dominant strategies do not exist, and other stability criteria are needed and the most basic one is the *Nash equilibrium*.

2.4.5 Computational Efficiency

Mechanisms should be designed so that when agents use them, as little computation is needed as possible. The mechanisms with the lowest computational overhead have been generally preferred to others.

2.4.6 Distribution and Communication Efficiency

Distributed protocols should be designed that prevents a single point of failure and a performance bottleneck. Also the mechanisms should minimize the amount of communication that is required to converge on a desirable solution. In some cases these two goals conflict.

2.4.7 Incentive Compatibility

As Takahashi et al. [30] define incentive compatibility as a mechanism where truthful bidding is the only dominant strategy. In an allocation satisfying incentive compatibility, false bidding does not have any effect on the agent; therefore, an effective allocation is always realized.

2.5 Classification of Auctions

As we have seen in the standard auction types section, there are four basic types of auctions that are generally used: English (first-price open-cry, or ascending) auction, Dutch (descending) auction, first-price sealed-bid auction, and Vickrey (second-price

sealed-bid) auction. But there are many other types of auctions, and they can be classified from many different perspectives. Wurman et al. [34] classify the auctions as follows:

Single or double auctions: In single auctions (i.e. single-sided auctions) the bidders are either of type "buyers" or of type "sellers". English, Dutch, first-price sealed-bid, and second-price sealed-bid auctions are single auctions. On the other hand, double auctions are double-sided. They allow multiple buyers and sellers at once. The continuous double auction (CDA), a general model for commodity and stock markets, and clearing market or call market, markets that aggregate bids over time and clear at schedule intervals, are showed as example for double auctions.

Sealed-bid or out-cry: In sealed-bid auctions the bids submitted by the participants are not known until the auction closes. On the other hand, in the out-cry (or we can call open-cry) auctions the bids are made public at the time they are made.

Ascending or descending: In the ascending auctions the bids begin low and keep increasing until a deal is made. In the descending auction, the seller (or the auctioneer) begins with a higher price and lowers it continuously until someone bids on it.

Their classification can be seen better in Figure 2, which is taken from [34]

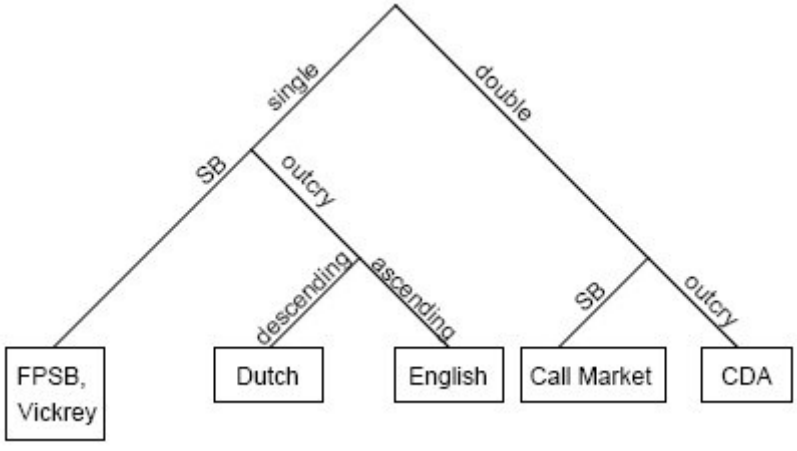


Figure 2 A classification of classic auction types (from [34])

A more complete classification can be found by looking at the E-negotiations Web Page (http://enegotiations.wu-wien.ac.at/) and in [2]. They identify four aspects of e-negotiations as: the "people" aspect which deals with the participants in the negotiation, the "goods" aspect which is dedicated to the object of the negotiation, the "process" aspect which is concerned with the negotiation protocol to be followed by the participants, and finally the "evaluation criteria" aspect which covers the ways to evaluate a negotiation process.

In the people aspect, e-negotiations are classified as number of parties (two or more parties participating in a negotiation), bidding activity (single-sided or double-sided), admission (seller-open (buyer-open) meaning that there is no restriction on the admission of sellers (buyers) to participate in the negotiation, or seller-closed (buyer-closed) meaning that there exist restrictions), collusion (collusive or non-collusive, where collusion refers to agreements between buyers and/or sellers in order to achieve mutual benefits) , and Anonymity(anonymous or non-anonymous).

In the goods aspect, they classify the e-negotiations as number of items (single-item or multi-item), number of attributes (single-attribute or multiple-attribute), and homogeneity (items can be homogeneous or heterogeneous).

In the process aspect, e-negotiations are classified as reverse versus forward, Single-phased versus multi-phased (in a single-phased negotiation the rules are the same from the beginning to the end of the negotiation, but in multi-phased negotiation rules are allowed to change.), single-stage versus multi-stage (in a single-stage negotiation all the attributes of the item are negotiated at the same time, but in a multi-stage negotiation the parameters of the negotiation can be changed after a stage, enabling another round of bidding), synchronized versus sequential versus combinatorial (synchronized is also called simultaneous or parallel auction), open-cry versus sealed-bid, information revelation (transparency of the market and the amount of information available in the negotiation process), agent-mediated versus manual bidding, discriminative versus non-discriminative, and non-repudiation.

In the evaluation criteria aspect, they classify the e-negotiations as incentive compatibility, computational complexity, convergence, speed of convergence, stability, integrative versus distributive, efficiency, fairness, and correctness.

After analyzing these classification aspects, we merge the findings from the Wurman et al. [34] and Benyoucef et al. [2] and we decide to classify the auctions using the following aspects and criteria, as shown below.

2.5.1 People Aspect

People aspect deals with the participants in the auction.

Bidding Activity:

An auction can be single-sided meaning that only buyers or only sellers are allowed to submit bids, or it can be double-sided meaning that multiple buyers and multiple sellers are both allowed to submit bids. Dawid [5] describe the importance of double auction models for the understanding of price formation in markets. He studies the learning behavior of a population buyers and a population of sellers whose members are repeatedly randomly matched to engage in a sealed-bid double auction by using GA (Genetic Algorithm).

2.5.2 Goods Aspect

Goods aspect is used to classify the object of the auction.

2.5.2.1 Number of items:

Items in an auction can be goods or services. If one item is subject to auction, then the auction is said to be a single-item auction. The standard auction types discussed before were also explained from the single-item auction perspective. If more than one item is subject to the auction, then the auction is called a multi-item auction.

2.5.2.2 Number of attributes:

In an auction, if an item is negotiated on one attribute, which is generally the price of the item, these auctions can be called as single-attribute auctions. If the item is negotiated on multiple attributes of a deal (e.g., price, quality, terms of delivery), then these auctions can be called as multiple-attribute auctions. Throughout this work we used only single-attribute auctions.

2.5.2.3 Homogeneity:

In multi-item auctions items can be homogeneous (i.e. indistinguishable) or heterogeneous (i.e. distinguishable) (e.g., a plane ticket and a ski trip). Also in heterogeneous multi-item auctions some items may have multiple units (e.g. *i* units of *x* item, *j* units of *y* item, also in section x bandwidth allocation will be modeled as heterogeneous multi-item multi-unit auction)

2.5.3 Process Aspect

Process aspect deals with the auction's mechanism.

2.5.3.1 Reverse versus forward (Descending vs. Ascending):

In a reverse auction or that is called as descending auctions the bids go down, i.e. the seller (or the auctioneer) begins with a higher price and lowers it continuously until someone bids on it. In a forward auction the bids go up, i.e. the bids begin low and keep increasing until a deal is made; here the bids are made by the buyer agents. Online reverse auctions are very popular in e-business and e-sourcing (electronic sourcing) area. In online reverse auctions, multiple supplier (i.e. sellers in the industry) bid for a contract from a buyer (the buyer submits a request for purchase (RFP)) for selling goods and/or services. Reverse and forward auctions are generally include multiple rounds of bidding. According to Jap [15] the popularity of online auctions can be attributed to three factors, which are:

- Creating immediate financial savings.
- Creating many process efficiencies.
- Enabling capabilities of emerging technologies .

More information about online reverse auctions and their use in business sector can be found in [15].

2.5.3.2 Open-cry versus sealed-bid:

In an open-cry auction the bids made by a participant are known by the other participants. Depending on the type of the auction the bids can either go up or down (ascending/forward or descending/reverse). In a sealed-bid auction the participants make secret bids. At the close of the auction the winning bid is determined and announced. In a sealed-bid auction the bids can go up and down.

2.5.3.3 Rounds of auction:

An auction can include multiple bidding rounds or it can single round. Generally ascending auctions (English auctions or forward auctions) and descending auction (Dutch auctions or reverse auctions) include multiple bidding rounds, i.e. involves iterations. But the first-price sealed-bid auctions and the Vickrey (also called second-price sealed-bid) auctions involve no iteration.

2.5.3.4 Discriminative versus non-discriminative:

Usually, once the bidding phase is over, the bidder with the highest bid gets the item being auctioned, but the price she pays could be the same as what she bid or lower. In a Discriminative Auction (also known as Yankee Auction or first-price auction), the winners pay what they bid. In a non-discriminative auction, people with winning bids pay the price paid by the winning bidder with the lowest bid (for the sale of multiple similar items). Finally, in a Vickrey Auction (also referred to as second price sealed bid auction) the winner pays the price bid by the second highest bidder (for the sale of a single item).

2.5.3.5 Non-Repudiation:

In a negotiation, the participants are allowed or not to break their commitments i.e. bidders can or cannot retract (withdraw) submitted bids

2.5.3.6 Sequential versus Parallel versus combinatorial:

These processes occur in multi-item (homogeneous or heterogeneous) auctions.

2.5.3.6.1 Sequential Auctions

In a sequential auction, the items are auctioned one at a time, i.e. n items are auctioned individually one after the other. Determining the winners in such protocols is easy because that can be done by picking the highest bidder for each item separately. However if a bidder has preferences over bundles (combinations of items), bidding in such auctions is difficult. To determine his valuation for an item, the bidder needs to guess what items he will receive in later auctions, which requires speculation of what the others will bid in the future because that affects what items he will receive. Furthermore, what the others bid in the future depends on what they believe of others, etc. This counter speculation introduces computational cost. In auctions with large number of items, such look-ahead in the game tree is intractable [23, 24]. Moreover, even if look-ahead were computationally manageable, usually some uncertainty remains about the others' bids because agents do not have exact information about each other. Also as said before when bidders have preferences over bundles, that means bidder's valuation for the bundle need not to be equal to the sum of the his valuations of the individual items in the bundle, i.e. valuation need not to be additive so to determine valuation for an item from the bundle is very hard, also from this perspective. These situations can lead to inefficient allocations where bidders do not get the combinations that they want and do get combinations that they do not want.

2.5.3.6.2 Parallel Auctions

In a parallel (also called simultaneous or synchronized) auction there are n items and the items are opened for auction in parallel. A participant can make m distinct bids on m distinct items (m not greater than n). The m bids are made simultaneously. The auction usually runs multiple rounds of sealed bids, announcing the bids after each round. Parallel auctions have the advantage that the others' bids partially signal to the bidder about what the others' bids will end up being for the different items, so the uncertainty and the need for look-ahead is not as drastic as in a sequential auction. However, the same problems still exist. For example, when bidding for an item, the bidder does not know his valuation because it depends on which other items he wins (and also if he has a preferences over bundles, this is not additive as said before), which in turn depends on how others will bid.

In parallel auctions, an additional difficulty arises. Each bidder would like to wait until the end to see what the going prices will be, and to optimize his bids so as to maximize payoff given the final prices. Because every bidder would want to wait, no bidding would occur. As a solution to this problem, activity rules have been used [19]. But the equilibrium bidding strategies in such auctions are not game-theoretically known [23].

Examples of Parallel Auctions

This idea of simultaneous auctions was first seen in 1994 by the Federal Communication Commission (FCC). FCC uses such a format to sell licenses to use bands of radio spectrum in the United States. The structure adopted was that of a *simultaneous ascending auction* [19].

A simultaneous ascending auction is an auction for multiple items in which bidding occurs in rounds. At each round, bidders simultaneously make sealed bids for any items in which they are interested. After the bidding, round results are posted. For each item, these results consist of the identities of the new bids and bidders as well as the

"*standing high bid*" and the corresponding bidder. The initial standing high bid for each item is given (it may be zero) and the "*corresponding bidder*" is the auctioneer. As the auction progresses, the new standing high bid at the end of a round for an item is the larger of the previous standing high bid or the highest new bid and the corresponding bidder is the one who made that bid. In addition to the round results, the minimum bids for the next round are also posted. These are computed from the "standing high bid" by adding a predetermined bid increment. For spectrum licenses, the increments are typically the larger of some fixed amount or a fixed percentage of the standing high bid.

A bid represents a real commitment of resources by the bidder. In the most common version of the rules, a bidder is permitted to withdraw bids, but there is a penalty for doing so: if the selling price of the item is less than the withdrawn bid, the withdrawing bidder must pay the difference. In other applications, bid withdrawals are simply not permitted.

A bidder's eligibility to make new bids during the auction is controlled by the "*activity rule*." The rule is based on a "quantity" index, such as spectrum bandwidth or population covered by a license that roughly corresponds to the value of the license. During the auction, a bidder may not have active bids on licenses that exceed its *eligibility*, measured in terms of the index.

At the outset of the auction, each bidder establishes its *initial eligibility* for bidding by making deposits covering the quantity of spectrum for which it wishes to be eligible. During the auction, a bidder is considered *active* for a license at a round if it makes an eligible new bid for the license or if it owns the standing high bid from the previous round. At each round, a bidder's *activity* is constrained not to exceed its eligibility. If a bid is submitted that exceeds the bidder's eligibility, the bid is simply rejected.

The auction is conducted in a sequence of three *stages*, each consisting of multiple rounds. The auction begins in stage 1 and the administrator advances the auction to

stage 2 and later to stage 3 when there are two or more consecutive rounds with little new bidding.

There are several different options for rules to close the bidding that were filed with the regulator. One proposal, made by McAfee [19], specified that when a license had received no new bids for a fixed number of rounds, bidding on that license would close. That proposal was coupled with a suggestion that the bid increments for licenses should reflect the bidding activity on a license. Another proposal, made by Wilson and Milgrom [19], specified that bidding on all licenses should close simultaneously when there is no new bidding on any license.

When the auction closes, the licenses are sold at prices equal to the standing high bids to the corresponding bidders.

A second example that uses parallel auctions' strategy is the work of Courcoubetis et al. [4]. They design their auction mechanism for bandwidth allocation over paths (as said before bandwidth allocation is also an area where preferences over bundles situation occurs). Their mechanism consists of a set of simultaneous multi-unit Dutch (i.e. descending price) auctions (MIDAS), one per link of the network.

2.5.3.6.3 Methods for fixing inefficient allocations in sequential and parallel auctions

In sequential and parallel auctions, the computational cost of look-ahead and counter speculation cannot be recovered, but attempts to fix the inefficient allocations that stem from the uncertainties discussed above, have been developed.

One approach is to set up an after market where the bidders can exchange items among themselves after the auction has closed. This approach can prevent some inefficiency, but it may not lead to a Pareto efficient allocation in general and even if it does, it may be impractical for large number of exchanges [23].

Another approach is to allow bidders to retract their bids if they do not get the combinations that they want. For example, in the FCC's bandwidth auction, the bidders were allowed to retract their bids as said before. In case of a retraction, the item was opened for re-auction. If the new winning price was lower than the old one, the bidder that retracted the bid had to pay the difference. This guarantees that retractions do not decrease the auctioneer's payoff. But, this brings to the retracting bidder considerable risk [23]. Sandholm and Lesser explain *Leveled Commitment* protocol that can be use to eliminate this risk [23]. This protocol allows the bidders to decommit but it also allows the auctioneer to decommit. A bidder may want to decommit for example if he did not get the combination that he wanted but only a subset of it. The auctioneer may want to decommit for example if he believes that he can get a higher price for the item later on.

Each one of the methods above can be used to implement bid retraction before and/or after the winning bids have been determined.

All these approaches try to fix inefficient allocations achieved in sequential or parallel auctions, adapting a basically non-combinatorial process in some manner to take account of combinatorial values. It would be desirable to get efficient allocations right away in the auction itself, so no fixing would be required. Also since some of the inefficiencies are fixed, generally but the solution may not be a optimum or a Pareto efficient allocation as said before. Combinatorial auctions are an approach to achieve efficient allocations in the first step. Here bidders can place bids on combinations of goods which allows expressing dependencies and complementarities between goods.

2.5.3.6.4 Combinatorial Auctions

Combinatorial auctions are promising. Because they can be used in case of multiple identical goods. These mechanisms are especially useful in situations where one has to assign multiple heterogeneous goods simultaneously and bidders have preferences over

different combinations of goods (i.e. preferences over bundles and valuation of the bundles are not additive). Combinatorial (also called bundled) auctions allow for a participant to make a single bid for m items. In a combinatorial auction, bidders may place bids on combinations of items. This allows the bidders to express complementarities between items instead of having to speculate into an item's valuation the impact of possibly getting other, complementary items. For example, the FCC sees the desirability of combinatorial bidding in their bandwidth auctions, but so far combinatorial bidding has not been allowed due to perceived intractability of winner determination. Determining the winners in order to maximize revenue is NP-complete. [3, 19, 23, 24, 26, 31, 32]

Previous work on computational aspects of combinatorial auctions can be divided into two parts. One part try to solve the unrestricted problem using search algorithms [23, 24, 26], second part try to solve the problem using OR (Operation Research) methodologies and/or deals with the identification of tractable cases of the combinatorial auctions problem and/or find an approximate solution [31, 32]. (Some works deals with using sequential or simultaneous auctions in order to solve the combinatorial auctions problem as stated above, but they do not consider the bidder's preference over bundles)

2.6 Winner Determination in Combinatorial Auctions

The determination of winners in non-combinatorial auctions is easy. Determining what items each bidder gets, can be done by picking the highest bidder for each item separately. This takes O(nm) time, here n is the number of bidders and m is the number of items in the auction. In these auctions, determining the Vickrey price of each item, can also be done O(nm) time by simply finding the second highest bid for each item.

The determination of winners in combinatorial auctions is more difficult. Let M be the set of items to be auctioned. Then any bidder agent, i, can place a bid, $b^i(S) > 0$, for any combination $S \subset M$.

Clearly, if several bids have been submitted on the same combination of items, and if there are at most one copy of each item, for winner determination purposes the bid with the highest price can simply be kept and the others can be discarded as irrelevant. (The formulation for multi-item multi-unit case will be also given.)The highest bid price for a combination is

$$b(S) = \max_{i \in N} b^i(S) \quad (1)$$

If agent i has not submitted a bid on combination S, then $b^i(S) = 0$ (Sandholm states that this assignment need not actually be carried out as long as special care is taken of the combinations that received no bids [24]. Also he shows more compact representation by not using the combinations that received no bids which will also be explained)

Sandholm explains the winner determination in a combinatorial auction problem as a goal to find a solution that maximizes the auctioneer's revenue given that each winning bidder pays the prices of her winning bids.

$$\max_{W \in A} \sum_{S \in W} b(S) \quad (2)$$

where W is a partition, that is a set of subsets of items so that each item is included in at most one of the subsets. Formally, let $S = \{S \subseteq M\}$. Then the set of partitions is

$$A = \{W \subseteq S \mid S, S' \in W \rightarrow S \cap S' = \emptyset\} \quad (3)$$

The winner determination problem can also be formulated as an integer programming problem where the decision variable $x_S = 1$ if the (highest) bid for combination S is chosen to be winning, and $x_S = 0$ if not[1].

Formally:

$$\max \sum_{S \subset M} b(S) x_S$$

$$\text{s.t.} \sum_{i \in S} x_S \leq 1 \quad \forall i \in M \tag{4}$$

$$x_S = 0, 1 \ \forall S \subset M$$

Formulation 4 assumes that there is at most on copy of each item and no item in M is assigned to more that 1 bidder, by the following constraint.

$$\sum_{i \in S} x_S \leq 1 \quad \forall i \in M$$

Other Integer Program models can be seen in the next section. One way to optimally solve the winner determination problem is to enumerate all exhaustive partitions of items [23,24] But Sandholm [24] indicate that the number of exhaustive partitions is $O(m^m)$ and $w(m^{m/2})$. Therefore it is possible to enumerate all of the exhaustive partitions only if the number of items in the auction is very small.

2.6.1 Integer Programming Formulation Approach to Combinatorial Auctions

Combinatorial auction problem (CAP) can be formulated as an Integer Program. In the some of the previous studies' integer program formulations every subset S of the items being auctioned are considered, but in the coming sections other formulations which consider only the given bids will be also explained.

Vries et al. in their survey gives examples of integer formulations of CAP in their study. Formulation (4) is one of the formulations that they describe [32]. Let's call

[1] In this formulation every subset S of the items being auctioned are considered, but in the coming pages other formulations can be seen in which only the bids will be considered instead of every possible subset.

formulation (4) as CAP1. They note that, CAP1 correctly models the CAP when the bid functions b^i are all superadditive[2], i.e. $b^i(A) + b^i(B) \leq b^i(A \cup B)$ for all $i \in N$ and $A,B \subset M$ such that $A \cap B = \emptyset$ which corresponds to the idea that the goods complement each other.But Vries et al. state that when goods are substitutes, i.e. $b^i(A) + b^i(B) > b^i(A \cup B)$ for some $i \in N$ and $A,B \subset M$, formulation 4 is incorrect. An optimal solution to CAP1 may assign sets A and B to bidder i and *incorrectly* record a revenue of $b^i(A)+b^i(B)$ rather than $b^i(A \cup B)$ to that allocation. This problem can be removed by the introduction of dummy goods, g, as Vries et al point out. The bidder is then instructed to replace the bids $b^i(A)$, $b^i(B)$ and $b^i(A \cup B)$ with $b^i(A \cup g)$, $b^i(B \cup g)$ and $b^i(A \cup B)$ and to replace M by $M \cup g$. In this way by the constraints of the integer programming formulation, if the set A is assigned to i then so is g and thus B cannot be assigned to i.

Another way to write the problem for (not necessarily superadditive) bids without explicitly involving dummy items, described by Vries et al. [32], is the following integer program. Let $y(S, j) = 1$ if the bundle $S \subseteq M$ is allocated to $i \in N$ and zero otherwise.

$$\max \sum_{j \in N} \sum_{S \subseteq M} b^j(S) y(S, j)$$

$$\text{s.t.} \sum_{i \in S} \sum_{j \in N} y(S, j) \leq 1 \quad \forall i \in M \tag{5}$$

$$\sum_{S \subseteq M} y(S, j) \leq 1 \quad \forall j \in N$$

$$y(S, j) = 0,1 \quad \forall S \subseteq M \,, \; j \in N$$

Let's call formulation (5) as CAP2. The first constraint ensures that overlapping sets of goods are never assigned and the second ensures that no bidder receives more than one subset. Again in this formulation CAP2 every subset S of the items being auctioned are

[2] In this study we deal with superadditive bids.

considered. Formulations CAP1 and CAP2 are an instance of the Set Packing Problem (SPP).

SPP include applications like switching theory, the testing of VLSI circuits, line balancing and scheduling problems where one wishes to satisfy as much demand as possible, without creating conflicts [32]. But as, Vries et al. state SPP is NP hard, and no polynomial time algorithm for the SPP is known. In the coming sections other different approaches to tackling the winner determination problem is discussed.

2.6.1.1 Dynamic Programming

Dynamic programming approach is discussed by Rothkopf et al. [23, 24].Based on the *b(S)* function, the dynamic programming algorithm determines for each set *S* of items the highest possible revenue that can be acquired using only the items in *S*. The algorithm proceeds systematically from the smallest sets to the largest. The needed optimal substructure property comes from the fact that for each set *S*, the maximal revenue comes either from a single bid *b(S),* or from the sum of the maximal revenues of two disjoint exhaustive subsets of *S*. For each *S*, all possible subsets (together with that subset's complement in *S* are tried.

Since the revenue maximizing solutions for the subsets need not be computed over and over again, but only once, it has many savings compared to exhaustive search. The dynamic programming algorithm takes $\Omega(2^m)$ and $O(3^m)$ steps and unfortunately this is still too complex to scale to large numbers of items. Also as Sandholm points out that the dynamic programming algorithm takes the same number of steps independent of the number of actual bids. This is because the algorithm generates each combination *S* even if no bids have been placed on *S*. [23, 24]

2.6.1.2 Generalized Vickrey Auction

Generalized Vickrey Auction (GVA) or the Vickrey-Clarke-Groves (VCG) is another well known mechanism applied to combinatorial auctions generalizes the second price

auction proposed by Vickrey [3]. GVA is an incentive compatible mechanism, in which true relevation is the dominant strategy for a bidder. GVA maximizes the sum of the declared utilities which are the true valuations of the bidders (incentive compatibility). Therefore the allocation maximizes the social welfare. It is also Pareto optimal. In other words, GVA assigns goods efficiently i.e. puts the goods in the hands of the bidder who values it most.

Let M denote the set of items, N be the number of bidders. The GVA algorithm works as follows as written in [32]:

1. Agent j reports v^j. (Here $v^i(.)$ represents the value to bidder i of a particular subset.)
2. The seller chooses the allocation that solves:

$$\begin{aligned} \text{V} = \max \ & \sum_{j \in N} \sum_{S \subseteq M} v^j(S) y(S, j) \\ \text{s.t.} \ & \sum_{i \in S} \sum_{j \in N} y(S, j) \leq 1 \quad \forall i \in M \\ & \sum_{S \subseteq M} y(S, j) \leq 1 \quad \forall j \in N \\ & y(S, j) = 0,1 \ \ \forall S \subseteq M \ , \ \ j \in N \end{aligned} \tag{6}$$

   This optimal allocation is called as y^*.

3. To compute the payment that each bidder must make let, for each $k \in N$,

$$\begin{aligned} \text{V}^{-k} = \max \ & \sum_{j \in N \setminus k} \sum_{S \subseteq M} v^j(S) y(S, j) \\ \text{s.t.} \ & \sum_{i \in S} \sum_{j \in N \setminus k} y(S, j) \leq 1 \quad \forall i \in M \\ & \sum_{S \subseteq M} y(S, j) \leq 1 \quad \forall j \in N \setminus k \\ & y(S, j) = 0,1 \ \ \forall S \subseteq M \ , \ \ j \in N \setminus k \end{aligned} \tag{7}$$

Denote by y^k the optimal solution to this integer program. Thus y^k is the efficient allocation when bidder *k* is excluded.

4. The payment that bidder *k* makes is equal to

$$V^{-k} - [V - \sum_{S \subseteq M} v^k(S) y^*(S,k)] \quad (8)$$

Bidder *k*'s payment is the difference in "welfare" of the other bidders without him and the welfare of others when he is included in the allocation. The payment made by each bidder to the auctioneer is non-negative. In fact this is not true in all economic environments.

If a seller were to adopt the GVA scheme, her total revenue would be [32]

$$\begin{aligned} &\sum_{k \in N} V^{-k} - \sum_{k \in N} [V - \sum_{S \subseteq M} v^k(S) y^*(S,k)] \\ &= \sum_{k \in N} \sum_{S \subseteq N} v^k(S) y^*(S,k) + \sum_{k \in N} (V^{-k} - V) \quad (9) \\ &= V + \sum_{k \in N} (V^{-k} - V). \end{aligned}$$

If the number of bidders N and number of items M is large, GVA is impractical to implement [3, 19, 32]. Since all the combinations are considered, the number of combinations is $2^M - 1$.

Biswas et al. describe that there are two problems with this GVA scheme.

1) GVA may not be budget balanced i.e. may yield low revenue for the seller in forward auction or very high price for the buyer in the reverse auction.
2) GVA requires optimal solution of N+1 allocation problems (where N is the number of agents) which are NP-hard.

The second problem has led to many approximate solution schemes and interesting auction algorithms. In he next sections two alternative solutions are described.

2.6.1.2.1 Iterative Dutch Combinatorial Auctions

Biswas et al. [3] suggest an iterative Dutch auction scheme to reduce the complexity of the two problems of GVA as stated above.

- In their iterative Dutch mechanisms the reserve prices for items are a natural outcome in each iteration, since as known GVA is not budget balanced, and setting the reserve prices for the items is difficult (since the agents bid for bundles instead of individual items).
- As known the second problem of GVA as stated above (i.e. GVA is NP-hard) depends on the size of the input. Therefore, in their algorithm, they reduce the time to solve the problem by dividing the one shot GVA into smaller GVAs in each iteration significantly. The overall solution obtained may not be optimal. But they show that the solutions obtained using these iterative schemes lie within provable worst case bounds. The time taken to solve the smaller GVAs in each iteration is much less than the time taken to solve the complete problem, since the solution time grows exponentially with size of the problem.

2.6.1.2.2 Adaptive User Selection Mechanism (AUSM)

Since it is infeasible to specify all relevant combinations in GVA, one idea to economize on computing power is to specify combinations as the auction progresses. One of the such proposal is based on a procedure called the "Adaptive User Selection Mechanism" (AUSM) which was developed in experimental economics laboratories for solving what the experimenters regarded as "difficult" resource allocation problems [19].

AUSM differs from the simultaneous ascending auction in a number of respects. First, allow bidding to take place continuously in time, rather than forcing bidders to bid simultaneously in discrete rounds. Second, in place of an activity rule, follow the experimenters' technique of using random closing times, which motivate bidders to be active before the end of the auction. Third, permit bids for combinations of items,

rather than just for individual items. Fourth, allow the use of a "*standby queue*" on which bidders may post bids that cannot, by themselves, displace existing bids but which become available for use in new combinations [19].

2.6.1.3 More Compact CAP Representation

As seen in the previous formulations of CAP every subset S of items being auctioned are considered, but in fact if no bid is received on some combination *S*, then those partitions *W* that include *S* need not be considered [24]. Sandholm notes that by capitalizing on this observation, one can restrict attention to *relevant partitions* [24]. The set of *relevant partitions* is:

$A' = \{W \in A \mid S \in W \rightarrow$ bid has been received on S $\}$ (10)

Meaning that we can restrict attention to the following set of combinations of items:

$S' = \{S \subseteq M \mid$ bid has been received on S$\}$ (11)

The number of relevant bids is n = | S' | . Now the winner determination can be formulated as the following integer program as stated by Sandholm [24]

$$\max_{x} \sum_{S \in S'} b(S) x_S \quad \forall S \in S' : x_S \in \{0,1\} \text{ and } \forall i \in M : \sum_{S \mid i \in S} x_S \leq 1 \tag{12}$$

Unfortunately as Sandholm states formulation (12) is also NP-Complete.[24].

Because of this NP-Completeness, some part deals with the identification of tractable cases of the combinatorial auctions problem. Previous results in this category have introduced a general technique for tackling the complexity of combinatorial auctions. Several of them have dealt with the problem of winner determination in combinatorial auctions as an integer programming [IP] problem, and considered linear programming [LP] relaxations of that problem for isolating tractable cases of the general problem [32]. Tennenholtz [31] expose and explore the use of b-matching techniques for the combinatorial auctions setup, and employ b-matching techniques in various ways in order to efficiently address several non-trivial instances of the combinatorial auctions problem.

In some of the previous works, they tried to solve the problem approximately. However it is shown by a reduction from the maximum clique problem, that no polynomial time algorithm for the winner determination problem for CAP can guarantee a solution that is close to optimum [24]. As reviewed in [24], certain special cases can be approximated slightly better.

Another approach is to restrict the allowable bids. For certain restrictions, which are severe in the sense that only a small fraction of the combinations can be bid on, it is seen that winners can be determined in polynomial time. Restrictions on the bids can cause some economic inefficiencies because bidders may not be able to bid on the combination they prefer [24, 26].

Lastly the third try to solve the unrestricted problem using their sophisticated search algorithm. This method was shown to work very well on average, scaling optimal winner determination up to hundreds of items and thousands of bids [23, 24, 26].

2.6.2 Search Solutions to CAP

One of the most well known solutions is generated by Sandholm , which try to solve the unrestricted problem using search, the details of it can be found in [24]. It is based on highly optimized search. The main characteristics of his method can be summarized as follows:

- The method allows bidding on all combinations in order to avoid all of the inefficiencies that occur in noncombinatorial auctions (The auctions that restrict the combinations. Those auctions lead to economic inefficiency and computational burden for the bidders that need to look ahead in a game tree and counterspeculate each other.).
- The method finds the optimal solution (given enough time), unlike the approximation algorithms.
- The method completely avoids loops and redundant generation of vertices, when searching the graph of allocations

- The method capitalizes heavily on the sparseness of bids. Sandholm points that in practice the space of bids is likely to be extremely sparsely populated. Sparseness of bids implies that the relevant partitions are a small subset of all partitions. Unlike the dynamic program, this algorithm only searches in the space of relevant partitions, so this method depends on the number of bids received, while in the dynamic program it does not.

The method uses tree search to achieve these goals. Since only the highest bid is kept for every combination of items for which a bid was received (all other bids are deleted), the inputs is a list of bids, one for each $S \in S$:

$$\{ B_1, ..., B_n\} = \{(B_1.S, B_1.\bar{b}), ..., (B_n.S, B_n.\bar{b})\} \quad (13)$$

Here $B_i.S$ is the set of items in the bid, and $B_i.\bar{b}$ is the price of the bid i.

Each path in the search tree consists of a sequence of disjoint bids, i.e., bids that do not share items with each other ($B_i.S \cap B_k.S = \emptyset$ for all bids i and k on the same path). As a result at any point in the search, a path corresponds to a relevant partition (feasible allocation).

Let U be the set of items that are already used on the path:

$$U = \bigcup_{j|B_j \text{ is on the path}} B_j.S \quad (14)$$

F be the set of free items:

$$F = M - U \quad (15)$$

A path ends when no bid can be added to the path and this occurs when for every bid, some of its items have already been used on the path ($\forall j, B_j.S \cap U \neq \emptyset$).

A tally g is kept of the sum of the prices of the bids on the path, as the search proceeds down a path:

$$g = \sum_{j|B_j \text{ is on the path}} B_j.\bar{b} \quad (16)$$

At every search node, the revenue from the path, i.e., the g-value, is compared to the best g-value found so far in the search tree to determine whether the current solution (path) is the best one so far. If so, it is stored as the best solution found so far. Once the search completes, the stored solution is an optimal solution (found up to that time if every path could not be investigated in the given time bounds, i.e. not enough time to finish the solution).

In this method the auctioneers also have the ability to keep some of the items. The auctioneer's possibility of keeping items is implemented as a preprocessing step in the algorithm by placing *dummy bids* of price zero on those individual items that received no bids alone.

In this method every relevant partition is represented in the search tree by exactly one path from the root to a node. This is done by restricting the children in the search tree according to the following criteria:

- Include the item with the smallest index among the items that have not been used on the path yet ($i^* = \min \{i \in \{1, \ldots, m\} : i \notin U\}$)
- Not include items that have already been used on the path.

Formally, for any node, θ , of the search tree,

$$\text{children}(\theta) = \{B \in \{B_1, \ldots, B_n\} : i^* \in B.S, B.S \cap U = \emptyset\} \quad (17)$$

This search can be seen in Figure 3 below, which is taken from [24]

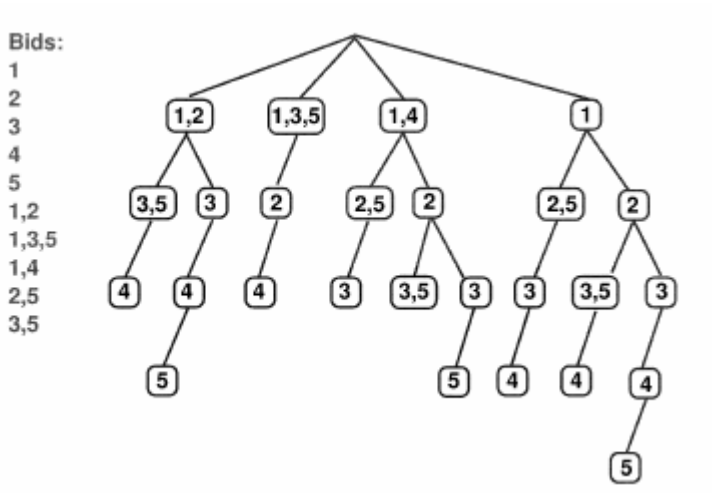


Figure 3 SEARCH1. This example search space corresponds to the bids listed on the left. In the figure for each bid, the items are shown but the price is not (From [24])

At any given node, θ, of the tree, children(θ) has to be determined. An easy approach is to loop through a list of all bids at every search node, and accept a bid as a child if it includes i*, and does not include items in U. This takes O(nm) time per search node because it loops through the list of bids, and for each bid it loops through the list of items.

Sandholm propose a more sophisticated scheme to make child generation faster. He uses a secondary depth-first search, SEARCH2, to quickly determine the children of a node. SEARCH2 takes place in a different space: a data structure which he calls the *Bidtree*. Bidtree is a binary tree in which the bids are inserted up front as the leaves (only those parts of the tree are generated for which bids are received) (*Bidtree* data structure can be seen in Figure 4 which is taken from [24].

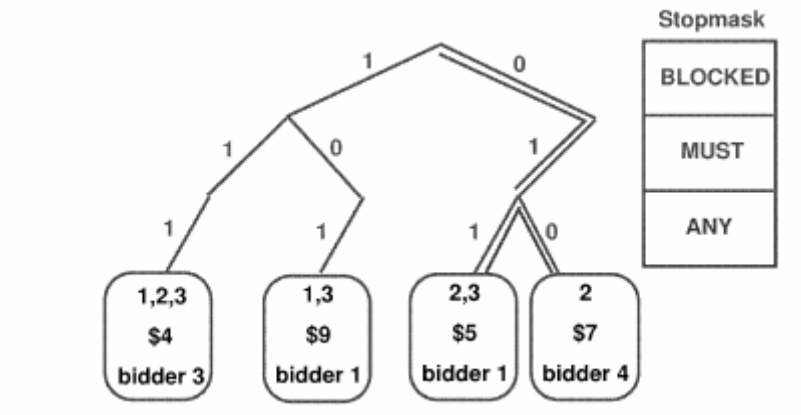


Figure 4 Bidtree data structure and stopmask (from [24])

Bidtree uses a *Stopmask* which differentiates the Bidtree from a classic binary tree. The Stopmask is a vector with one variable for each item, $i \in M$. Stopmask[i] can take on any one of three values: BLOCKED, MUST, or ANY. If Stopmask[i] = BLOCKED, SEARCH2 will never progress left at depth i (the root of the tree is at depth 1). This has the effect that those bids that include item i are pruned instantly and in place. If, instead, Stopmask[i] = MUST, then SEARCH2 cannot progress right at depth i . This has the effect that all other bids except those that include item i are pruned instantly and in place. Stopmask[i] = ANY corresponds to no pruning based on item i : SEARCH2 may

go left or right at depth i . The basic principle is that at any given node of the main search, Stopmask[i] = BLOCKED for all $i \in U$, and Stopmask [i*] =MUST, and Stopmask[i] = ANY for all other values of i .

The algorithm is implemented as depth-first search, which executes in linear space. The depth-first strategy causes feasible allocations to be found quickly as Sandholm state, and the solution improves monotonically since the algorithm keeps track of the best solution found so far. This implement the anytime feature: if the algorithm does not complete in the desired amount of time, it can be terminated before the proper time, and it guarantees a feasible solution that improves monotonically in time.

As Sandhom notes, the worst case complexity is $O(n^m)$. Where n is the number of bids actually received, not the number of allowable combination of items. This method uses an iterative deepening A* search (IDA*). It is a search strategy with a heuristic that they developed specifically for the winner determination application, which is guaranteed to find an optimal solution if heuristic is admissible. It does not comprise optimality. Furthermore, before the main search a preprocessing search is run which removes all the bids that are provably noncompetitive. Sakurai et al. [22] introduce the idea of *limited discrepancy search* (LDS) instead of IDA* algorithm in the Sandholm's method [24]. The benefit of LDS is that it can avoid time-consuming re-computation of heuristic function $h(\cdot)$, since LDS is less sensitive to the quality of $h(\cdot)$.

Sakurai et al. introduce LDS techniques to limit the search efforts to the regions where good solutions are likely to exist. LDS is shown to be very effective for many application problems. Compared to IDA*, the search performance of LDS is less sensitive to the quality of the heuristic function. Therefore, LDS can find good solutions without a precise heuristic function. Experimental evaluations using various problem settings shows that the allocations are very close to the optimal solutions, and this method can be extended to very-large-scale problem instances. Sakurai et al. point out more specifically that, LDS can achieve better than 95% of the optimal solution in about 1% of the running time compared with IDA*.

In their algorithm they use also some part from Fujishima et al. [22]. Fujishima et al. present an algorithm called the Combinatorial Auction Structured Search (CASS) algorithm, which is based on the depth-first branch-&-bound algorithm. In their method, Sakurai et al. use data structure called bins (bins are special data structures used in the CASS algorithm) which is different from Sandholm's method (as stated above he uses a binary tree called Bidtree). Sakurai et al. use bins to determine the children of each node in a search tree

Sandholm et al. [26] introduce a more sophisticated algorithm for optimal winner determination. The enhancements include structural improvements that reduce search tree size, faster data structures, and optimizations at search nodes based on driving toward, identifying and solving tractable special cases. They also generalize combinatorial auctions to auctions with multiple units of each item, to auctions with reserve prices on singletons as well as combinations, and to combinatorial exchanges. They also give algorithms for determining the winners in these generalizations.

The skeleton of the algorithm is a depth-first branch-and-bound tree search that branches on bids. Sandholm et al. note that both of the previous search algorithms for winner determination Fujima et al. [26] and Sandholm previous algorithm [24] branch on items [26]. In the branch-on-items formulation, as a preprocessing step, a dummy bid of price zero is submitted on every individual item that received no bids alone (to represent the fact that the auctioneer can keep items) [24] but in branch-on-bids (BOB) there is no need to place dummy bids. The differences between branch-on-items and branch-on-bids can be seen in Figure 5 which is from [26].

In Figure 5 we see that in arrow means choosing the bid and out arrow means not choosing the bid. For example, for the bids {1,2}, {2,3}, {3}, {1,3} if we choose bid {1,2} (i.e. branch in {1,2}) , we can also choose bid {3} or not, but not the other bids {2,3} and {1,3} (of course this figure is for multi-item single-unit, if this figure was for multi-item multi-unit, then the remaining bids that can be branched in would be decided

according to remaining resources. We take into account this factor in our search model as can be seen in section 3.2.1.1)

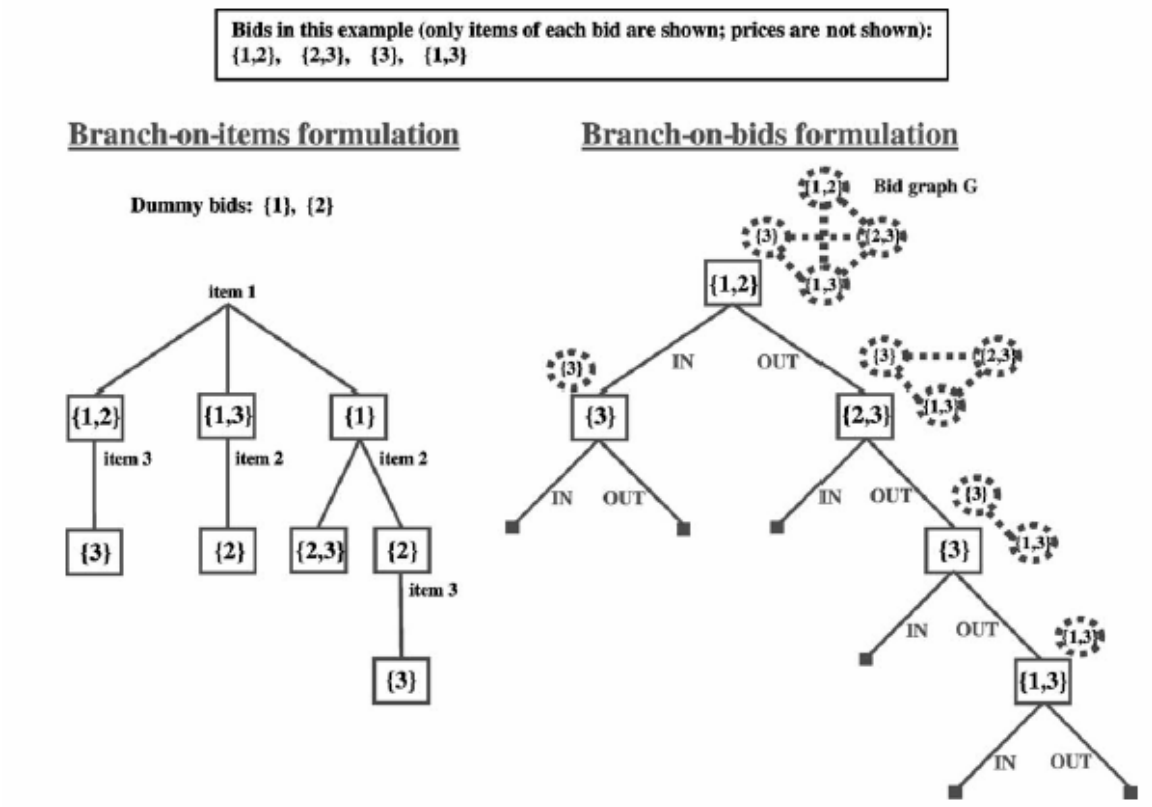


Figure 5 Branch-on-items vs. Branch-on-bids (from [26])

The main advantage of BOB compared to the branch-on-items formulation is that BOB is in line with the AI principle of least commitment [21]. In a branch-on-items tree, all bids containing an item are committed at a node, while in BOB, choosing a bid to branch on does not constrain future bid selections. BOB allows more refined search control (better bid ordering).

Another advantage of the BOB algorithm can be seen when we consider multi-unit setting. Previous winner determination algorithms cannot be used in the multi-unit setting because they branch on items (Fujima et al. [26] and Sandholm previous algorithm [24]). Even if each unit is treated as a separate item, the earlier algorithms cannot be used if the demands are real-valued instead of integer [26].

In this thesis, BOB algorithm is modeled because of it is advantageous to be used in multi-item multi-unit settings (one of the most complex setting in Combinatorial Auctions). More details of the BOB algorithm and the implemented model can be found in chapter 3.

One of the models that is built is inspired from BOB algorithm. The second one is a parallel auction mechanism. The last one is an IP (Integer Programming) formulation which is similar to the formulation that is given in the next section. A simple web server is also constructed which has the ability to the combinatorial auction and can be used for online auctions and e-procurement systems. The details can be found in chapter 3.

2.7 Multi-item Multi-unit Setting Formulation

In some auctions, there are multiple indistinguishable units of each item for sale. The bids can be compressed and speed up winner determination by not treating every unit as a separate item, since the bidders do not care which units of each item they get [26]. The bids are formulated as $B_j = \{(\lambda_j^1, \lambda_j^2, ..., \lambda_j^m), p_j)$, where $\lambda_j^k \geq 0$ is the requested number of units of item k, and p_j is the price.

The winner determination problem formulated by Sandholm et al. [26] is:

$$\max \sum_{j=1}^{n} p_j x_j$$

$$\text{s.t.} \sum_{j=1}^{n} \lambda_j^i x_j \leq u_i, \quad i = 1, 2, ..., m \tag{18}$$

$$x_j \in \{0,1\}$$

where u_i is the number of units of item i for sale and $x_j = 1$ if bid j wins and $x_j = 0$, if bid j loses.

CHAPTER 3

Implementation

When we survey some of the previous works that are done on Combinatorial Auctions Problem[3], we see that the previous works can be categorized into three parts. One part tries to solve the unrestricted problem using search algorithms [23, 24, 26], second part tries to solve the problem using OR (Operation Research) methodologies and/or deals with the identification of tractable cases of the combinatorial auctions problem [31, 32] and the last part deals with the usage of sequential or simultaneous auctions in order to solve the combinatorial auctions problem [3, 4, 19].

As can be seen in chapter 2, many of the previous studies were for single-unit Combinatorial Auctions Problem and they cannot be used in multi-item multi-unit settings. Also many of the previous studies compare different approaches to solve the CAP, but they generally compare them by not considering multi-unit models excepts in [28]. As a result, in this study, we decided to focus on multi-unit combinatorial auctions. We decided to compare different approaches in order to solve multi-unit combinatorial auctions problem, based on the settings given in chapter 4. We built three different models from three different approaches. We compare their performances using computer simulations where we model Bandwidth allocations system. And we decided to build a simple combinatorial auction tool which can be used for online

[3] More details of the previous works can be seen in chapter 2 – Review of Literature

combinatorial auctions and can be used as a test bed for combinatorial auctions. In this section the models that are built will be explained.

3.1 Introduction

The first model we build, tries to solve the CAP using search. The search algorithm in this model is very similar to Sandholm's sophisticated search algorithm but some modifications are done [24, 26, 27]. This model finds the optimal solution when it is given enough time. The second model is built by considering the simultaneous auctions mechanisms used in order to solve the CAP. We were inspired from Courcoubetis et al. for this model [4]. The third model is an IP formulation like the formulation given in section 2.7 and uses the GNU Linear Programming Kit (GLPK)'s callable library in order to find the solutions.

The performance of each model is compared to each other and analyzed using computer simulations. We model bandwidth allocation among multiple users, since as stated before bandwidth allocation is one of the well known examples where preferences of bundles situation occurs. It can be modeled as multi-item multi-unit auctions which is one of the most complex situations that occur in CAP. Based on the performance of the models and some other situations, we chose one of the models and built a simple Combinatorial Auction tool (CATool) which has the ability to the combinatorial auction and can be used for online auctions, and e-procurement systems and can be used as a test bed for combinatorial auctions.

All of the three models and the tool are developed by using C++ programming language. The three models are developed by using Microsoft Visual Studio in ANSI C++ but the tool is developed by using Borland C++ Builder and using its VCL (Visual Component Library). Object-oriented analysis and design methodologies are utilized throughout the development. The presented diagrams are created by using Rational Rose tool. The code generation directly from the class diagrams functionality of the

Rational Rose tool is also utilized. By using this functionality the integrity between the class diagrams and codes is achieved. The component diagram can be seen in Figure 6.

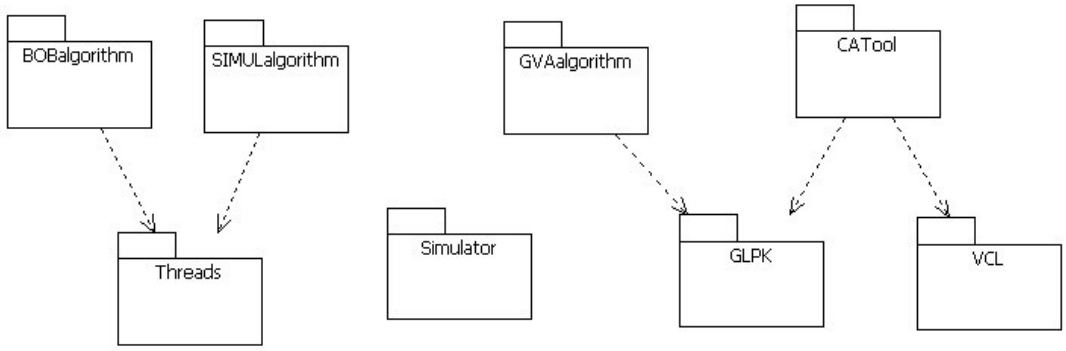


Figure 6 Component Diagram

The Threads package implements the thread and mutex classes. These classes are used by BOBalgorithm package and SIMULalgorithm package classes, since these models are threaded applications. The relations between the classes in this package are depicted as a class diagram, which is shown in Appendix-A Figure A8. The LThread class is used to implement a thread class. Some of the BOBalgorithm and SIMULalgorithm package classes are derived from this class since they are threaded applications. LThread is built in a similar logic of the Borland C++ Builder's TThread abstract class. The Terminate method terminates the execution of the LThread. The Terminated method checks whether the LThread is terminated or not. The Execute method is responsible for checking the value of the Terminated property to determine if the thread needs to exit. A LThread calls the Execute method when the LThread is created and the CreateSuspended method is set to false, or when Resume is first called for the LThreads that are created suspended by setting the CreateSuspended to true. And until the LThread is terminated, the Suspend method is called or the Execute method finishes the LThread executes the Execute method.

By the LMutex class the synchronization between threads are achieved. This is done by Critical Sections. When a LMutex is created, it initializes critical section and deletes

this critical section when the LMutex is deleted. The Lock method enters the critical section and the Unlock method leaves the critical section. These are made by windows functions.

The BOBalgorithm package implements the classes that implement the first model. The relationships between the classes in this package are depicted as a class diagram, which can be seen in Appendix-A Figure A2 and more detail is given in section 3.2.1.1 .

The SIMULalgorithm package implements the classes that implement the second model. The relationships between the classes in this package are depicted as a class diagram, which can be seen in Appendix-A A4 and more detail is given in section 3.2.1.2.

The GVAalgorithm package implements the classes that implement the third model. This package uses functionalities from the GLPK package. As stated before GLPK is a free Linear Programming Toolkit, this toolkit has callable libraries that can be called from C++ code. In this work, we used this callable library in order to solve the IP model. The relationships between the classes in this package are depicted as a class diagram, which can be seen in Appendix-A A6 and more details about GLPK and the IP model is given in the section 3.2.1.3

The GLPK package is a set of routines written in ANSI C and organized in the form of a callable library. This package is intended for solving large-scale linear programming (LP), mixed integer linear programming (MIP), and other related problems. This is a free program. GLPK version 4-4 is used in this work. More information about GLPK can be found at [10]

The Simulator package is a free ANSI C library for multi-stream random number generation. More details can be found in [20]. In this package, we only collect the ANSI C functions into a C++ class (Simulator). The class diagram for Simulator package can be seen in Appendix-A Figure A9.

The VCL package is the Borland C++ Builder's Visual Component Library (VCL).

The CATool package implements the classes of the simple combinatorial auction tool which has the ability to do the combinatorial auction and can be used for online auctions. More details will be given in the 3.2.2.

3.2 Detailed Descriptions

We can divide the work done into two main parts. In the first part the three models are built and the performance of each model is compared and the models are analyzed using a computer simulation[4]. In the second part based on the performance of the models and some other situations, a simple Combinatorial Auction Tool (CATool) is built. In this section brief overviews of the deployed classes of the models and the tool will be presented.

3.2.1 Information about Models

In Figure 7 Main class diagram can be seen. MainClass is the class that starts the process. MainClass creates input files by using CreateInputFile method by using the Simulator class functionalities[5]. By the pointers that MainClass includes the same input file is evaluated by all of the three models, and the results of the evaluations and the time passed to find the solutions for each model are calculated by MainClass. The results can be seen in Chapter 4. Since the sequence diagram of this process from start to end is too long and complex, we have split this sequence diagram into three in order to explain the initialization process of the three models better.

---

[4] The details of the simulation results can be found in Chapter 4

[5] More details about the criteria used for creating the input files can be found in Chapter 4

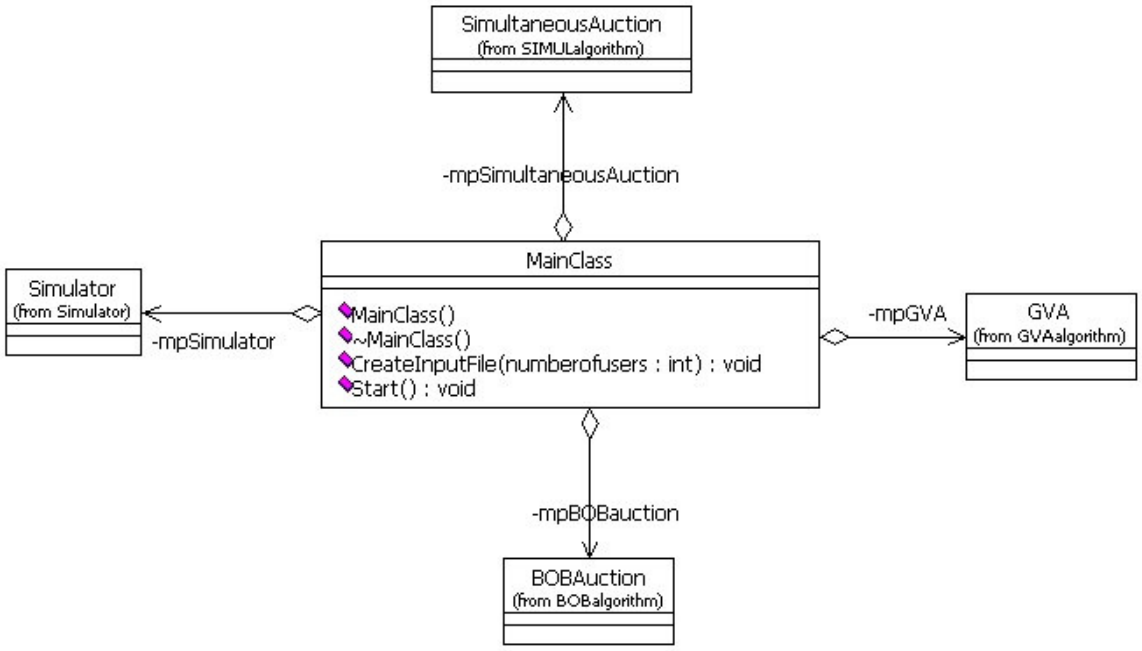


Figure 7 Main Class Diagram

3.2.1.1 The Search Model (SM)

To build this model, we were inspired by [24, 26, 27]. The high-level idea of branching on bids was proposed by Sandholm et al. in the BOB algorithm [26]. However BOB algorithm was not implemented. Sandholm et al. implements an algorithm called CABOB (Combinatorial Auction Branch on Bids), which incorporates many of the techniques proposed in BOB algorithm and some additional ones [27].

In our search model, we implement many features from the BOB algorithm, also get some idea from CABOB algorithm and also use some features from the [24] like BidTree. The entire search is done in the Graph class as seen in Appendix-A Figure A2. Our algorithm finds optimal solution given enough time and our algorithm can be used for multi-item multi-unit settings as [26] and [27].

The Algorithm can be summarized as below. The skeleton of our algorithm is a depth-first tree search that branches on bids. See Figure 5 for Branch on Bids formulation. Below the sequence of operations can be seen.

Algorithm 1

1. Create the adjacency list, from the given bids and items, in order to represent the bid graph[6] (look at the BidTree class for details)
2. Create the first graph whose component no[5] is 0. (a new instance of Graph class)

Until the created graph terminates do

   a. Run depth-first-search (DFS)[7] in order to find the number of connected components, say n.
   b. If n shows that no nodes to branch on[8]. Then update the bid graph nodes, optimum path cost and its corresponding path. If the optimum path is found, then terminate else continue.
   c. If n is 1 (i.e. the bid graph is connected), find an articulation point[9] to branch on[10]. And then update the current optimum path and current path cost.
   d. If n >1,
      i. Create a new instance of Graph class $Graph_i$ for each component[11] i.

---

[6] see the BidTree class for details

[7] By the DFS method of the Graph class.

[8] All of the bid graph nodes with component no same as the Graph object's component no are deleted, see BidTree class and Graph class.

[9] By the FindArticulation method of Graph class

[10] By the MarkInDeleted method of Graph class

[11] Different component numbers are decided and updated in the DFS method as seen in the DFS method's explanation in Graph class.

ii. Calculate an upper bound[12] for the optimum solution that can be found in $Graph_i$. If the total heuristic upper bound values and the current path cost is more than the optimum path cost, then each the $Graph_i$ starts its process (i.e. goes to step a). Then wait until each $Graph_i$ is terminated.

iii. Update current path cost. Then delete every child $Graph_i$[13] . Then update the bid graph nodes, optimum path cost and its corresponding path. If the optimum path is found, then terminate else continue.

e. Go to step a if the graph is not terminated (i.e. time out occurs) or the optimum solution has not been found.

BOBAuction Class

As seen from the sequence diagram Appendix-A Figure A1 the search model starts when the MainClass creates the BOBAuction class. Then MainClass calls the StartBOBAuction method to start the Search Model's process. When this method is called, BOBAuction class creates the BidTree class in order to create the Adjacency List (for the input file that is created by the MainClass). Then BOBAuction class creates the Graph class and then calls the Execute method of the Graph in order to start the Search algorithm to find the solution for the given input. As seen from Appendix-A Figure A1 then MainClass recursively check if a solution is found by calling the IsBOBAuctionFinished method of the BOBAuction class, if in a given time limit the solution can not be found then the search process is terminated by the main class and the current optimal solution is get (not necessarily the optimal solution).

---

[12] By the CalculateHeuristic method of the Graph class

[13] The BacktoComponentno method of the Graph class is called for every $Graph_i$ in order to update the component numbers

BidTree Class

The BidTree class creates the adjacency list (AdjList). The adjacency list is used to represent the bid graph (The vertices of the bid graph are the bids and two vertices share an edge if the bids share items). The bid graph idea was given in [26] but our representation is a bit different from this representation. In [26] they use an array to store the nodes of bid graph G. The array entry for a node j points to a doubly-linked list of bids that share items with the node j. But in our implementation, the adjacency list is a vector (from C++ Standard Template Library - STL). It is a vector of Vertex pointer type. As seen in Appendix-A Figure A2 Vertex is a struct defined in order to store the bids information. Also vertex structure includes componentno information which is used to determine connected components in the bid graph after DFS (depth first search) which is done by the Graph class. In our algorithm there is one adjacency list i.e. bid graph that is created by BidTree class once but as seen in [27], in the CABOB algorithm there is a bid graph for every connected component. Below in Table 1 an example of adjacency list for bids {1, 2}, {1, 3}, {2, 3}, {3} can be seen. BidTree class uses the BidTree data structure which is explained in [24]. The BidTree data structure is used for every node j in the adjacency list to find the neighbor bids (bids that share items with node j). The BidTree data structure is not used in [26] and in the CABOB algorithm [27].

Table 1 Adjacency List structure

{1,2}	→	{1,3}	{2,3}	
{1,3}	→	{1,2}	{2,3}	{3}
{2,3}	→	{1,2}	{1,3}	{3}
{3}	→	{1,3}	{2,3}	

Graph Class

The Graph class is the class where the entire search is done. The Graph class is derived from LThread class, i.e. the graph class is a thread. The graph class is designed as a thread because decomposition leads to time saving since search time is super linear in the size of G [27]. A Graph instance executes until the optimum solution is found or it is terminated (i.e. time out occurs). While the Graph class is created the component no

information is also given in the constructor. Every graph class is only interested on the adjacency list members that have the same component number as the Graph class. So at a time every graph class's component number should be different than each other. This is done by the Component class. Below some of the important methods are explained in order to have a better understanding of the Algorithm 1.

DFS method finds the number of connected components. In [26] and [27] it is advised that, if the set of items can be divided into subsets such that no bid includes items from more than one subset, the winner determination should be done in each subset separately. Since the search is super linear in the size of the problem, such decomposition leads to a speedup. By using depth-first-search on the bid graph G, DFS method finds the number of connected components, updates the component number of each node by a different component number (every connected component has the same component number). If the entire bid graph nodes are deleted this method also finds that. If there are more that one connected component, for each connected component a Graph class is created.

BacktoComponentno method updates the component numbers of the nodes back to the original ones.

FindArticulation method finds articulation points. In addition to checking whether decomposition has occurred, we pick a bid that leads to decomposition, if FindArticulation finds an articulation bid. The algorithm to find an articulation point can be found in [33]. In Figure 8 articulation points can be understood more easily, here point D is an articulation point. This figure is from [26].

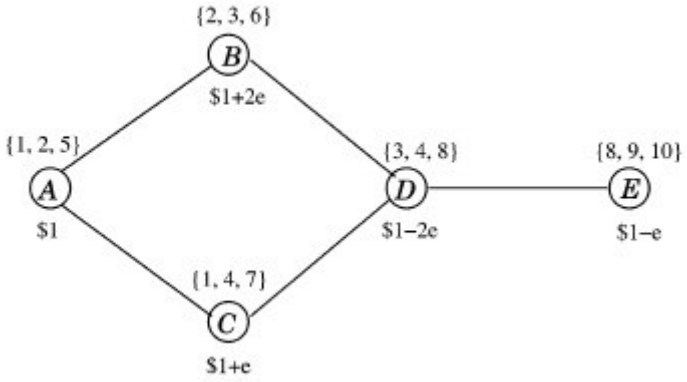


Figure 8 Example for articulation points (from [26])

MarkInDeleted method is called when to branch on that bid (the "in" arrow in Figure 5). It sets the isdeleted attribute of the bid node to true. It updates the resources of the items (for multi-item multi-unit setting), which the "branched in" bid wants. According to the remaining resources it deletes some of its neighbors, then.

CalculateHeuristic method calculates an upper bound for the optimum solution that can be found. Admissible heuristic setting is the same as [26]

$$h = \sum_{i \in M} [(u_i - \Lambda_i) \max_{j \in V_G | \lambda^i_j > 0} \frac{p_j}{\sum_{i \in S_J} \lambda^i_j}] \tag{19}$$

$u_i - \Lambda_i$ is the remaining units of item i. As showed earlier bids are formulated in our setting as $B_j = \{(\lambda_j^1, \lambda_j^2, ..., \lambda_j^m), p_j\}$, where $\lambda_j^k \geq 0$ is the requested number of units of item k, and p_j is the price.

In CABOB algorithm [27] the upper bound is found by solving an LP by LP Solver. It has been demonstrated that the upper bound from LP is significantly tighter than those proposed by other combinatorial auction winner determination algorithms, by their experiments [27]. Also the time taken to solve the LP at every node is negligible compared (I think this depends on the LP solver) to the savings in the search due to

enhanced pruning. In the algorithm CABOB, they control a special case called INTEGER. If the LP returns integer values ($x_j = 0$ or $x_j = 1$) for each bid j, they use this solution and since it is an optimal solution. Here the important point to note is that, they state that this occur more often than they expected. It has been thought that the speed of the CABOB algorithm mainly depends on the LP solver because of INTEGER special case and finding upper bound by solving an LP. Sandholm et al. benchmark CABOB against a general purpose integer programming package, CPLEX 7.0 (the results can be found in [27].

3.2.1.2 The Descending Simultaneous Auctions Model (DSAM)

In this model we built descending simultaneous auction mechanism. To build this model, we were inspired from [4] and [3]. Courcoubetis et al. [4] developed an auction mechanism for bandwidth allocation in a network. Their mechanism consists of a set of simultaneous multi-unit Dutch (i.e. descending price) auctions (MIDAS), one per link of the network. Courcoubetis et al. [4] state that they have proved that when employing ascending auctions, it is impossible to synchronize the auctions of the various links so that all of them terminate at the same time. Also more information about Dutch auctions can be found in [3]. Although high level idea was given in [4], we implement our own rules and also the details of the model is decided and designed by us. By this model we implement a descending simultaneous auctions model in order to solve the combinatorial auctions problem. As explained before descending auctions can occur in multi rounds (see section 2.3). In this model also auction occurs in multi round. As explained before this model will not necessarily find the optimum solution (see section 2.3.5.6).

This model is implemented by the SIMULalgoithm package. In Appendix-A A3 sequence diagram of the initialization process by the MainClass can be seen. As seen from the class diagram for the SIMULalgorithm package in Appendix-A A4, this model is also a threaded application. This model can be used for multi-item multi-unit

auctions. Each item is represented by a Seller class instance, and for each bidder there is a corresponding Bidder class instance. There are also a single Auctioneer class instance and a single Ortak class instance to help synchronization between sellers and bidders. Here Ortak class is written by singleton pattern technique (design pattern). State of the process can be one of the five possible states, which are: NewRound state, StartBidding state, StartAllocation state, Result state and NotStarted state. Sellers are only interested in the NewRound state, and StartAllocation state, whereas Bidders are only interested in StartBidding state, and Result state. Auctioneers are interested in all of the states. SimultaneousAuction class instance makes the initializations of the model and the starts the model process.

The sequence of operations in every round, for the model can be summarized as below.

Algorithm 2

Every new round starts with a NewRound State. The state of the process is changed by the Auctioneer class.

1) In the NewRound state all Sellers looks at their remaining units and to the entire Bidders' list in order to see if there are some Bidders still not terminated. If there are still unallocated units of the item and there is at least one Bidder active, Seller determines the new price of the item[14] (in every round the items unit price is decreased). Else terminates the seller. And lastly increase the seller count in order to determine the number of Sellers that has processed in the NewRound state.
2) While the Seller threads process as in the 1st step Auctioneer thread controls whether all of the Sellers has processed the NewRound state or not, in parallel. When every Seller has processed the NewRound state, the Auctioneer finds the current active seller count. If it shows that all of the Sellers have terminated,

[14] By calling CalculateSellingPrice method that is explained below

then terminate the Auctioneer instance (i.e. the algorithm finishes and the solution is found), else change the state to StartBidding state.

3) When the state is the StartBidding state, the Bidders process. Every Bidder first controls if any of the Seller which sells one of the item the Bidder wants has terminated, or has enough remaining units, and if the Bidders valuation is higher than, all of the corresponding Sellers' selling price. For example a Bidder $Bidder_j$ wants 2 units from item a, and 3 units from item b and the total valuation of the bidder is 10. If $Seller_a$ or $Seller_b$ has terminated i.e. item a has finished or item b has finished than $Bidder_j$ has no chance to buy his bid and should terminate. If $Seller_a$ and $Seller_b$ has not terminated but the remaining units from a or b is smaller than the requested unit by $Bidder_j$, than again $Bidder_j$ should terminate since it has no chance to buy his bid, since as explained before $Bidder_j$'s valuation is for the specified units, i.e. not for below or more. Lastly if $Seller_a$ and $Seller_b$ are both active and have enough units from item a and b, but total value of $Seller_a$'s unit selling price times 2 and $Seller_b$'s selling price times 3 is bigger than $Bidder_j$'s total valuation, than $Bidder_j$ again does not bid at this round and waits for the next round. We accept this rule since we want to minimize the chance of misallocations. For example think that $Bidder_j$ gets one of its items and waits for the next rounds for the unit selling price to decrease to buy the other items, and say that at the end of the auction it can't buy the other items, than in this condition an unwanted allocation will occur. In order to prevent this kind of unwanted allocations a Bidder bids for items only if the Bidder can bid for all of the items simultaneously in the same round. And lastly increase the bidder count in order to determine the number of Bidders that has processed in the StartBidding state.

4) While the Bidder threads process as in the 3rd step, Auctioneer thread controls whether all of the Bidders has processed the StartBidding state or not, in parallel. When every Bidder has processed the StartBidding state, the Auctioneer finds the current active bidder count. If it shows that all of the Bidders have terminated, then terminate the Auctioneer instance (i.e. the

algorithm finishes and the solution is found), else change the state to StartAllocation state.

5) In the StartAllocation state, allocations are done by Sellers based on the bids made by the Bidders. Each Seller first sorts its corresponding Bidders that bid for the item that the Seller sells at the current round based on Bidders' unit valuation. Since the unit selling price is fixed for the item (different for each item), the difference between Bidders is decided by their unit valuation. Then each Seller considers its corresponding Bidders one by one from the highest to the lowest. If the Seller has more remaining units than the Bidder's request and the Bidder is still active, than Seller allocates the Bidder's requested unit to it updates its resource and evaluates the other Bidders in the sequence, else (the Bidder wants more units than the remaining units) make the Bidder inactive. And lastly increase the seller count in order to determine the number of Sellers that has processed in the StartAllocation state.
6) When all of the allocations done, the Auctioneer instance change the state to Result state.
7) In the Result state the Bidders controls the items that they have been allocated in the StartAllocations state by the corresponding Sellers. As seen in step 5 if an item is allocated to a Bidder by a corresponding Seller, it is allocated as the requested units of the Bidder not below or more, or the item is not allocated to that Bidder at all (and the Bidder is made as inactive). Each Bidder first controls its activity. If it is active which means that the Bidder has got all of the items it wants, than the Bidder terminates by increasing the revenue of the process. If it is inactive (which means that it could not get some items that it wants in the StartAllocation state), then it gives all of the items that it has got in order to prevent unwanted allocations, and the Bidder again becomes active and waits for the next rounds. Of course this can also cause inefficient allocations but in order to prevent unwanted allocations, we accept this inefficiency. And lastly increase the bidder count in order to determine the number of Bidders that has processed in the Result state.

8) When every Bidder has processed the Result state, the Auctioneer finds the current active bidder count. If it shows that all of the Bidders have terminated, then terminate the Auctioneer instance (i.e. the algorithm finishes and the solution is found), else change the state to NewRound state i.e. a new round starts and go to step 1.

At every round the unit selling prices for every item are determined by the corresponding Seller by calling the CalculateSellingPrice method. In this method, slightly modified Variable Reduction Rates (VRR) policy is selected to reduce prices (for more information about price reduction policies you can look at [4]. This policy involves a decrement rate and at each round r the unit selling price of item i is determined by the following equation where $p_i(r)$ is the selling price of item i at round r

$$p_i(r) = \max\{p_i(r-1) - \max\{[C_{remaining}(r;i) / C_{init}(r)] * MaxDrop, 1\}, \text{reserve price}\} \quad (20)$$

That is the decrement rate of item i at a round r is proportional to the fraction of the current remaining units $C_{remaining}(r;i)$ divided by its initial value $C_{init}(r)$. Thus the prices reflect the demand. By this policy less demanded items' unit prices will drop faster and more demanded items' unit prices will drop slower. The price at each link is reduced at every round at least by 1 and at most by MaxDrop.

SimultaneousAuction Class

As seen from the sequence diagram in Appendix-A A3 the Descending Simultaneous Auctions Model starts when the MainClass creates the SimultaneousAuction class. Then MainClass calls the StartSimultaneousAuction method to start the Descending Simultaneous Auctions Model's process. When this method is called, SimultaneousAuction class calls the SetStatus method of the Ortak class in order to set the status of the process as NotStarted. Then for each item, SimultaneousAuction class creates a Seller class and inserts it to the seller list of the Ortak class. Then the Resume method of the Seller is called. Then for each bid a Bidder class is created. For each item requested in the bid, Bidders AddItem method is called and the item is added to the

bidder's list and the bidder is added to the corresponding Seller's list . The Bidder is also added to the Bidder list of the Ortak class by calling the AddBidder method. The Bidder's resume method is called. Lastly an Auctioneer class is created and the status is set to NewRound status. Then MainClass recursively check if a solution is found by calling the IsSimultaneousAuctionFinished method of the SimultaneousAuction class, if in a given time limit the solution can not be found then the process is terminated by the MainClass. As seen in Appendix-A A3 the Release method is called in order to delete all Sellers and all Bidders.

Seller Class

The Seller class is derived from LThread class, i.e. the Seller class is a thread. The Seller class is designed as a thread because it needs to do parallel operations with Bidder class and Auctioneer class as in the real marketplace. Sellers are only interested in the NewRound state and StartAllocation state.Each item is represented by a Seller class instance.

Bidder class

The Bidder class is derived from LThread class, i.e. the Bidder class is also a thread. The Bidder class is designed as a thread because it needs to do parallel operations with Seller class and Auctioneer class as in the real marketplace. Bidders are only interested in StartBidding state, and Result state. For each bidder there is a corresponding Bidder class instance.

Auctioneer Class

The Auctioneer class is derived from LThread class, i.e. the Auctioneer class is also a thread. The Auctioneer class is designed as a thread because it needs to do parallel operations with Bidder class and Seller class as in the real marketplace. Auctioneer is interested in all of the states and controls the process. There is a single Auctioneer Class.

Ortak Class

In this model the Ortak class is a singleton class. It has a private static pointer of type Ortak class. Instance method, which is also a static method, returns this reference. And as a result every instance has a chance to reach that single instance and this helps to synchronize between threads. There is a single Ortak Class.

3.2.1.3 IP Formulation Model (IPFM)

The IP Formulation in this model as the same as the formulation as explained in section 2.7. Although the name of the class is GVA, GVA algorithm is not implemented in this formulation because of the problems that occurs in the GVA algorithm as explained in section 2.6.1.2. To solve the IP model we choose the GLPK version 4.4.

GLPK (GNU Linear Programming Kit) is a set of routines written in the ANSI C programming language and organized in the form of a callable library. It is intended for solving linear programming (LP), mixed integer programming (MIP), and other related problems. GLPK is currently developed and maintained by Andrew Makhorin, Department for Applied Informatics, Moscow Aviation Institute, Moscow, Russia. GLPK is currently licensed under the GNU General Public License (GPL). GLPK is free software; it can be redistributed and/or modified under the terms of the GNU General Public License as published by the Free Software Foundation.

GLPK assumes the following formulation of linear programming (LP) problem:

Minimize (or maximize)

$$Z = c_1x_{m+1} + c_2x_{m+2} + \ldots + c_nx_{m+n} + c_0 \quad (21)$$

Subject to linear constraints

$$\begin{aligned} x_1 &= a_{11}x_{m+1} + a_{12}x_{m+2} + \ldots + a_{1n}x_{m+n} \\ x_2 &= a_{21}x_{m+1} + a_{22}x_{m+2} + \ldots + a_{2n}x_{m+n} \\ &\ldots\ldots\ldots\ldots \\ x_m &= a_{m1}x_{m+1} + a_{m2}x_{m+2} + \ldots + a_{mn}x_{m+n} \end{aligned} \quad (22)$$

And bounds of variables

$$l_1 \leq x_1 \leq u_1$$

$$l_2 \leq x_2 \leq u_2 \qquad (23)$$

…….

$$l_{m+n} \leq x_{m+n} \leq u_{m+n}$$

where: x_1 , x_2, ..., x_m - auxiliary variables; x_{m+1}, x_{m+2}, ... , x_{m+n} - structural variables; Z - objective function; c_1 , c_2 , ... , c_n - objective coefficients; c_0 - constant term ("shift") of the objective function; a_{11} , a_{12} , ... , a_{mn} - constraint coefficients; l_1 , l_2 , ... , l_{m+n} - lower bounds of variables; u_1 , u_2 , ... , u_{m+n} - upper bounds of variables.

Auxiliary variables are also called rows, because they correspond to rows of the constraint matrix (i.e. a matrix built of the constraint coefficients). Analogously, structural variables are also called columns, because they correspond to columns of the constraint matrix.

Bounds of variables can be finite as well as infinite. Besides, lower and upper bounds can be equal to each other. Thus, the following types of variables are possible:

Bounds of variable	Type of Variable
$-\infty < x_k < +\infty$	Free (unbounded) variable
$l_k \leq x_k < +\infty$	Variable with lower bound
$-\infty < x_k \leq u_k$	Variable with upper bound
$l_k \leq x_k \leq u_k$	Double-bounded variable
$l_k = x_k = u_k$	Fixed variable

To solve the LP problem (21) - (23) is to find such values of all structural and auxiliary variables, which:

a) Satisfy to all the linear constraints (22, and

b) Are within their bounds (23), and

c) Provide a smallest (in the case of minimization) or a largest (in the case of maximization) value of the objective function (21).

For solving LP problems GLPK uses a well known numerical procedure called the simplex method. The simplex method performs iterations, where on each iteration it transforms the original system of equality constraints (22) resolving them through different sets of variables to an equivalent system called the simplex table (or sometimes the simplex tableau).

MIP problem is LP problem in which some variables are additionally required to be integer. GLPK assumes that MIP problem has the same formulation as ordinary (pure) LP problem (21) – (23), i.e. includes auxiliary and structural variables, which may have lower and/or upper bounds. However, in case of MIP problem some variables may be required to be integer. This additional constraint means that a value of each integer variable must be only integer number. As seen IP (Integer programming) is subset of MIP and can also be solved by GLPK. The MIP solver currently is based on branch-and-bound, so it is unable to solve hard or very large problems with a probable practical limit of 100-200 integer variables. However, sometimes it is able to solve larger problems of up to 1000 integer variables. More information about GLPK and GLPK API routines can be found in [11]. In the GLPK_FAQ.txt file, which is in the GLPK distribution doc directory, it is stated that on very large-scale instances CPLEX 8.0 dual simplex is 10-100 times faster than the GLPK simplex solver and, much more robust. It is also stated that in many cases GLPK is faster and more robust than lp_solve 4.0 for pure LPs as well as MIP's. In the GLPK distribution doc directory, there is also a bench.txt file which gives netlib benchmark results for GLPK. Other benchmarks can be found for some of the well known LP and MIP solvers such as CPLEX, GLPK, lp_solve, and OSL can be found on Hans Mittelmann's webpage at [13]

In the GVA class the SolveMip method formulates the IP for the given input, which is created by the MainClass as seen from the sequence diagram Appendix-A A5. Then it solves the IP by calling the appropriate GLPK API routines. One thing to be careful while solving the MIP is, to first solve the LP relaxation of the MIP. If the simplex method can’t solve the LP successfully than the MIP solution can not be found. Also the optimum solution to the LP should be found otherwise the MIP can not be solved

by the branch-and-bound method. As seen from the sequence diagram Appendix-A A5, and the GVA class in Appendix-A A6, there is no mechanism to control the time limit. This is because by the help of the GLPK API, we can limit the computing time, by setting the control parameter LPX_K_TMLIM. Also by the GLPK API routine lpx_print_mip the solution can be printed into a specified file.

3.2.2 A Simple Combinatorial Auction Tool (CATool)

After examining the experiment results, it can be seen that IP formulation gives better results than the other two models considering evaluation times and revenue results (see chapter 4). Because of this reason a simple Combinatorial Auction Tool (CATool) is developed, which uses the IP formulation method to find winners. This tool is built by Borland C++ Builder and using its valuable VCL. This tool has the ability to do online combinatorial auctions. This tool has also the ability to be used as a simple web server. This is a simple client-server application. Clients (bidders) connect to the web site of the auction. Server gets the request from the clients, sends them corresponding pages. Server accepts wishes until the current auction ends, then evaluates the winners, updates the auction variables (including quantity, status, etc...) and sends the results to clients (bidders). The Server and client side structures can be seen in Figure 9 and Figure 10 respectively.

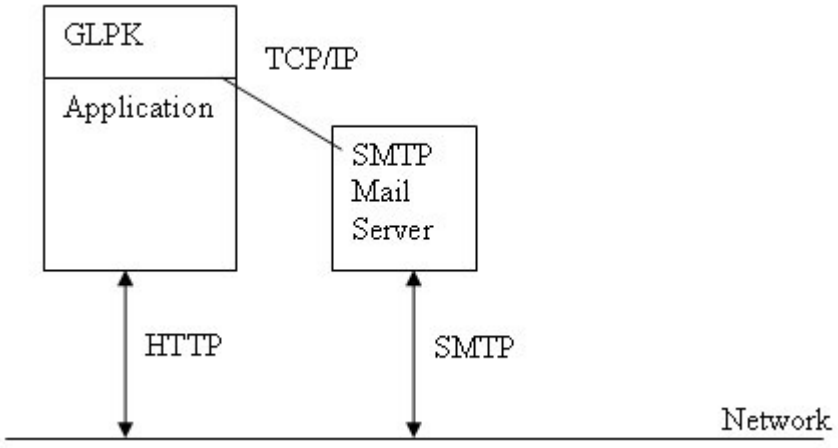


Figure 9 Server side structure

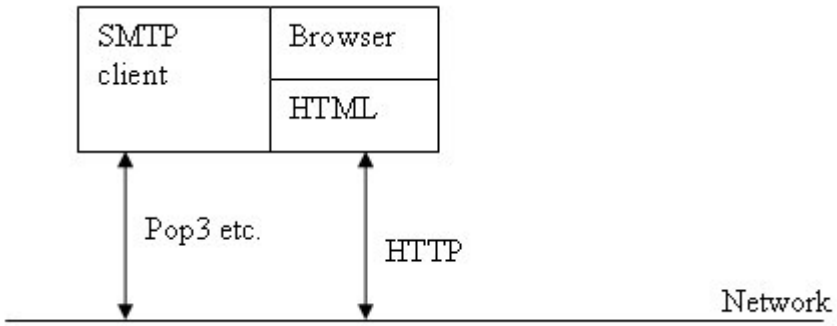


Figure 10 Client Side structure

3.2.2.1 Functional Requirements

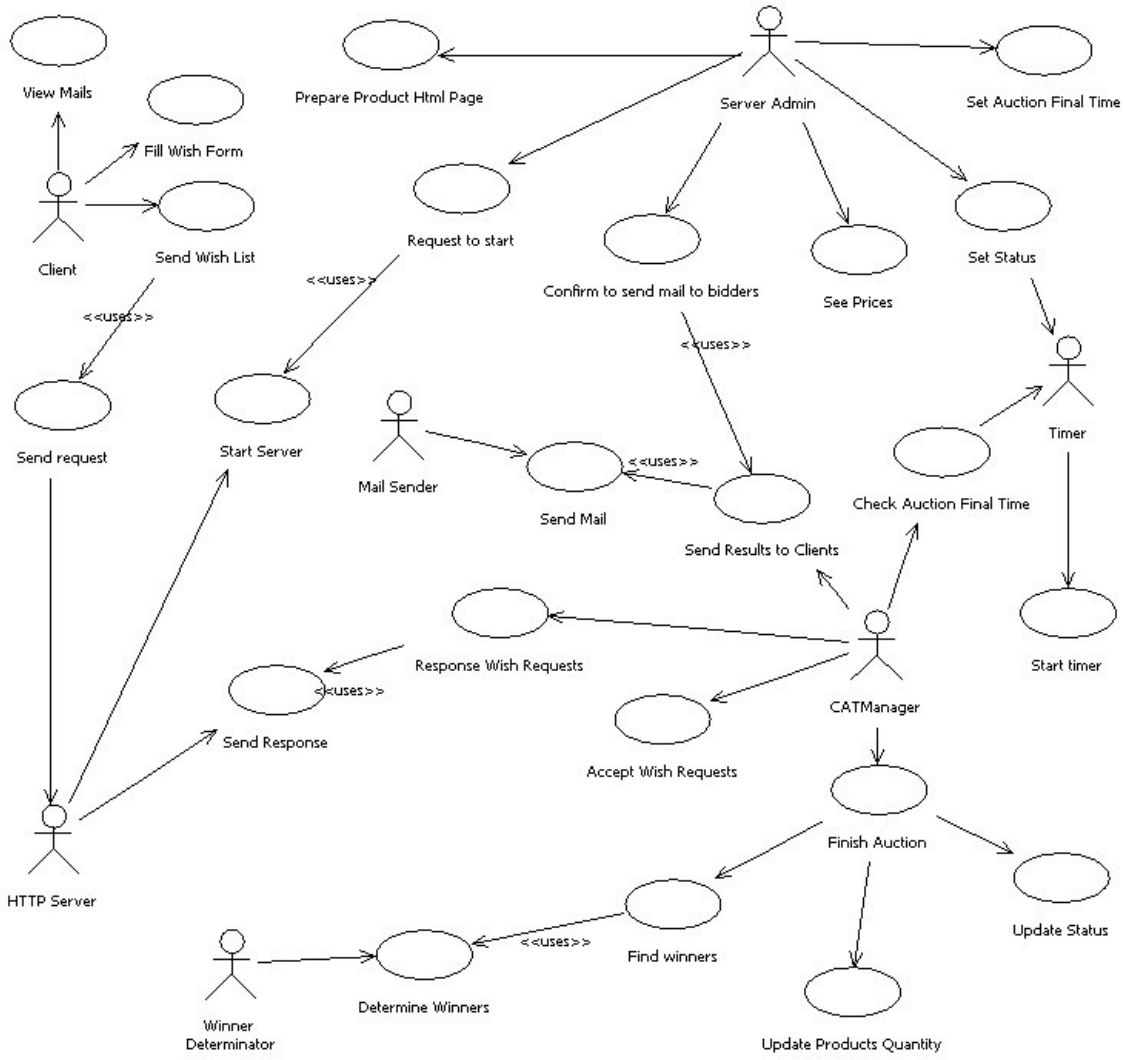


Figure 11 UC01 Use Case Model

The functional requirements of the CATool are depicted with the help of use case diagrams. Use cases can be seen in Figure 11. As seen the actors are Client, Winner Determinator, Server Admin, CATManager, HTTP Server, Mail Server, and Timer. The descriptions of the use cases are given in the following table, Table 2.

Table 2 Use Case Descriptions

UC #	UC Name	UC Description
UC01-1	View Mails	This use case is used by the client to see the mails received about the results of the auction
UC01-2	Fill Wish Form	This use case is used by the client to fill the wish list and other information like total valuation, name and email information
UC01-3	Send Wish List	This use case is used by the client in order to send his wishes to the server.
UC01-4	Send Request	By this use case the request of the clients are sent to the HTTP Server
UC01-5	Prepare product html page	This use case is used by the server admin to prepare the products html page. The server admin enters the products that will be sold in the auction, sets their quantities, names, and other related information. The corresponding html is created by the given name and the resources are updated
UC01-6	Set auction final time	This use case is used by the Server Admin. The server admin sets the time when the corresponding auction will end.
UC01-7	Set Status	This use case is used by the Server admin to set the status in order to start the auction, and enables timer
UC01-8	See prices	This use case is used by the Server Admin in order to see the prices given by the bidders throughout the auction. This gives a chance to analyze the auction.

Table 2 Use Case Descriptions Cont.

UC01-9	Request to start	This use case is used by the Server admin in order to start the HTTP Server. This use case uses the "Start server" use case in order the start the HTTP Server
UC01-10	Confirm to send mail to bidders	This use case is used by the server admin in order to confirm to send to results to the bidders. This use case uses "Send Results to Clients" use case in order to send the results
UC01-11	Start server	This use case is used by the HTTP Server in order to start the server.
UC01-12	Send Respond	This use case is used by the HTTP Server in order to send respond to clients
UC01-13	Start Timer	This use case is used by the Timer in order to start the timer.
UC01-14	Send Results to Clients	This use case is used by the CATManager in order to send results to bidders. This use case uses "send mail" use case.
UC01-15	Check Auction Final Time	This use case is used by the CATManager in order to check whether the auction final time has come by the help of Timer
UC01-16	Response wish requests	This use case is used by the CATManager in order to respond requests. It uses "Send Response" use case.
UC01-17	Accept wish requests	This use case is used by the CATManager in order to accept wish requests. Wishes are accepted if an auction is started until the auction final time arrives.
UC01-18	Finish Auction	This use case is used by the CATManager in order to evaluate the auction winners. It uses "Find winners", "Update Products

Table 2 Use Case Descriptions Cont.

		Quantity", and "Update Status" use cases in order to do this.
UC01-19	Find Winners	This use case is used by the CATManager in order to find the auction winners. It uses "Determine Winners" use case to do this.
UC01-20	Update Status	This use case is used by the CATManager in order to update the status.
UC01-21	Update Products Quantity	This use case is used by the CATManager in order to update the products quantity after the auction
UC01-22	Determine Winners	This use case is used by the Winner Determinator in order to determine the winners by the IP Formulation Model
UC01-23	Send Mail	This use case is used by the Mail Sender in order to send mails about results to the bidders.

3.2.2.2 Design of the CATool

After the functional requirements analysis the design of the CATool is made. As seen in the class diagram, Appendix-A A7, there are four classes which are the CATMainForm, ProductsForm, MailForm, and WinnerDeterminator classes. Also VCL classes are used for implementing GUI and some other functionality as seen in Appendix-A A7.

CATMainForm class is the class where the main operations are done. The functionalities of the Server Admin and CATManager are implemented in this class. This class also has attributes for implementing GUI from VCL.

ProductsForm class is used to preview the created products HTML page. This class also has attributes for implementing GUI from VCL.

MailForm class sends results to bidders. This class also has attributes for implementing GUI and sending messages over SMTP, from VCL. To send e-mails the SMTP service is used that is provided by windows 2000.

WinnerDeterminator class is derived from TThread class of VCL. It implements the IP formulation model to find winners.

3.2.2.3 User Interface of CATool

In this section brief description of the CATool will be presented by presenting sample views of the system.

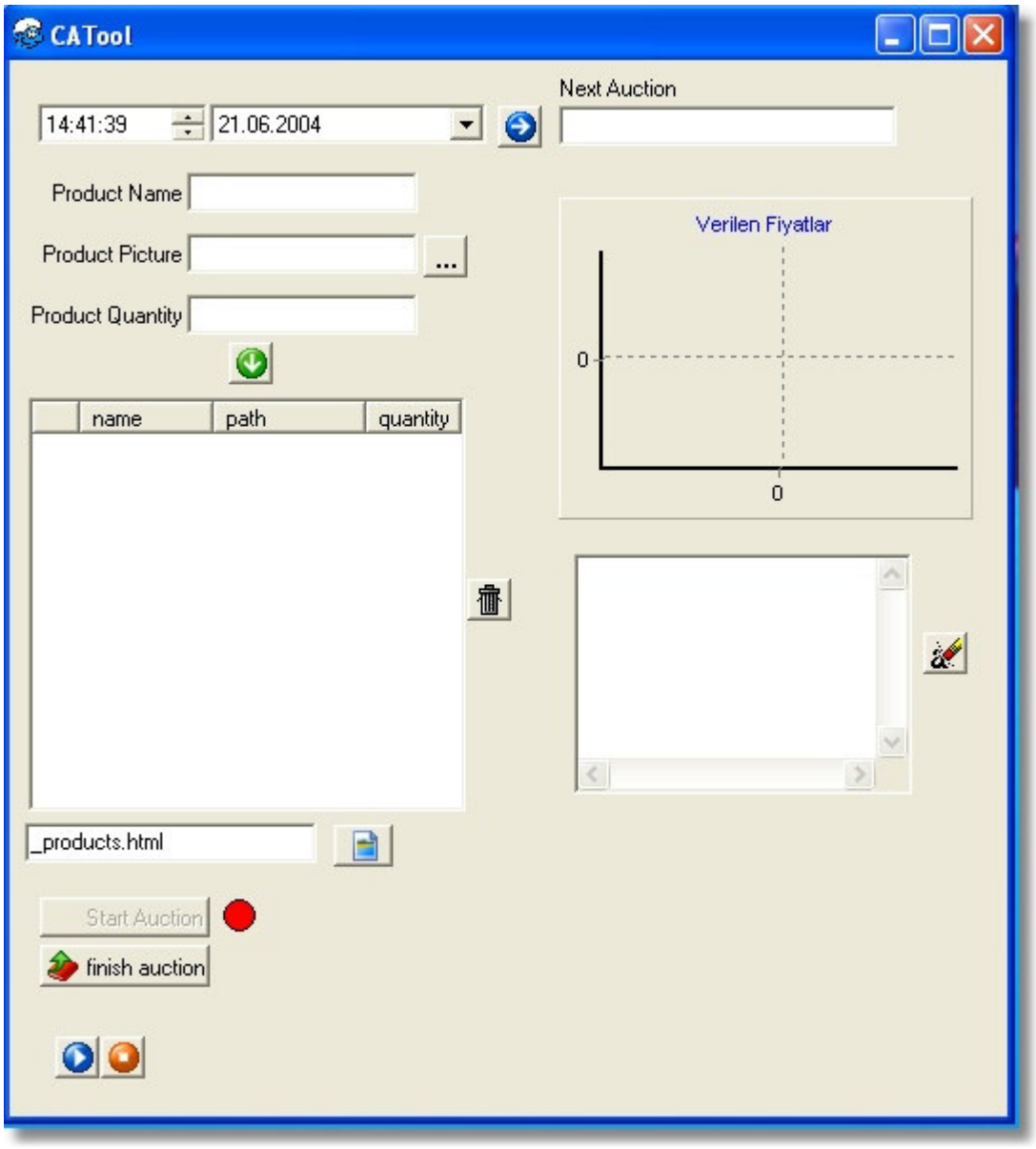


Figure 12 CATool starting screen

The starting screen of the CATool can be seen in Figure 12. The user here can set the auction end time, add the products that will be tried to be sold on the auction, create the

products html and previews it, start the auction[15] and HTTP server, and can see the logs of the system. The path of the CATool is found when the starting form is created. So when the HTTP server is created and for example for localhost 127.0.0.1 is entered automatically the index.html (which is in the same directory of the CATool) can be seen in the browser of the client. Index.html can be seen in Figure 13

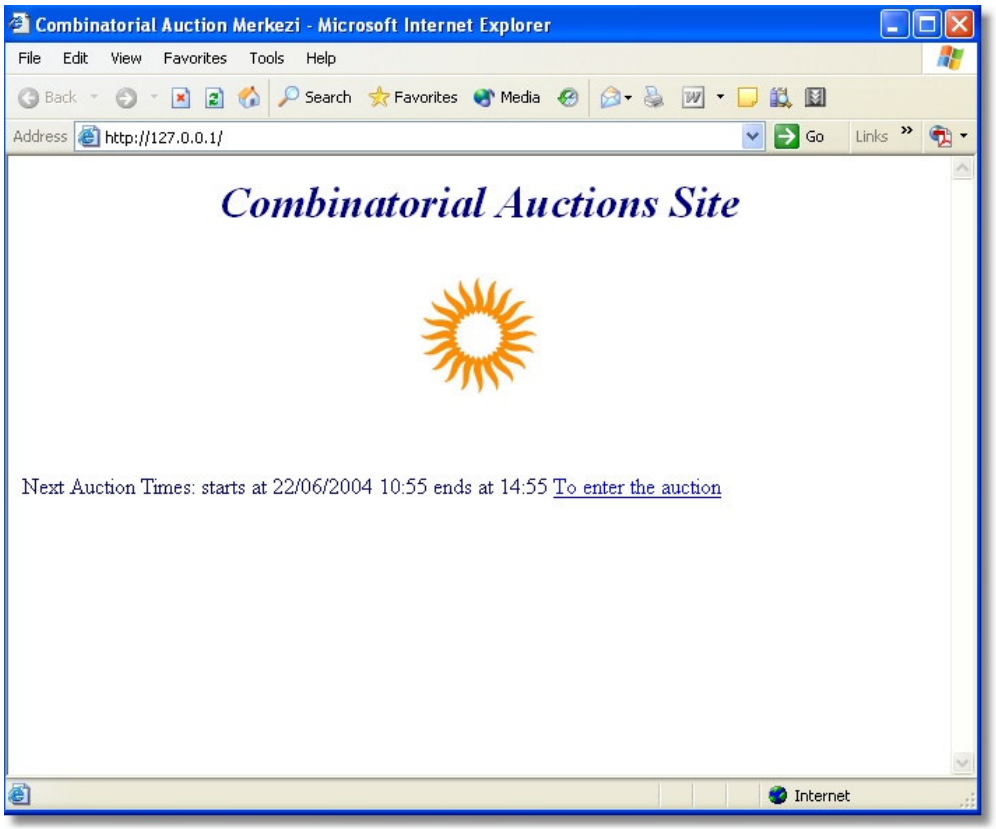


Figure 13 Index.html

In Figure 14 we can see the view, after the products are added and CreateHtml button is clicked. When the CreateHtml button is clicked the products html page is created by the given name and the preview is presented to the user.

---

[15] As seen in Figure 12 when the auction is not started the status bubble is in red, when the auction is started it turns to green and when the auction finishes it turns into blue until the updates are done and again it turns back to red.

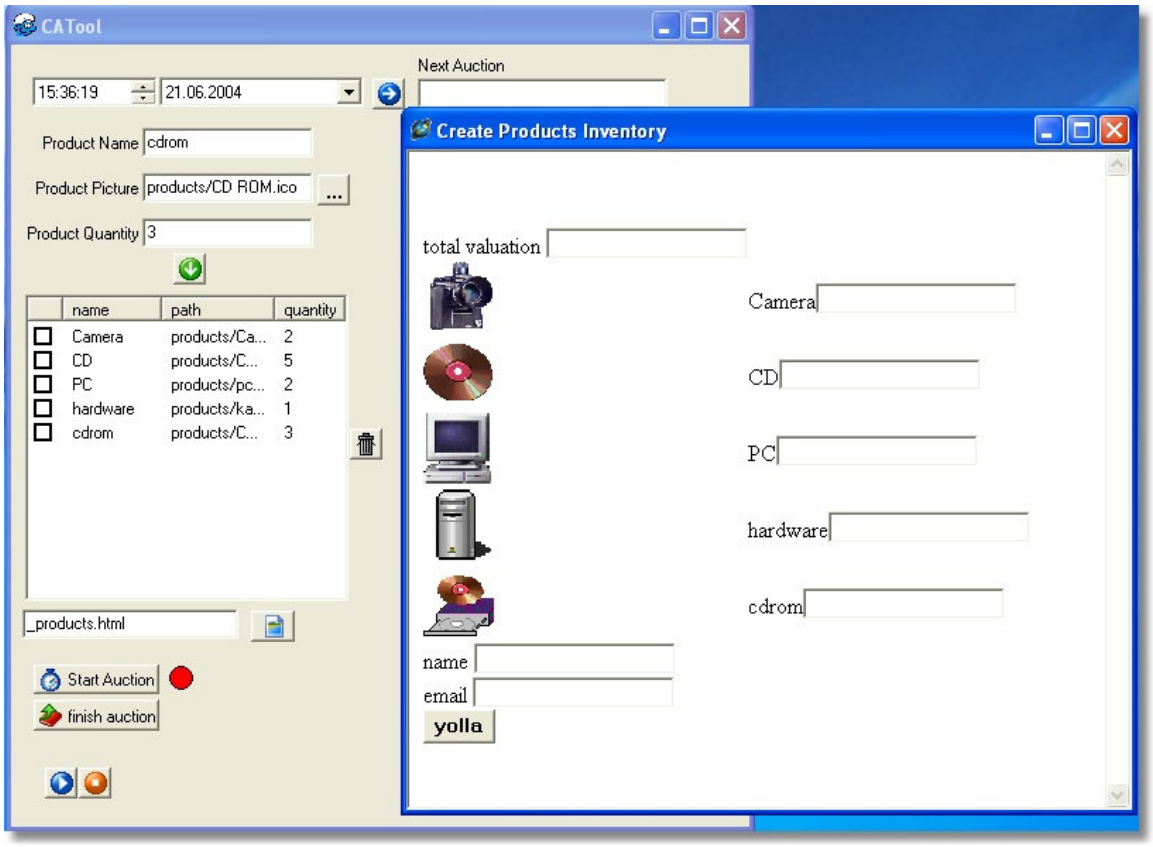


Figure 14 Preview of the created products Html

The screen after the auction end time arrives and winners are found can be seen in Figure 15. It can be seen that the graph is dynamically updated when the bidders give bids. And when the auction ends the status bubble turns into blue and the logs indicate that winners are found. Then when finish auction button is clicked the Mail Form is created as seen in Figure 16. In this form winners and losers are listed. And when Ok button is clicked e-mail is send to bidders and the graph, which is seen in Figure 15, is attached to the ongoing e-mails. After closing this form, the products and resources are also updated.

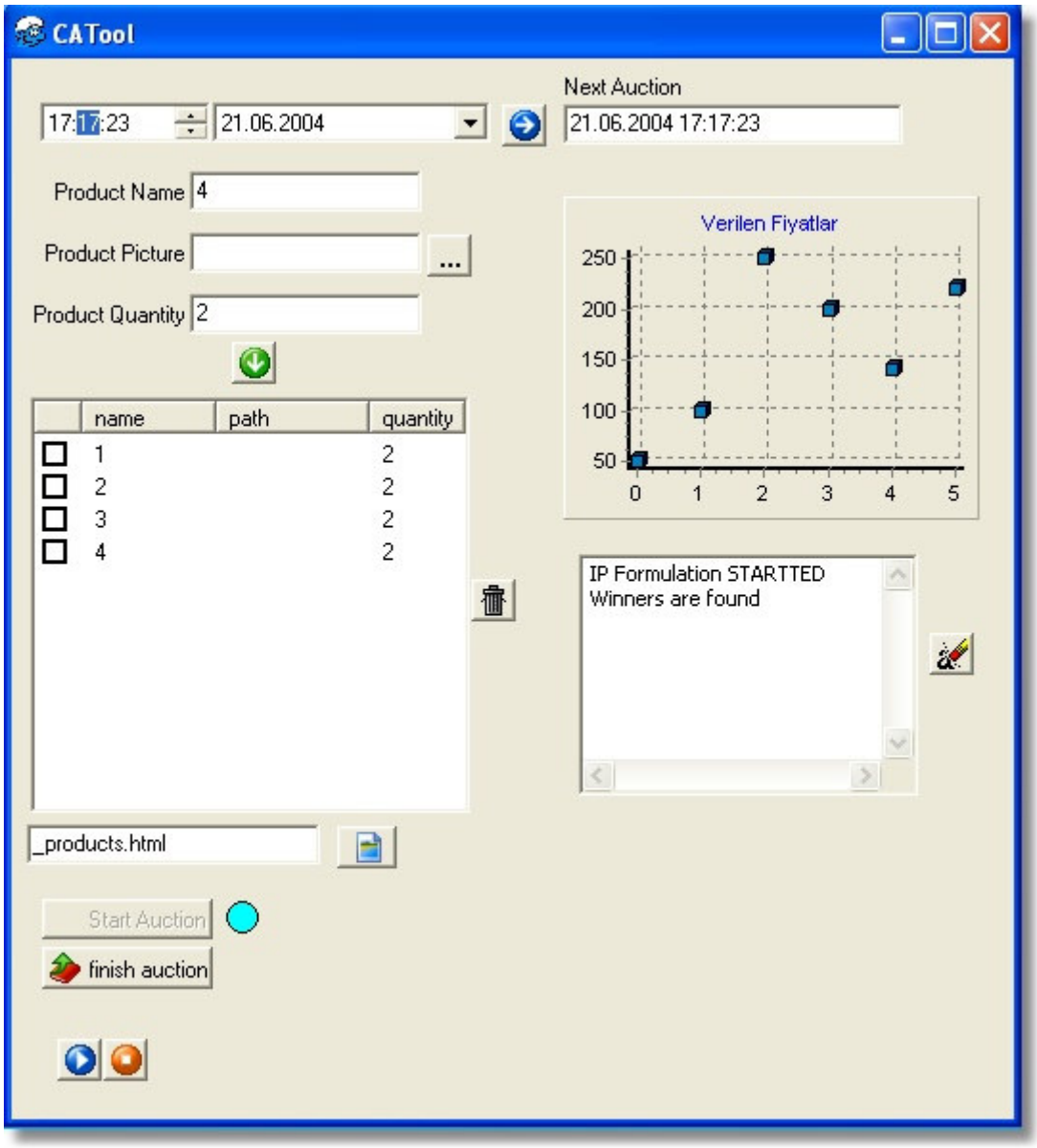


Figure 15 After the auction end time arrives and the winners found

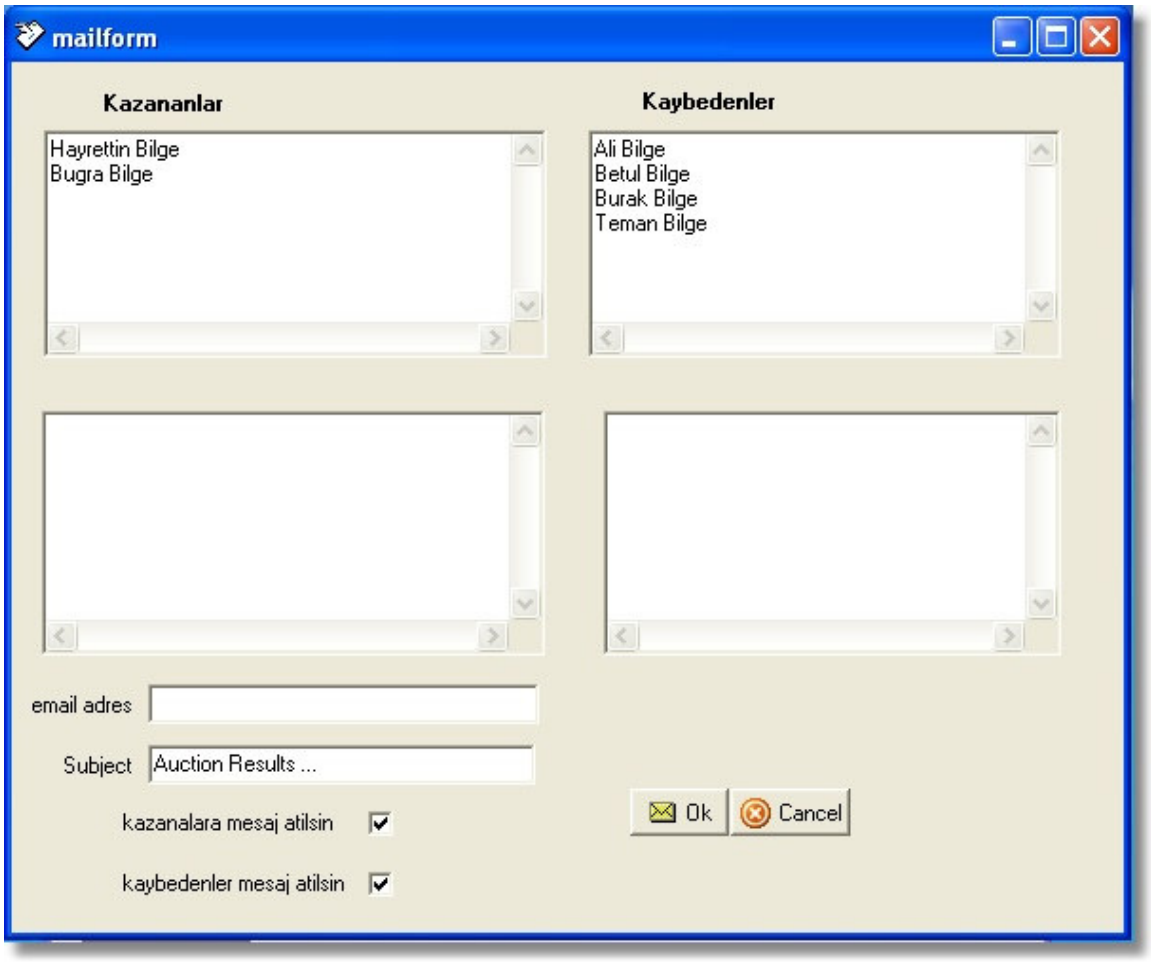


Figure 16 Mail Form

CHAPTER 4

Experimental Results

In this section, simulation model will be explained and the simulation results for the three models that explained in Chapter 3 will be presented.

4.1 Introduction

Auctions have been proposed and applied to perform contract negotiation and resource allocation in datagram networks and reservation-based networks. Many different approaches can be found in the literature.

Bandwidth allocation is one of the well known examples where preferences of bundles situation occurs, it can be modeled as multi-item multi-unit auctions which is one of the most complex situation that occurs in CAP. As stated before many of the previous studies can not be used in multi-unit settings. As a result, in order to analyze the performance of the three models built in order to solve CAP, we model bandwidth allocation system and by simulation we compare their performances.

4.2 Simulation Model

We model the network as in [30]. We choose a linear network to model as in Figure 17 which is taken from [30]. This is a network of N nodes. Let n denote a node ID, where n = {0, 1, 2, …, N}. It is supposed that the distance between any two adjoining nodes n-1, n is equal, and W denotes the total capacity of each link.

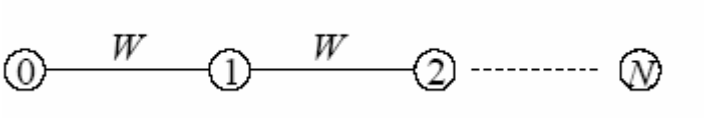


Figure 17 Network Model (from [30])

4.3 Simulation Parameters

Some parameters are as in [30]

- The last node ID N: 9
- Capacity of each link W : 9
- Number of users i : 10 -30
- Demand for bandwidth of each user w_i is random between 1 and 5
- Demand for links of user i (n_{i1}, n_{i2}) is random
- Total valuation of user i is random (it is calculated by unit valuation* w_i* $|n_{i1}-n_{i2}|$ all of which are random)
- Unit price of a link is random and initially all of the links unit price is same
- Unit reserve price is 0 for all of the links

We also assume that the user say true valuations.

4.4 Evaluation Criteria

To evaluate the performance of three models, we look at the following evaluation criteria

- Total revenue of the network which is the sum of total valuations of the users that are allocated resource, can be seen in Figure 18[16]
- Bandwidth allocation percentage which indicates what percent of the available bandwidth is allocated to users, can be seen in Figure 19
- Time to find the solution, can be seen in Figure 20

---

[16] Since the revenue of DSAM also depends on the price reduction policy. But since the users are ready to give their valuation, in the simulation results for DSAM the revenue are calculated from the winner's valuations as other models (not from the price of the unit link at the time of allocation)

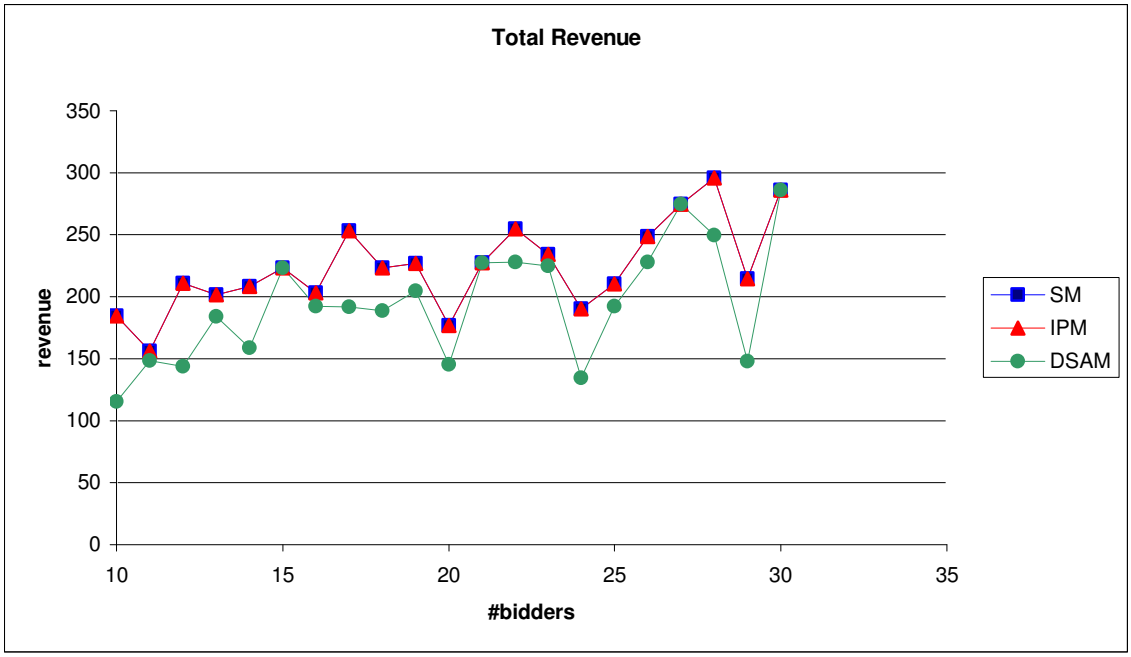


Figure 18 Total revenue of the network

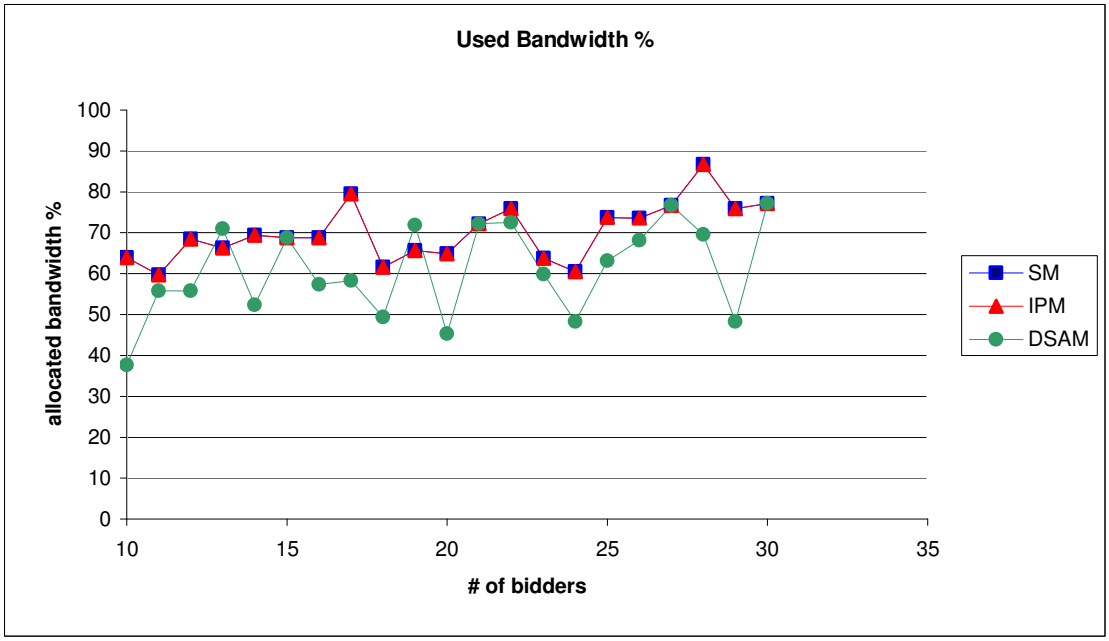


Figure 19 Bandwidth allocation percentages

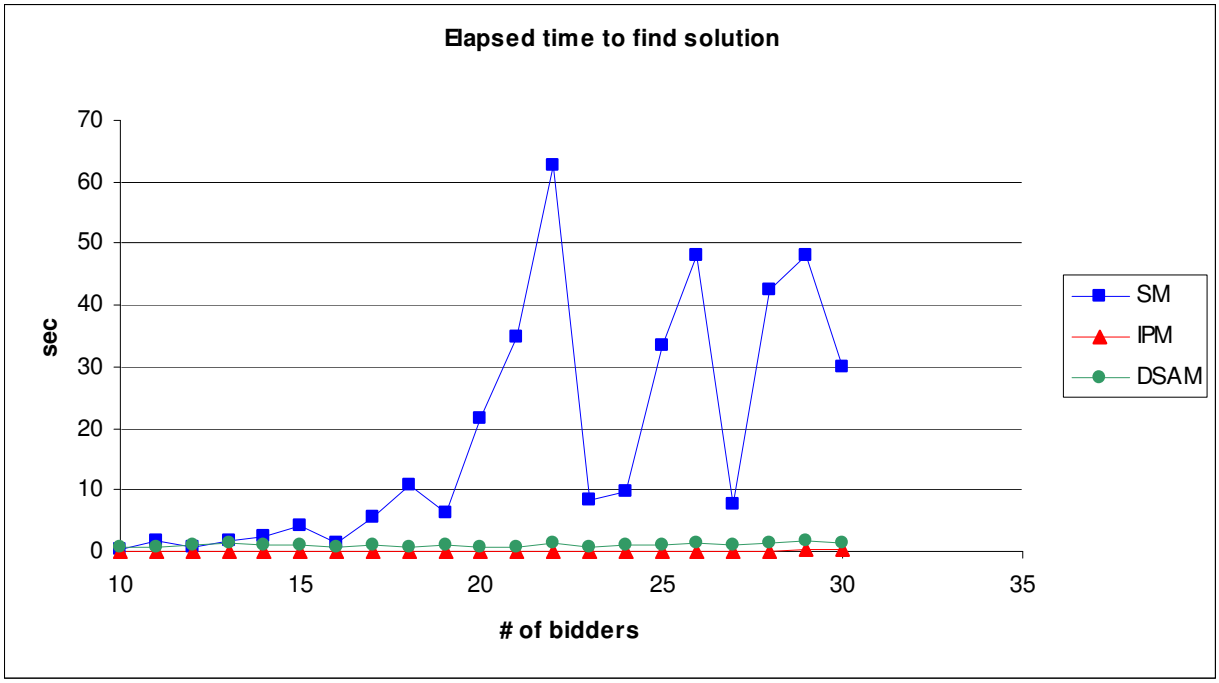


Figure 20 Time to find the solution

As seen from Figure 18 the total revenue is the same for SM and IPFM, and they are higher than DSAM. This is because, SM finds the optimum solution given enough time as stated before, and DSAM finds the optimum solution, if the corresponding MIP has an optimum solution. Since the formulations are correct and time is enough both SM and IPFM finds the optimum solutions, but DSAM could not always find the optimum solutions. This is a normal condition because DSAM is a descending simultaneous auction model and as explained before it is not designed to find optimum allocation. But when we look at Figure 20 we see that DSAM and IPFM are faster than SM. Here when we consider the three graphs we can say that IPFM is the most advantageous model, because it finds the optimum solution and the fastest one. But for big numbers (bidders and items i.e. more complex situation) we can't say that IPFM always is the fastest. Since the GLPK has variable limits when solving MIP (currently 100 variables).

Other simulation runs can be seen in Appendix-B. Here from the runs we see that DSAM's revenue is 87.43% of SM and IPFM's revenue. DSAM's allocated bandwidth is 84.74% of SM and IPFM's allocated bandwidth.

SM can be made faster by finding better upper bounds also Sandholm et al. has listed some techniques, which can be used at each search node to identify and solve tractable special cases [26]. SM can be made faster by also using these techniques. Some of these techniques are

- Avoiding branching on short bids
- Deleting items included in only one bid
- Interval bids. Sandholm et al. prove that if all n bids are interval bids in a linearly ordered set of items [1,m], then an optimal allocation can be computed in worst-case time O(n+ m).
- Identifying linear ordering. Sandholm et al. state that their interest is not to limit the auctions to interval bids only, but rather to recognize whether the remaining problem at any search node falls under this special case and to solve it by their specialized fast algorithm. This requires an algorithm to check whether there exists some linear ordering of items for which the given set of bids are all interval bids. It turns out this problem as the interval graph recognition problem, for which a linear-time solution exists [26]. The interval graph recognition problem is to decide whether G is an interval graph, and to also construct the intervals [26]. Algorithms on interval graphs that solves this problem can be found in the lecture notes of Gabriel Valiente (2002).

This simulation also gives a chance to compare the three well known approaches for multi-item multi-unit auctions. Many of the previous papers compare different approaches to solve the CAP, but they generally compare them by not considering multi-unit models except in [28]. So in this simulation three different approaches are compared as first time based on the settings given in chapter 4.

CHAPTER 5

Conclusion and Future Work

When we survey some of the previous works that are done on Combinatorial Auctions Problem, we see that the previous works can be categorized into three parts. One part tries to solve the unrestricted problem using search algorithms [23, 24, 26], second part tries to solve the problem using OR (Operation Research) methodologies and/or deals with the identification of tractable cases of the combinatorial auctions problem [31, 32] and the last part deals with the usage of sequential or simultaneous auctions in order to solve the combinatorial auctions problem [3, 4, 19].

As can be seen in chapter 2, many of the previous studies were for single-unit Combinatorial Auctions Problem and many of the previous studies cannot be used in multi-item multi-unit settings. Also many of the previous studies compare different approaches to solve the CAP, but they generally compare them by not considering multi-unit models excepts in [28]. As a result, in this study, we decided to focus on multi-unit combinatorial auctions (multi-item multi-unit auctions in which bidders have preferences over bundles and their valuation for the bundle is not additive). We decided to compare different approaches in order to solve multi-unit combinatorial auctions problem, based on the settings given in chapter 4. We built three different models from three different approaches. We compare their performances using computer simulations where we model Bandwidth allocations system based on the settings given in chapter 4. And we decided to build a simple combinatorial auction tool which can be used for online combinatorial auctions and can be used as a test bed for combinatorial auctions.

The first model we build tries to solve the CAP using search methods. The search algorithm of this model is very similar to Sandholm's sophisticated search algorithm but with some modifications [24, 26, 27]. This model finds the optimal solution when given enough time. The second model is built by considering the simultaneous auctions mechanisms used in to solve the CAP. We were inspired from Courcoubetis et al. for the second model [4]. In this one we built a descending simultaneous auction mechanism. In the last model we developed an IP formulation like the one given in section 2.7 and used the GNU Linear Programming Kit (GLPK)'s callable library in order to find the solutions. This model also finds the optimum solution, if the corresponding MIP has an optimum solution and it is given enough time.

The performance of each model is analyzed and compared with the other models using computer simulations, where we model bandwidth allocation among multiple users. Because, as stated before, bandwidth allocation is one of the well known examples where preferences of bundles situation occurs, it can be modeled as multi-item multi-unit auctions which is one of the most complex situation that occurs in CAP. Most of the previous studies can not be used in multi-unit settings. The simulation results of Chapter 4 gives a chance to compare the three well known approaches for multi-item multi-unit auctions based on the settings given in chapter 4. Many of the previous papers compare different approaches to solve the CAP, but they generally compare them by not considering multi-unit models except in [28]. So in our work, the three approaches are compared as the first time based on the settings given in chapter 4. The results have shown that the IP formulation model was advantageous for considerable numbers of bidders and items for the used simulation settings.

Based on the performance of the models and some other situations, we choose the IP formulation model and built a simple Combinatorial Auction tool (CATool) which can be used for online auctions and e-procurement systems. The CATool uses IP Formulation model in order to find winners. This tool is one of the first tools which can serve as a web server and finds the winners using the IP Formulation.

As a future work, SM can be improved by finding better upper bounds and also identifying and solving tractable special cases as explained in chapter 4. Some functionalities can be added to the CATool like personalization. Also the simulation model can be made more complex including the bursty traffic conditions, which can occur in bandwidth allocations and causes the variation of the required bandwidth. Also other simulation models where preferences over bundles situation occurs, like procurement of indirect materials, logistics marketplace and etc. can be built. By modeling different user behaviors and marketplaces a better analysis on the three models can be made. Building a more complex simulation model can also be a field of research.

REFERENCES

[1] Arpınar, S., Doğaç, A., &Tatbul N. (2000, July). An Open Electronic Marketplace through Agent-based Workflows: MOPPET. International Journal on Digital Libraries, 3(1), 36-59.

[2] Benyoucef, M., Alj, H., Vézeau, M., &Keller, R. K. (July 2001). Combined Negotiations in E-Commerce: Concepts and Architecture. Electronic Commerce Research Journal Special issue on Theory and Application of Electronic Market Design, 1(3), 277-299.

[3] Biswas, S., &Narahari, Y. Iterative Dutch Combinatorial Auctions., To appear in Special issue of Annals of Mathematics and Artificial Intelligence on the Foundations of Electronic Commerce.

[4] Courcoubetis, C., Dramitinos, M.P, &Stamoulis, G.D. (2001, December). An Auction Mechanism for Bandwidth Allocation over Paths. 17th International Teletraffic Congress (ITC)

[5] Dawid, H. (1999). On the convergence of genetic learning in a double auction market. Journal of Economic Dynamics & Control, 23, 1545-1567.

[6] eBay; URL: www.ebay.com

[7] Menczer, F., Street, W. N., &Monge, A.(2002). E. Adaptive Assistants or Customized E-shopping. IEEE Intelligent Systems 17(6), 12-19.

[8] Fox, M.S., Barbuceanu, M., & Teigen R.(2000). Agent-Oriented Supply-Chain Management. The International Journal of Flexible Manufacturing Systems, 12, 165-188.

[9] Weiss, G. (1999). Multiagent Systems A Modern Approach to Distributed Artificial Intelligence. MIT Press

[10] GLPK home page URL: http://www.gnu.org/software/glpk/glpk.html

[11] GNU Linear Programming Kit Reference Manual.

[12] Guttman, R.H., & Maes, P. (1998). Cooperative vs. Competitive multi-agent negotiations in retail electronic commerce. Proceedings of the Second International Workshop on Cooperative Information Agents (CIA'98), Paris, France, July 3-8

[13] Hans Mittelmann's webpage URL:http://plato.asu.edu/bench.html

[14] Hunsberger, L., Grosz, B.J. (2000). A Combinatorial Auction for Collaborative Planning. In Proceedings of the Fourth International Conference on Multi-Agent Systems (Boston, Massachusetts), IEEE Computer Society Press, pp. 151-158.

[15] Jap, S.D (2002). "Online Reverse Auctions: Issues, Themes, and Prospects for the Future". Invited article for the Marketing Science Institute- Journal of the Academy of Marketing Science Special Issue on Marketing to and Serving Customers through the Internet: Conceptual Frameworks Practical Insights and Research Directions. 30(4), 506-25.

[16] Klemperer, P. (May, 1999). Auction Theory: A Guide to the Literature. Journal of Economics Surveys, 13(3), 227-286.

[17] Lee, K.Y., Yun, J.S., & Jo, G.S. (2003). MoCAAS: auction agent system using collaborative mobile agent in electronic commerce. Expert Systems with Applications, 24, 183-187

[18] Maes, P., Guttman, R. H, &Moukas, A. G. (1999) Agents that buy and sell. Commun. ACM 42(3), 81-91.

[19] Milgrom, P. (2000, April). Putting Auction Theory to Work: The Simultaneous Ascending Auction. Journal of Political Economy

[20] Park, S., &Miller, K.(1998) Random Number Generators: Good Ones Are Hard To Find . Communications of the ACM

[21] S. Russell, P. Norvig (1995). Artificial Intelligence: A Modern Approach, Prentice Hall, Englewood Cliffs, NJ

[22] Sakurai, Y., Yokoo, M., &Kamei, K. (2000). An Efficient Approximate Algorithm for Winner Determination in Combinatorial Auctions, Second ACM Conference on Electronic Commerce (EC-00)

[23] Sandholm, T. (2000). Approaches to winner determination in combinatorial auctions. Decision Support Systems, 24, 165-176.

[24] Sandholm, T. (2002). Algorithm for optimal winner determination in combinatorial auctions. Artificial Intelligence, 135, 1-54.

[25] Sandholm, T., &Huai, Q (2000). Nomad: Mobile Agent System for an Internet-Based Auction House. IEEE Internet Computing, 4(2), 80-86.

[26] Sandholm, T., &Suri, S. (2003). BOB: Improved winner determination in combinatorial auctions and generalizations. Artificial Intelligence, 145, 33-58.

[27] Sandholm, T., Suri, S., Gilpin, A., &Levine, D. (2001). CABOB: A Fast Optimal Algorithm for Combinatorial Auctions. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Seattle, WA

[28] Sandholm, T., Suri, S., Gilpin, A., &Levine, D. (2002). Winner Determination in Combinatorial Auction Generalizations. In the Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), Bologna, Italy, July pp. 69-76.

[29] Shardanand, U., &Maes, P., (1995). Social Information Filtering: Algorithms for Automating "Word of Month". Proc ACM Conf. Human Factors in Computing Systems (CHI 95), ACM Press, 210-127. URL:http://www.acm.org/sigchi/chi95/Electronic/documnts/papers/us_bdy.htm.

[30] Takahashi, E., &Tanaka, Y. (2001). Bandwidth Allocation by Using Generalized Vickrey Auction. Asia Pacific Symposium on Information and Telecommunication Technologies (APSITT2001), Kathmandu, Nepal/ Atami, Japan, pp. 233-237.

[31] Tennenholtz, M. (2002). Tractable combinatorial auctions and b-matching. Artificial Intelligence, 140, 231-243.

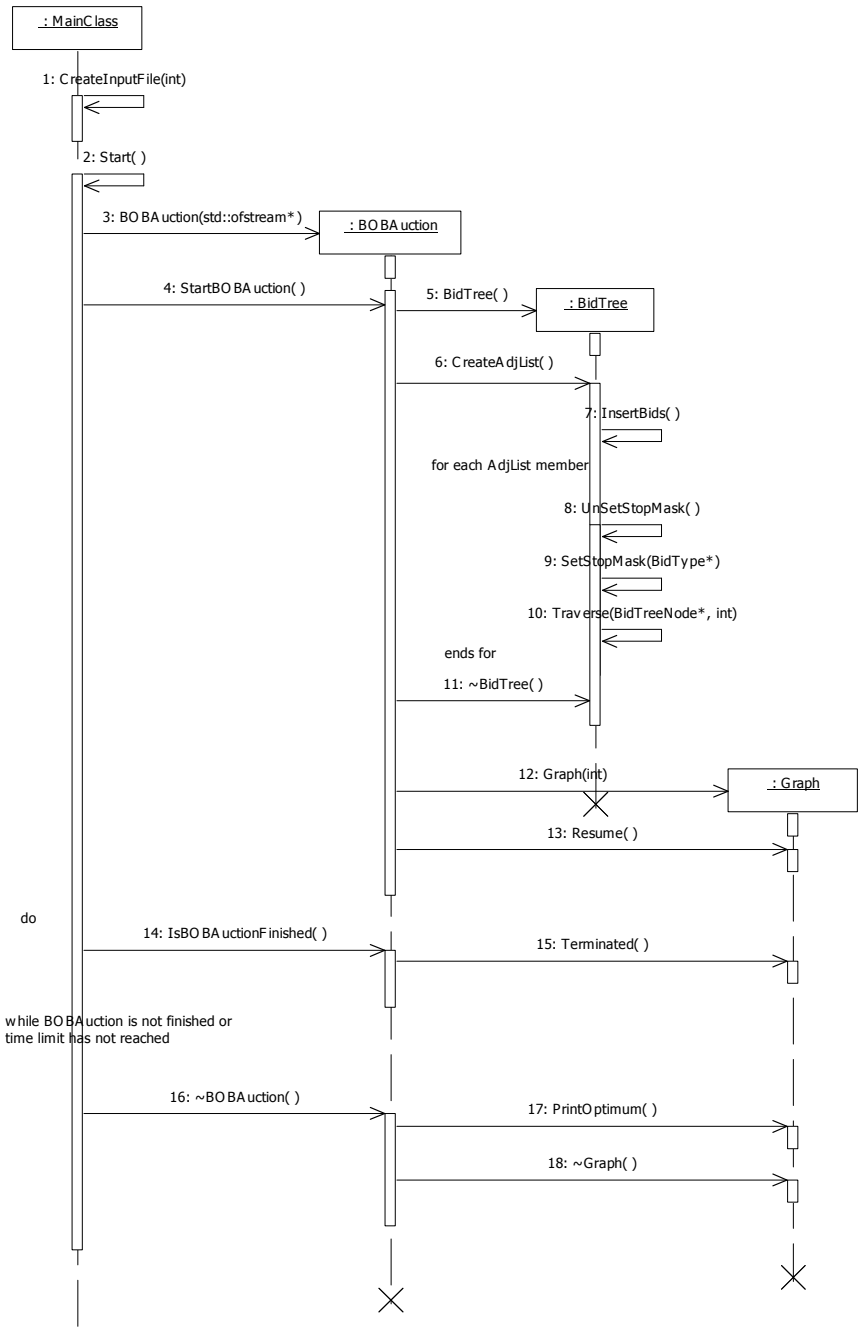
[32] Vries, S. de, &Vohra, R. (2003). Combinaorial Auctions: A Survey. INFORMS Journal on Computing, 15(3), 284-309.

[33] Weiss, M.A. Data Structures and Algorithm Analysis in C++, 2nd Edition. Addison-Wesley

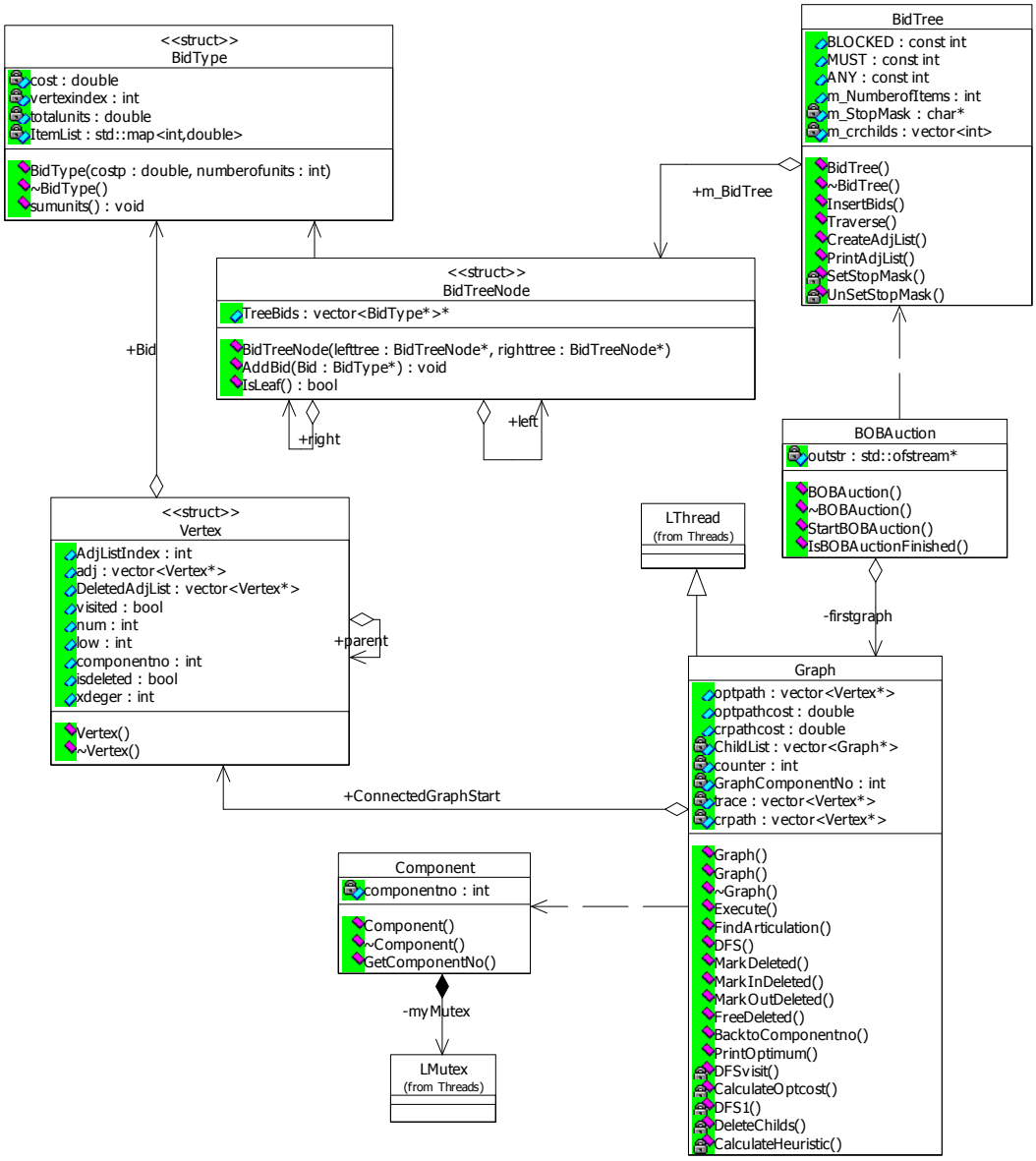
[34] Wurman , P. R., Wellman, M. P., &Walsh, W.E (1998). The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In Second International Conference on Autonomous Agents (AGENTS-98), pages 301-308, Minneapolis, MN

APPENDICES

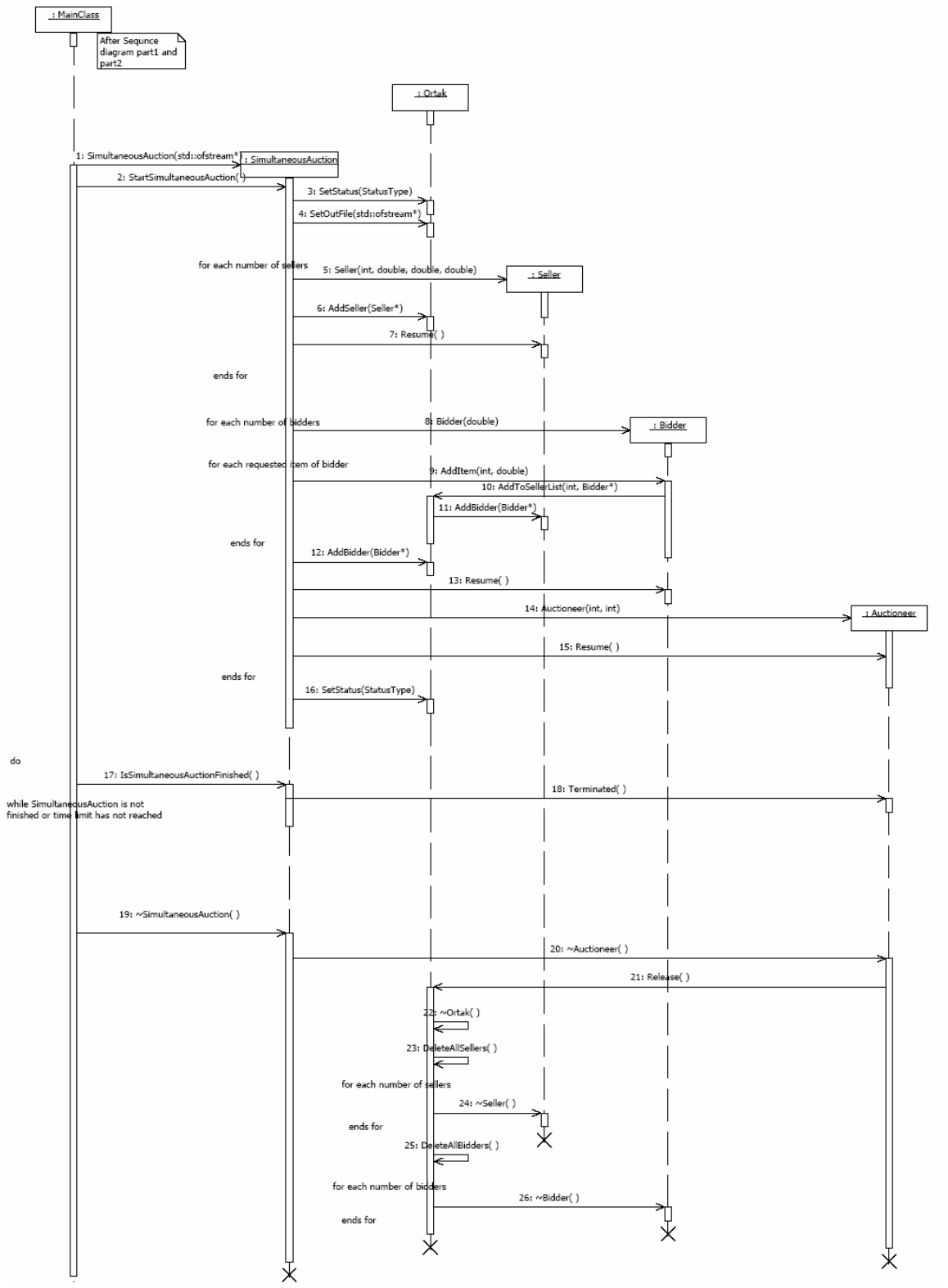
A: Class and Sequence Diagrams



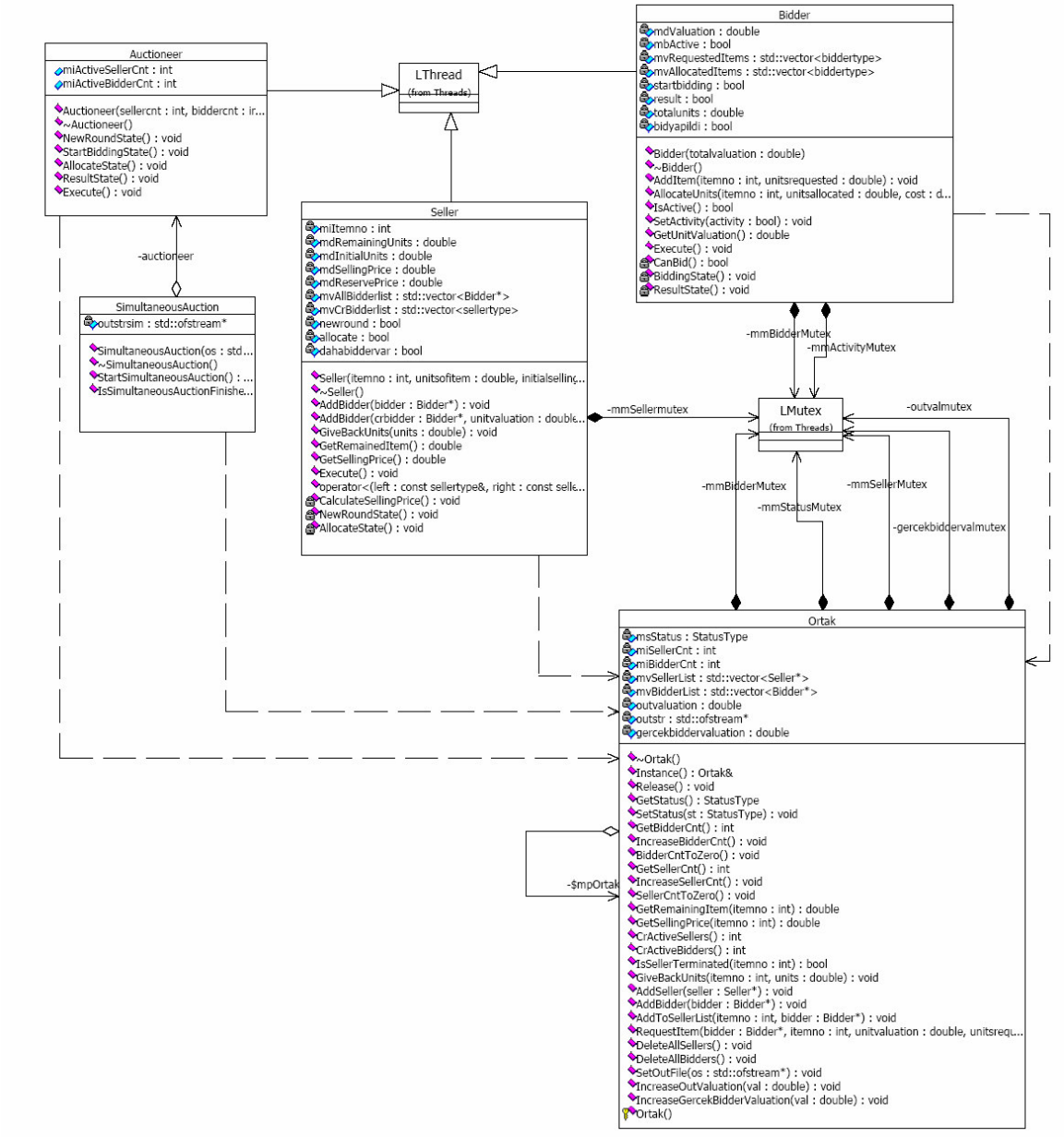
A 1 MainClass Initialization Sequence Diagram part1 (Initialization of the Search Model)



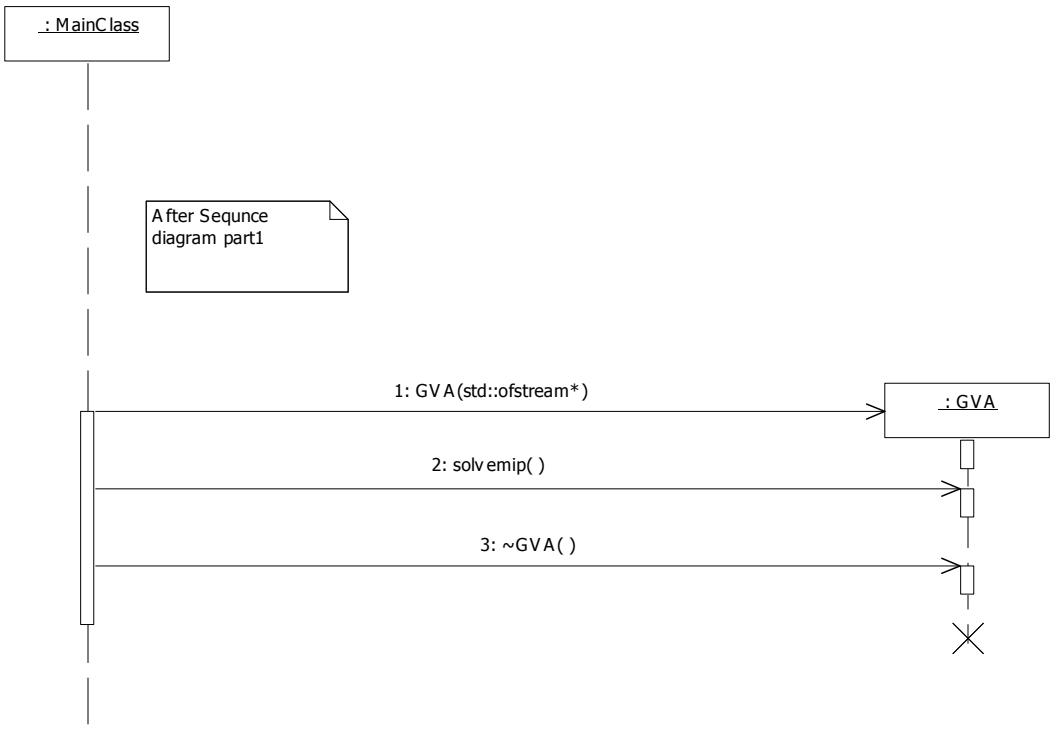
A 2 Class Diagram for BOBalgorithm Package



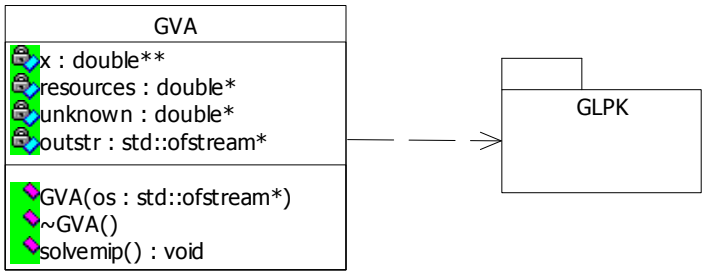
A 3 MainClass Initialization Sequence Diagram part3 (Initialization of the Descending Simultaneous Auctions Model)



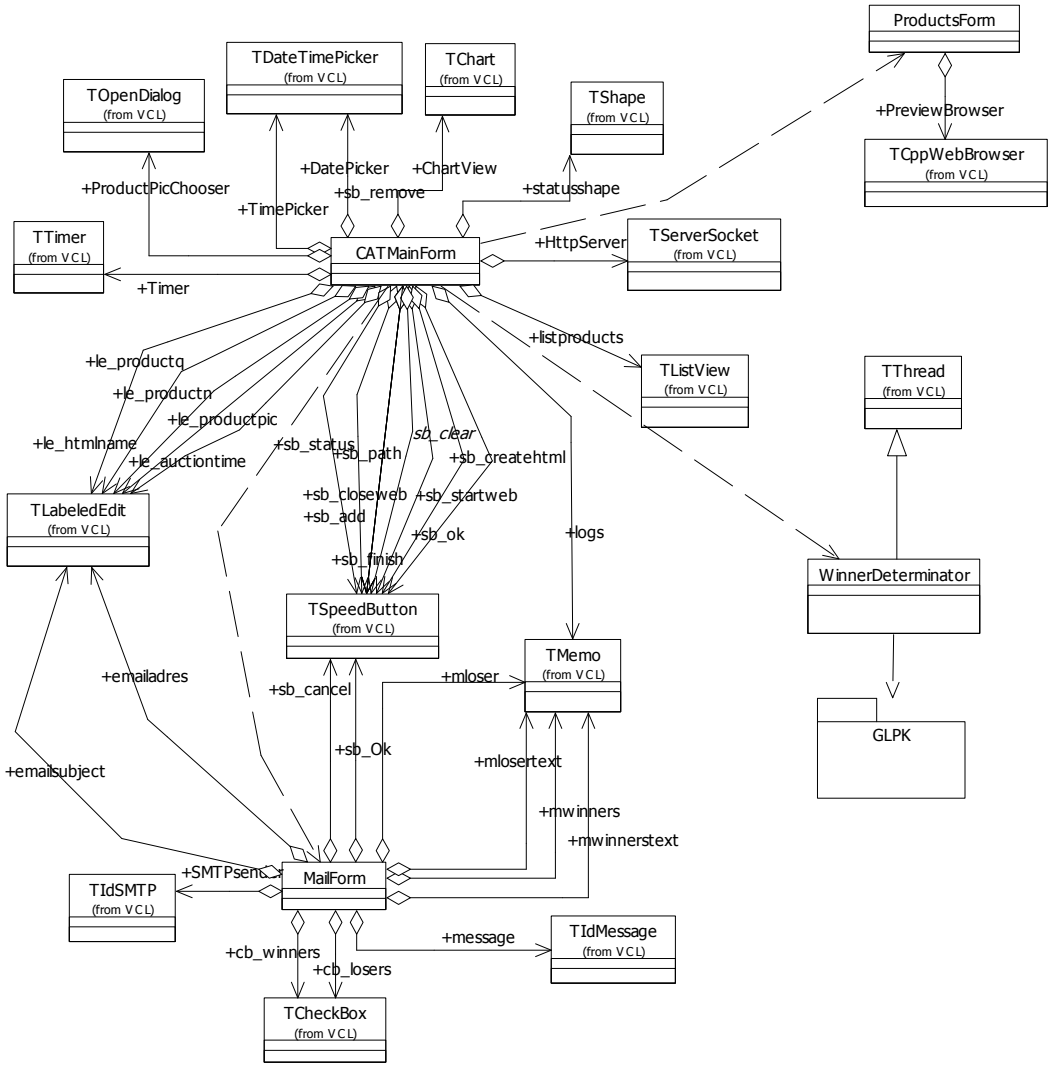
A 4 Class Diagram for SIMULalgorithm Package



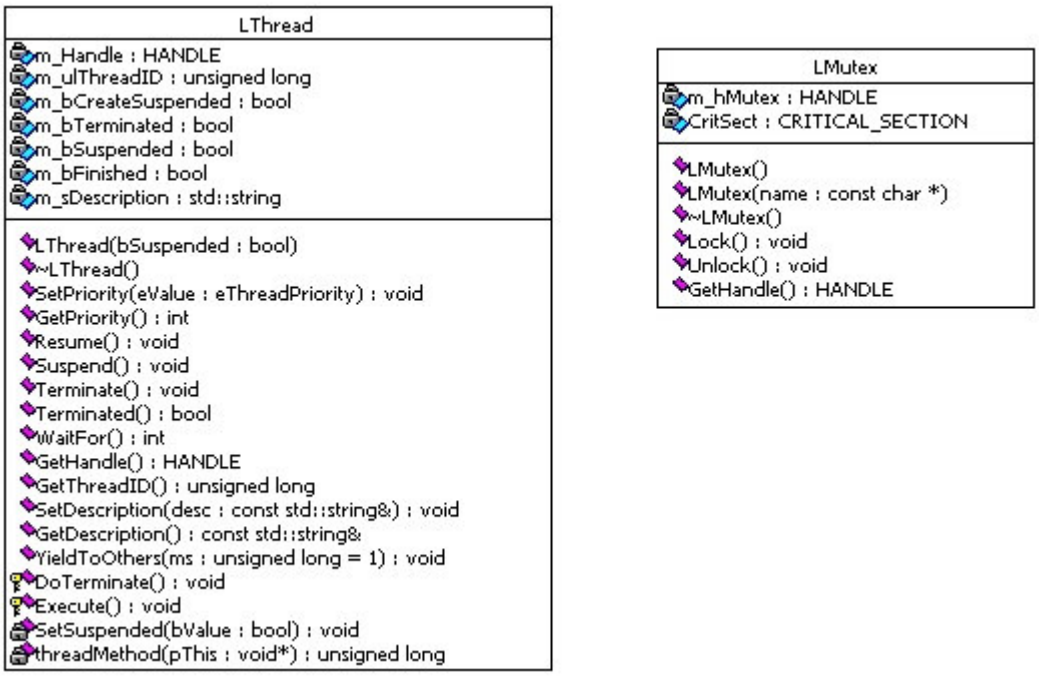
A 5 MainClass Initialization Sequence Diagram part2 (Initialization of the IP Formulation Model)



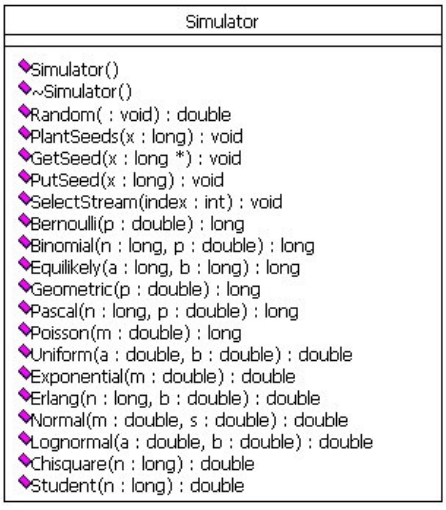
A 6 Class Diagram for GVAalgorithm package



A 7 Class Diagram of the CATool



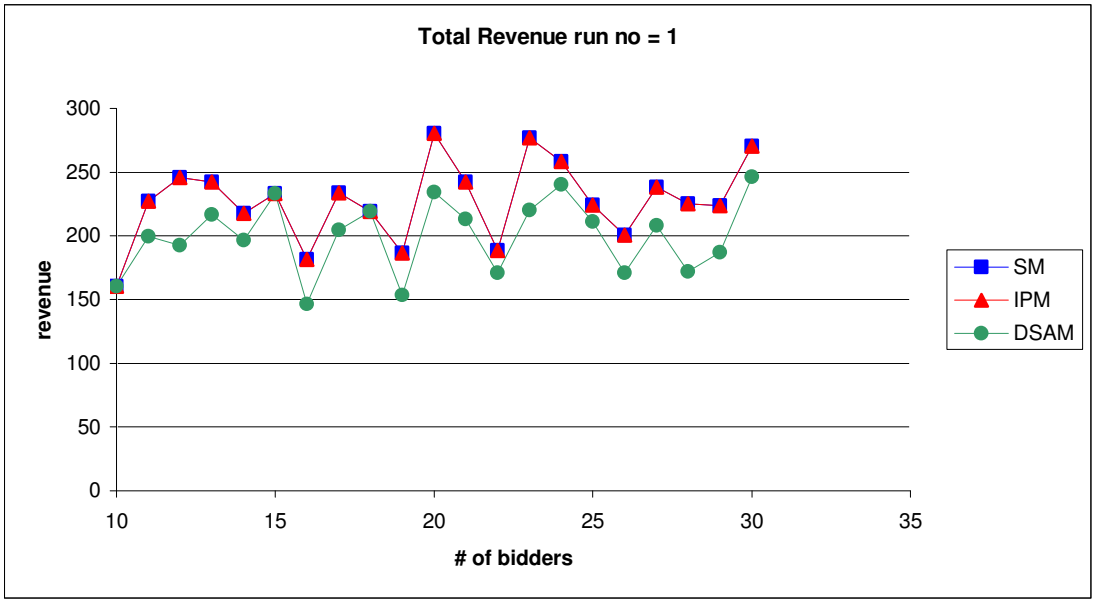
A 8 Class Diagram for Threads Package

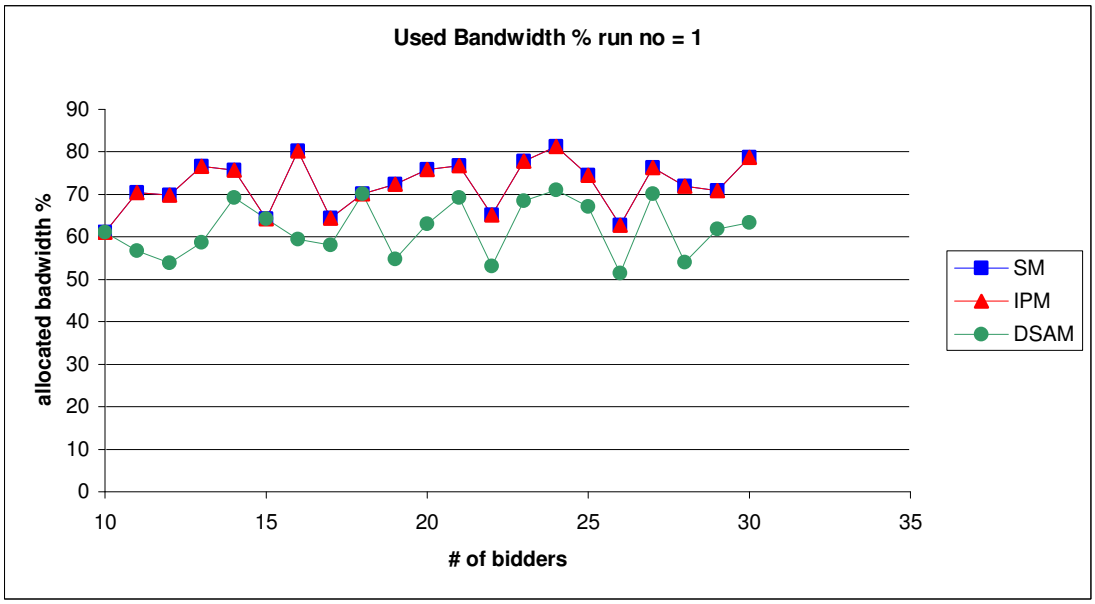


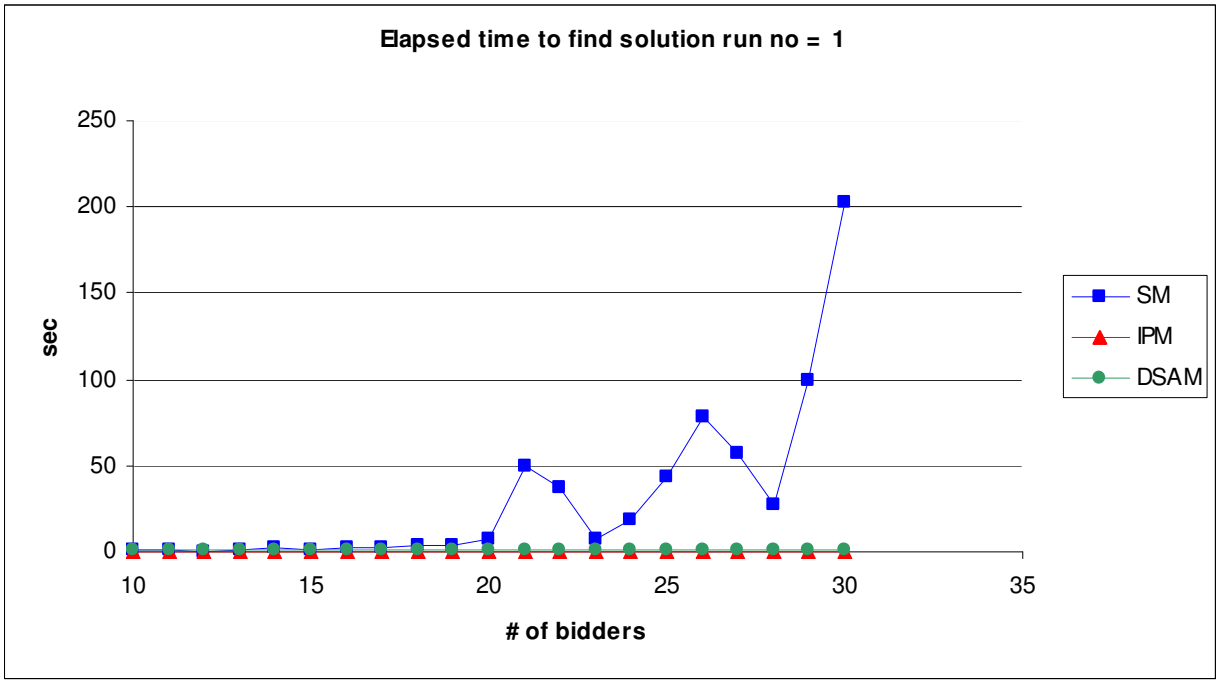
A 9 Class Diagram for Simulator Package

B: Simulation Runs

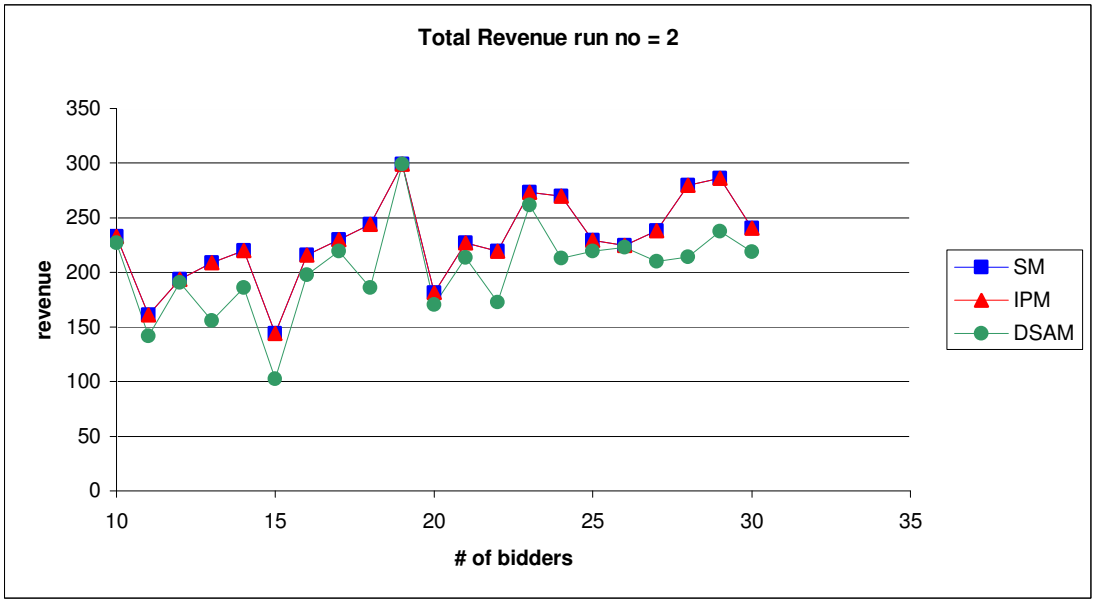
Simulation Run 1

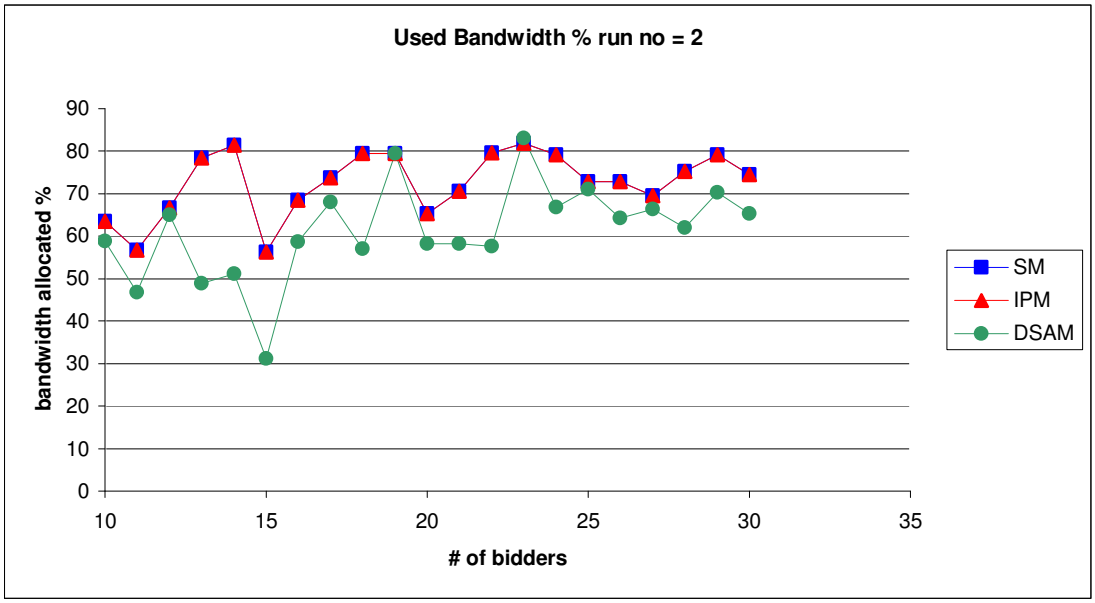


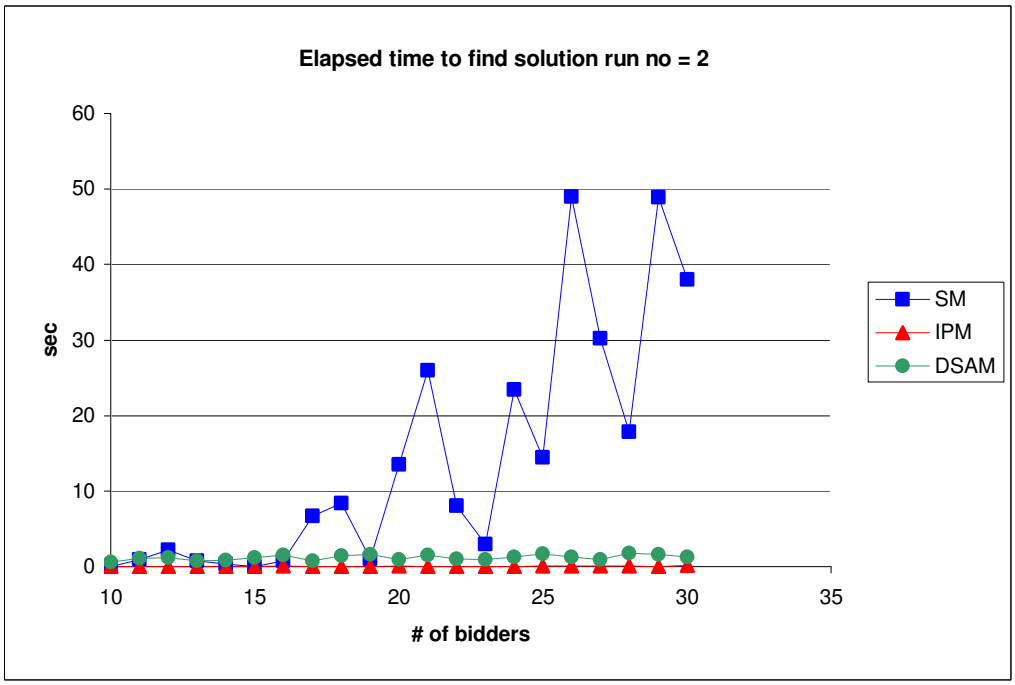




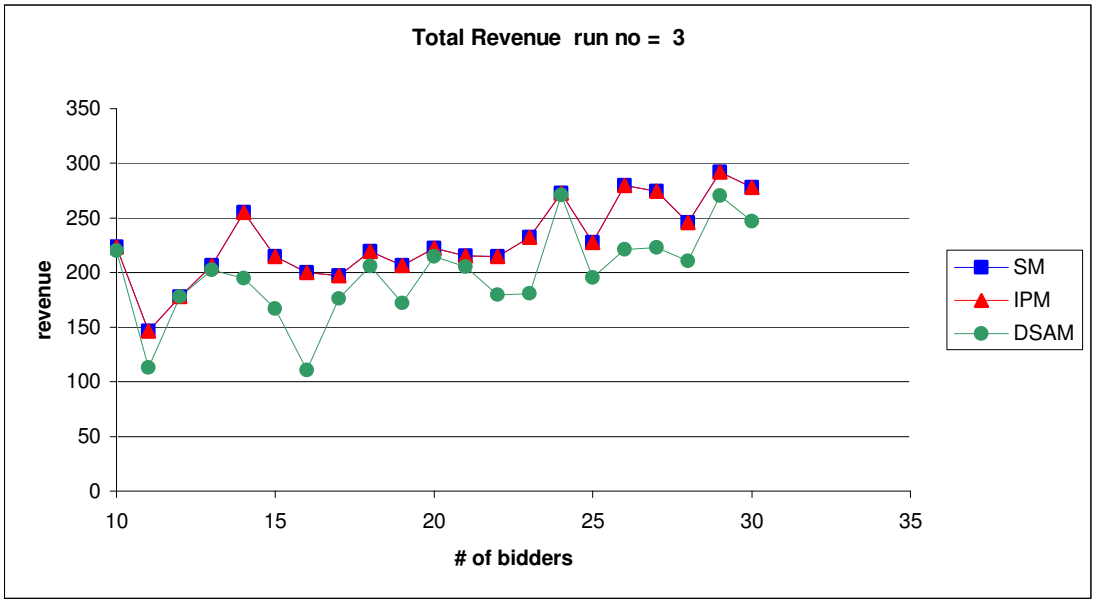
Simulation Run 2

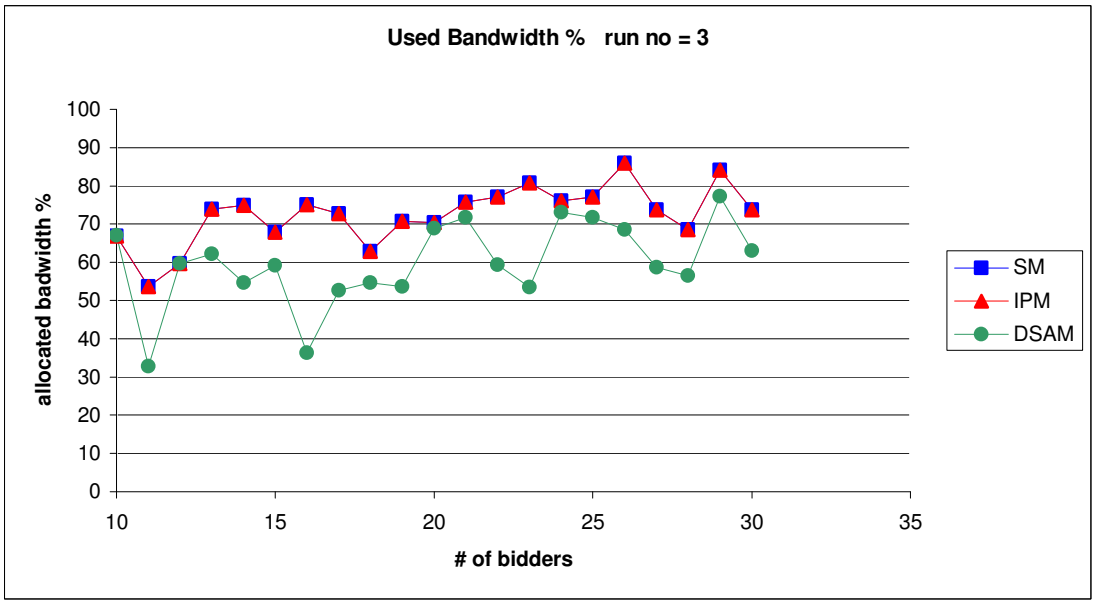


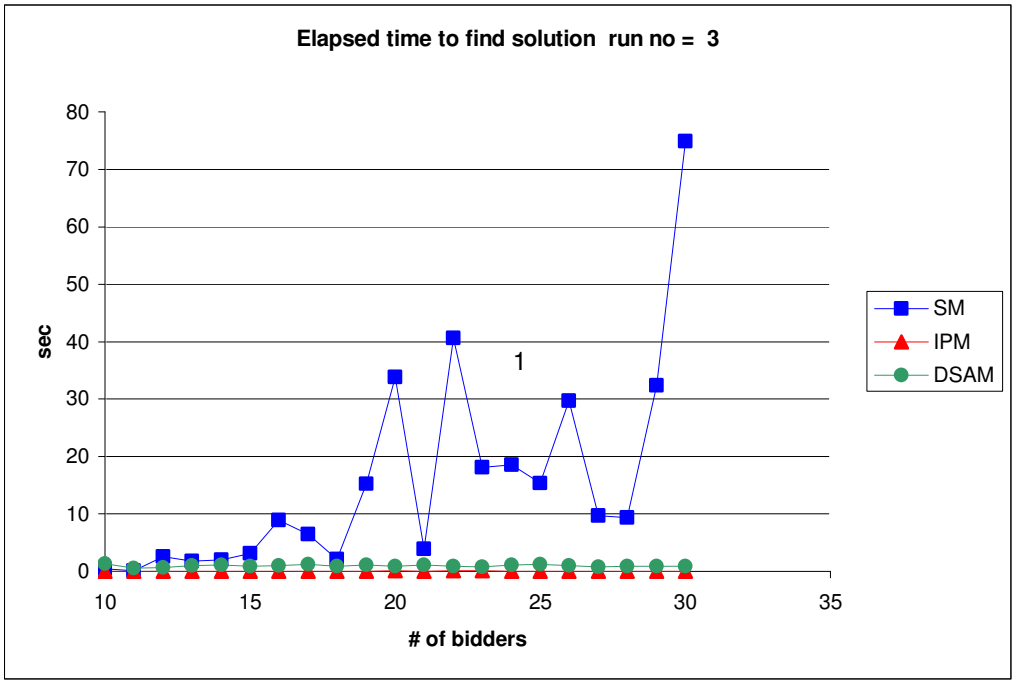

Used Bandwidth % run no = 2
bandwidth allocated %
90
80
70
60
50
40
30
20
10
0
10
15
20
25
30
35
# of bidders
SM
IPM
DSAM


Elapsed time to find solution run no = 2
sec
60
50
40
30
20
10
0
10
15
20
25
30
35
# of bidders
SM
IPM
DSAM

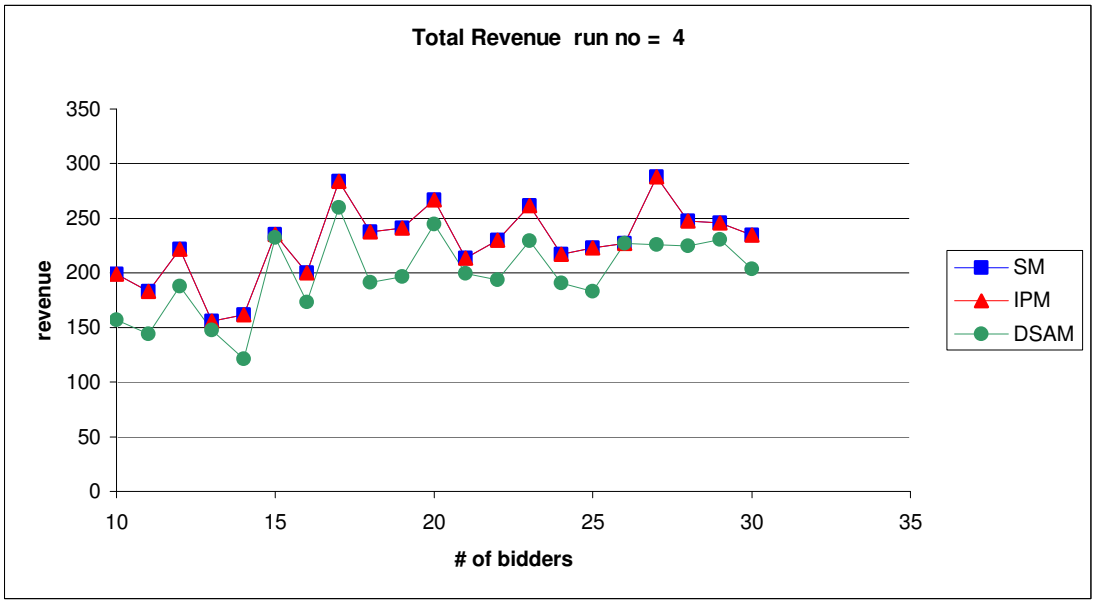
Simulation Run 3

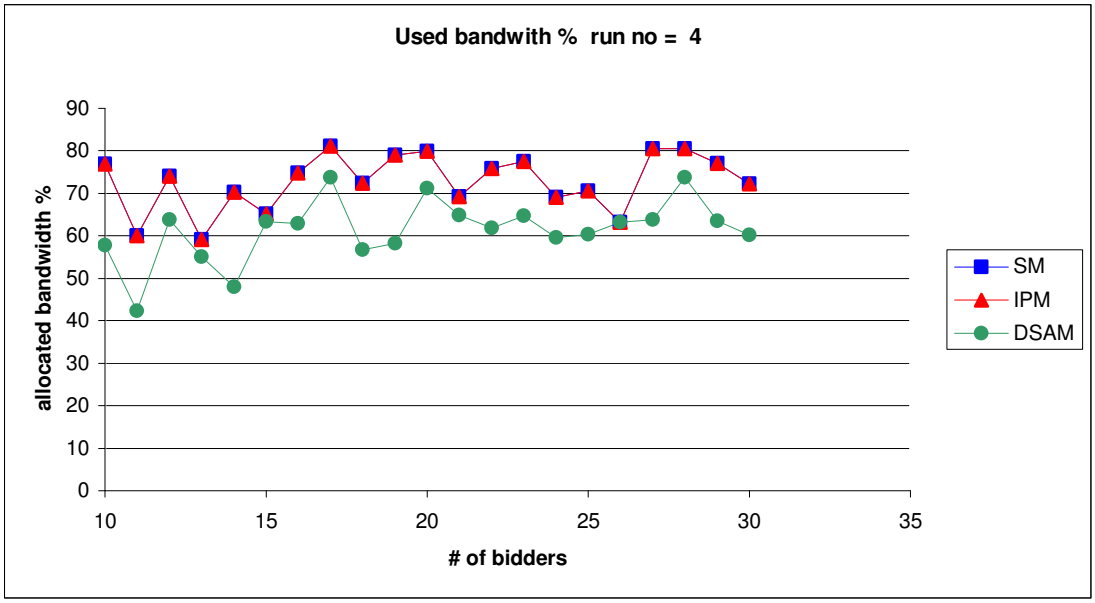


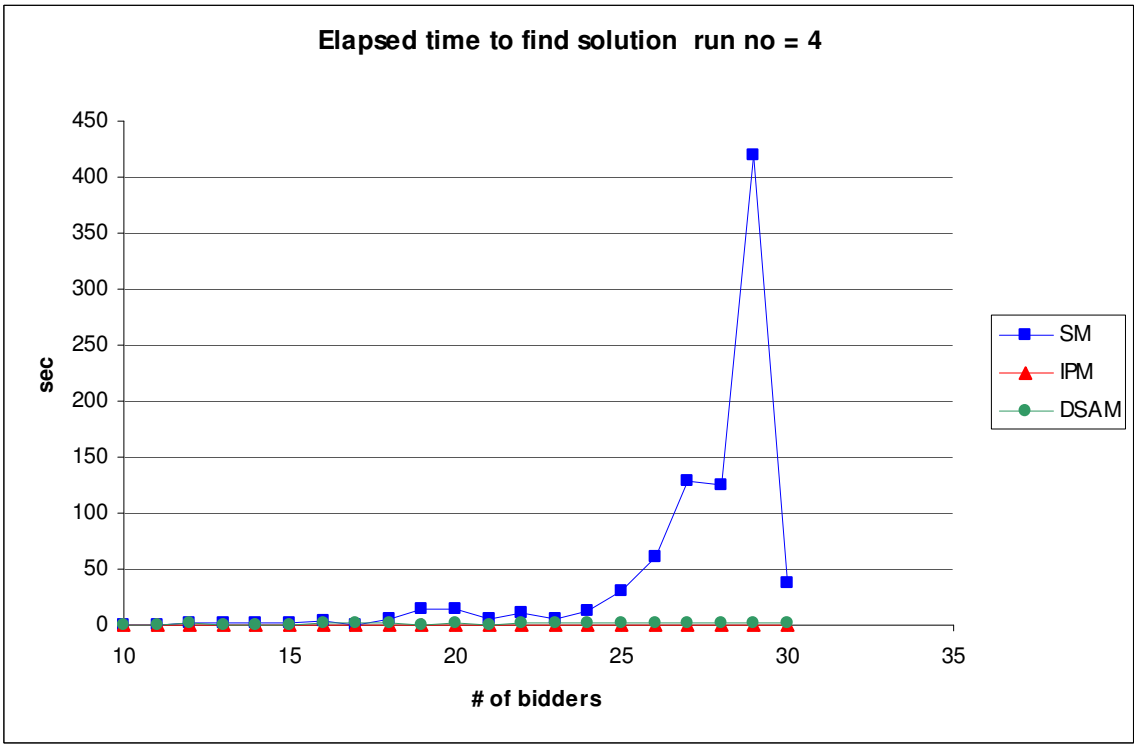




Simulation Run 4




Used bandwith % run no = 4
allocated bandwidth %
90
80
70
60
50
40
30
20
10
0
10
15
20
25
30
35
# of bidders
SM
IPM
DSAM


Elapsed time to find solution run no = 4
sec
450
400
350
300
250
200
150
100
50
0
10
15
20
25
30
35
# of bidders
SM
IPM
DSAM