ABSTRACT

ASHISH SUREKA. Techniques For Finding Nash Equilibria In Combinatorial Auctions. (Under the direction of Dr. Peter Wurman).

Auctions that allow participants to bid on a combination of items rather than just the individual items are called combinatorial auctions. For items that exhibit complementarity and substitutability, combinatorial auctions can be used to reach economically efficient allocations of goods and services. There has been a surge of recent research on combinatorial auctions because of the wide variety of practical situations to which they can be applied. There are several instances in which combinatorial auctions have already been applied to allocate scares resources, but there are still some challenging issues that need to be addressed before combinatorial auctions can be much more widely used in practice. Many different combinatorial auctions designs have been proposed by researchers and recently there has been a lot of work on studying the computational and strategic aspects of these auction designs. In this thesis, I analyze combinatorial auctions from a game theoretic perspective and propose techniques for determining pure strategy Nash equilibrium of combinatorial auctions. For a variety of reasons, combinatorial auctions pose serious computational challenges to compute Nash equilibria using current techniques. One problem is that the size of the strategy space in combinatorial auctions is very large and grows exponentially with the number of bidders and items. Another computational issue is that for combinatorial auctions it is computationally expensive to compute the payoffs of the players as a result of the joint actions. This makes it computationally expensive to determine the complete payoff matrix upfront and then determine Nash equilibrium. In this dissertation, we present techniques to overcome these problems. We present algorithms based on meta-heuristic search techniques, best response dynamics and linear programming to tackle these problems. We present empirical and theoretical results to support our claim that the algorithms perform well.

Techniques For Finding Nash Equilibria In Combinatorial Auctions

by

Ashish Sureka

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial satisfaction of the
requirements for the Degree of
Doctor of Philosophy

Department of Computer Science

Raleigh

2004

Approved By:

Dr. David Thuente	Dr. Michael Young
Dr. Peter Wurman Chair of Advisory Committee	Dr. Munindar Singh

Dedicated to my parents, Ashok Sureka and Kamla Sureka, and to my wife Pragya.

Biography

Ashish Sureka was born in a small village in the state of Rajasthan, India in May 1976. He grew up in Delhi and Pune where he got his Bachelors of Computer Engineering degree from Pune University in 1998. After working for a while in the area of software engineering he decided to go for an advanced degree and took admission in the PhD program in computer science at North Carolina State University. He got his M.S degree in May 2002 and has worked in a startup company in Cary and at IBM Research as an intern.

Acknowledgements

I take this opportunity to thank all those people who have provided me with their help, support and encouragement and without whose support this work would not have been possible. I would like to thank my advisor, Dr Peter Wurman for his direction, valuable advice and support. I am glad that I got an opportunity to work in his research group. I would also like to thank Dr. Munindar Singh, Dr. David Thuente and Dr. Michael Young for agreeing to be in my advisory committee and providing useful feedback and comments. Thanks to my parents and my wife for providing an unconditional love and support and always motivating me to do a PhD. I would like to extend my deepest gratitude to all the members of the Intelligent Commerce Research Group, Computer science department and North Carolina State University for providing a very encouraging environment. Special thanks to Gangshu, Jie and Tiejun for their comments and discussion on my research work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Game theory is the study of interactive decision making and was founded by the great mathematician John von Neumann. Game theory helps us to understand situations in which decision-makers interact and in which the outcome of the interaction depends on the interactive strategies of two or more persons who have opposed, or at best, mixed incentives. Those involved in the decisions are affected by their own choices as well as by the choices of others. Equilibrium (in this study, Nash equilibrium) is a stable outcome of the game in the sense that given that the other players adhere to their strategies, no single player wants to unilaterally deviate from its strategy. Any outcome that is reached by the play of strategies which do not form an equilibrium is an implausible way of playing the game because at least one player could improve by selecting another strategy. Nash equilibrium is the most widely used solution concept in game theory and is named after the Nobel Laureate John Nash. Game theory has been applied applied to a wide range of situations: firms competing for business, political candidates competing for votes, auctions, bankruptcy, arms races, cartel behavior, animals fighting over prey and more. Game theory can also be used to analyze the outcome and strategies of bidders competing in a type of auction called combinatorial auctions.

Auctions that allow participants to bid on a combination of items are called combinatorial auctions. For items that exhibit complementarity and substitutability, combinatorial auctions can be used to reach economically efficient allocations of goods and services. Recently, there has been a surge of research on combinatorial auctions, and many combinatorial auction designs have been proposed. In recent years, combinatorial auctions have gained a lot of interest because there are many situations in which combinatorial bidding seems natural, such as the series of FCC spectrum auctions [24, 4]. There are several instances in which combinatorial auctions have been applied to allocate scares resources, but there are some challenging issues that need to be addressed before combinatorial auctions can be much more widely used in practice. In order to analyze the effect of various combinatorial auction designs we need to study combinatorial auctions from a game theoretic perspective and determine equilibrium bidding strategies of bidders participating in the auction.

We model combinatorial auctions as multi-player, complete information games and assume that the bidders participating in the auction follow a myopic best response bidding strategy. Myopic best response bidding is a simple bidding strategy in which, in each round, the myopic bidder bids on the bundle that gives it the highest surplus as if this were the last round of the auction. Even in a restricted case of bidders following a myopic best response bidding strategy, the size of the strategy space is infinite and it is impossible to compute Nash equilibria by representing the game in normal form and populating the payoff matrix for each strategy profile. Thus, one of the problems that makes it hard to compute Nash equilibrium of combinatorial auctions is the very large size of the strategy space and the resultant normal form game. Another computational issue is that combinatorial auctions fall into a class of N-person games where it is computationally expensive to compute the payoffs of the players as a result of the joint actions. Previous algorithms to compute Nash equilibria are based on mathematical programming and analytical derivation and requires a complete payoff matrix before computing the Nash equilibria. However, determining a payoff matrix can itself be computationally intensive. In this dissertation, we present techniques to overcome these problems. We begin by first providing background information to the reader and then define the research problem. We then present a high level overview of our proposed solutions and describe each solution in detail in the later chapters of the dissertation.

1.1 Background

In order to understand the contributions of the dissertation, it is necessary to have some background in game theory. We provide a brief introduction to strategic form representation of a game, Nash equilibrium, combinatorial auctions, proxy bidding and best response dynamics.

1.1.1 Strategic form games

A strategic game is a model of interactive decision-making in which each decision-maker simultaneously chooses his plan of action from its strategy set and receives a payoff based on its utility function as well as the moves selected by the other players. A strategic form game is described by the list of players, the strategies available to each player, and the payoffs to any strategy combination, one strategy for each player. For example, the strategic form of the well known prisoner's dilemma game in Table 1.1 can be represented in a table in which the two rows correspond to the strategies of player 1 and the two columns correspond to the strategies of player 2. Each cell of the table represents the payoffs associated with that pair of strategies. The four possible strategy combinations in the game of prisoner's dilemma are (Don't Confess, Confess), (Don't Confess, Don't Confess), (Confess, Don't Confess) and (Confess, Confess). Similarly, the four possible strategy combinations in the game of matching pennies in Table 1.2 are (Heads, Heads), (Heads, Tails), (Tails, Heads), and (Tails, Tails). The payoffs are specified for each player for every one of the four strategy combinations. In the matching pennies game of Table 1.2, (1,-1) are the payoffs to the two players if the strategy combination (Heads, Heads) is played

Formally, a strategic game consists of:

- a finite set N, containing the set of players, $i \in N$.
- for each player $i \in N$, a nonempty set A_i (the set of actions available to player i). A_i is the list of strategies that player i can adopt
- for each player $i \in N$ a preference relation $\succeq_i$ on $A = \chi_{j \in N} A_j$ (the preference relation of player i). For each possible combination of strategies there is a list of payoffs for every player.

	Don't Confess	**Confess**
Don't Confess	3,3	0,4
Confess	4,0	**1,1**

Table 1.1: The game of prisoner's dilemma represented in strategic form.

	Heads	**Tails**
Heads	1,-1	-1,1
Tails	-1,1	1,-1

Table 1.2: The game of matching pennies represented in strategic form.

The high level of abstraction of the strategic form model allows it to be applied to a wide variety of situations. A player may be an individual human being or any other decision-making entity like a government, a board of directors, the leadership of a revolutionary movement, or even an animal [30]. For the purpose of our study we use the strategic form to represent games in which bidders (players) participate in combinatorial auctions and submit their bids (strategy or actions) simultaneously (or without the knowledge of other bidder's bid).

1.1.2 Nash equilibrium

Nash equilibrium is the most widely used solution concept in game theory and is named after the Nobel-prize winning mathematician John Nash. The notion of Nash equilibrium captures a steady state of the play of a strategic game in which each player holds the correct expectation about the other players' behavior and acts rationally [30]. If there is a set of strategies with the property that no player can benefit by changing her strategy while the other players keep their strategies unchanged, then that set of strategies and the corresponding payoffs constitute the Nash Equilibrium. In other words, we can say that Nash equilibrium of a game is a profile of strategies such that each player's strategy is an optimal response to the other players' strategies.

Formally, a Nash equilibrium of a strategic game $[N, (A_i), (\succeq i)]$, is a profile $a^* \in A$ of actions with the property that for every player :

$$(a^*_{-i}, a^*_i) \ \succeq i \ (a^*_{-i}, a_i) \ \ for \ all \ \ a_i \in A_i. \tag{1.1}$$

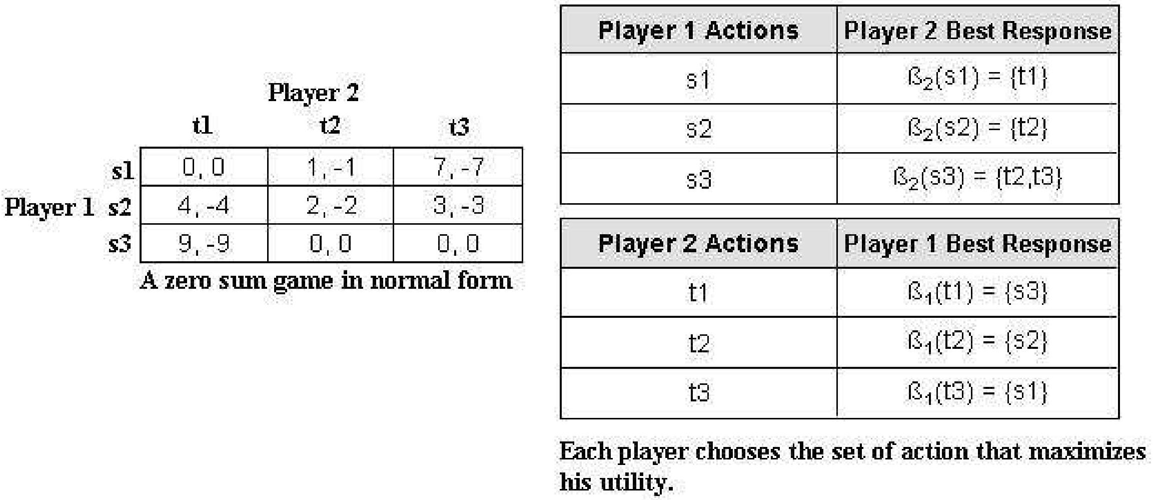
For each player there is a nonempty set A_i (the set of actions available to player i). Thus for a^* to be a Nash equilibrium it must be that no player i has an action yielding an outcome that he prefers to that generated when he chooses a^*_i , given that every other player j chooses his equilibrium action a^*_j.

Let's apply the definition of Nash equilibrium to the game of prisoner's dilemma shown in the Table 1.1. We can see that the strategy pair (Don't Confess, Don't Confess) for each player is not a Nash-equilibrium. From (Don't Confess, Don't Confess), each player can benefit by chosing the action of Confess, if the other player keeps her strategy unchanged. Applying the same logic we can see that the strategy pair (Confess, Don't Confess) and (Don't Confess and Confess) are also not Nash equilibrium. We can eliminate any strategy pair except the bottom right (Confess, Confess), at which both players get a payoff 1. The joint action (Confess, Confess) is the unique Nash equilibrium in the game.

Another example shown in Table 1.2 is the game of matching pennies which is classified as a zero-sum game with two players. A zero-sum game is a game in which players make payments only to each other. One player's loss is the other player's gain, so the total amount of money available remains constant. Each player shows either heads or tails from a coin. If both are heads or both are tails then player 1 wins, otherwise the player 2 wins. The payoff matrix is shown in the Table 1.2. We see that there is no Nash equilibrium to this game.

1.1.3 Best Response Functions

In small games where each player has only a few actions and the payoff matrix is small, we can sometimes find the Nash equilibria by examining each action profile in turn to see if it satisfies the conditions of equilibrium. But when the games become more complicated and when the strategy space is large or infinite, we can sometimes find the Nash equilibria using the best response functions of each player. Formally, a strategy, a, of player, i, is called a best response to the other player's strategy if a has greater payoff to i than any of $i's$ other strategies given the other player's strategy. In the following section we illustrate how Nash equilibria can be found by working with the best response function of the players.



		Player 2		
		t1	t2	t3
	s1	0, 0	1, -1	7, -7
Player 1	s2	4, -4	2, -2	3, -3
	s3	9, -9	0, 0	0, 0

A zero sum game in normal form

Player 1 Actions	Player 2 Best Response
s1	ß$_2$(s1) = {t1}
s2	ß$_2$(s2) = {t2}
s3	ß$_2$(s3) = {t2,t3}

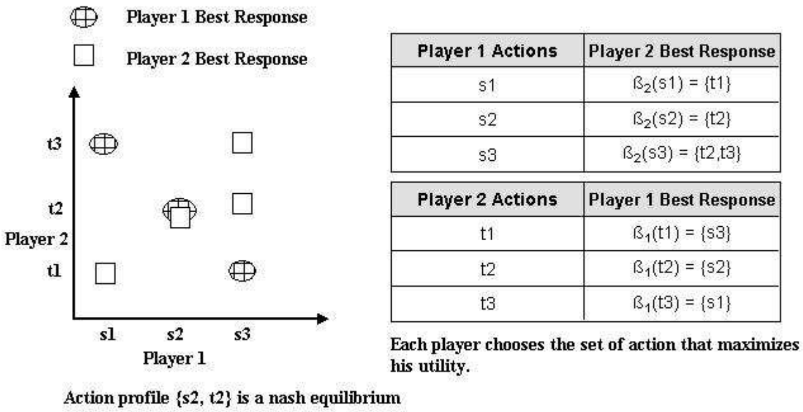
Player 2 Actions	Player 1 Best Response
t1	ß$_1$(t1) = {s3}
t2	ß$_1$(t2) = {s2}
t3	ß$_1$(t3) = {s1}

Each player chooses the set of action that maximizes his utility.

Figure 1.1: Figure showing a two player game in strategic form. The figure has two tables that lists each player's best response function by finding the action that maximizes its payoff for any given action of the other player.

Consider the strategic game in Figure 1.1 in which there are two players. The set of actions for player 1 is $s1, s2$ and $s3$ and the set of actions for player 2 is $t1, t2$ and $t3$. We are interested in finding the best response or best actions for each player. We determine the best actions of each player from the payoff matrix of the game. Figure 1.1 lists each player's best response function by finding the action that maximizes its payoff for any given action of the other player. We denote the best response function of player i for an action a of another player by $\beta_i(a)$. For example the action of player 1 that maximizes her payoff, given that player 2's action is $t1$, is called player 1's best response to $t1$ and is denoted by $\beta_1(t1)$. The value of $\beta_1(t1)$ in our example is $s3$. It is not necessary that each player has a single best response. In our example in figure 1.1 the best response of player 2 given that player 1's action is $s3$, is denoted by $\beta_2(s3)$ and its value is the set $\{t2, t3\}$. Both $t2$ and $t3$ are best actions for player 2 if player 1 chooses $s3$ because both yield a payoff of 0, and player 2 has no other action that yields a higher payoff. The best response function is set-valued, that is, it associates a set of actions with any list of the other player's actions.

Once we have the best response functions for each player we determine the Nash equilibria by plotting the best response functions and looking at their intersection. In Figure 1.2 we use a graphical approach to find the pair (a_1, a_2) of actions with the property that



Player 1 Actions	Player 2 Best Response
s1	β_2(s1) = {t1}
s2	β_2(s2) = {t2}
s3	β_2(s3) = {t2,t3}

Player 2 Actions	Player 1 Best Response
t1	β_1(t1) = {s3}
t2	β_1(t2) = {s2}
t3	β_1(t3) = {s1}

Each player chooses the set of action that maximizes his utility.

Figure 1.2: Finding Nash equilibria by plotting the best response functions and looking at its intersection. Player 1's best action is denoted by a oval and player 2's best action is denoted by a square. The game has a unique Nash equilibrium denoted by the action profile $s2, t2$

.

player 1's action is a best response to player 2's action, and player 2's action is a best response to player 1's action: $a_1 = \beta_1(a_2)$ and $a_2 = \beta_2(a_1)$. Player 1's actions $s1, s2$ and $s3$ are plotted on the horizontal axis and player 2's actions $t1, t2$ and $t3$ are plotted on the vertical axis. In Figure 1.2, player 1's best action is denoted by a oval that associates the best action of player 1 with every action of player 2. Similarly, player 2's best action is denoted by a square and is plotted on the graph. Now we find points on the graph where we have square as well as oval. Each such point is a Nash equilibrium. We conclude that the game has a unique Nash equilibrium denoted by the action profile $\{s2, t2\}$.

We can restate the definition of Nash equilibrium in terms of the best response function. For any $a_{-i} \in A_i$ we define $\beta_i(a_{-i})$ to be the set of player $i's$ best actions given a_{-i}:

$$\beta_i(a_i) \ = \ \{a_i \in A_i : \ (a_{-i}, a_i) \ \succeq i \ (a_{-i}, a_i^{'}) \ for\ all\ a_i^{'} \in A_i\} \tag{1.2}$$

A Nash equilibrium is a profile a^* of actions for which

$$a_i^* \in \beta_i(a_{-i}^*) \ for\ all\ i \in N. \tag{1.3}$$

This alternative formulation of the definition points us to a another method of

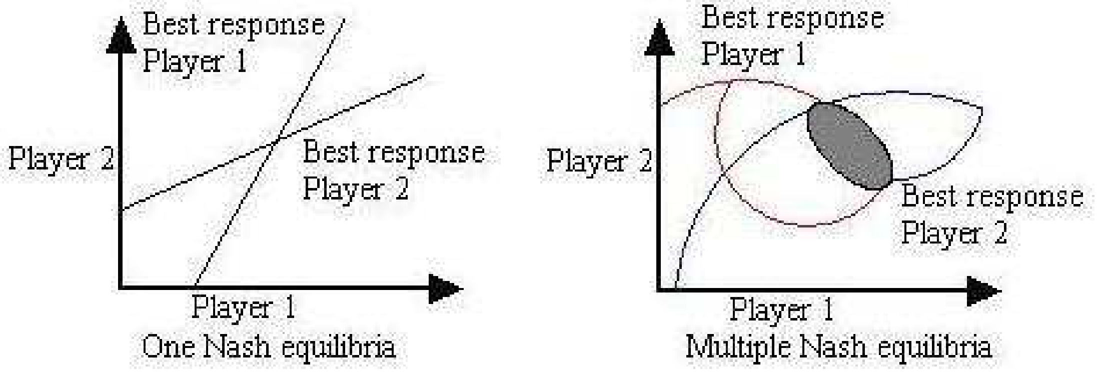
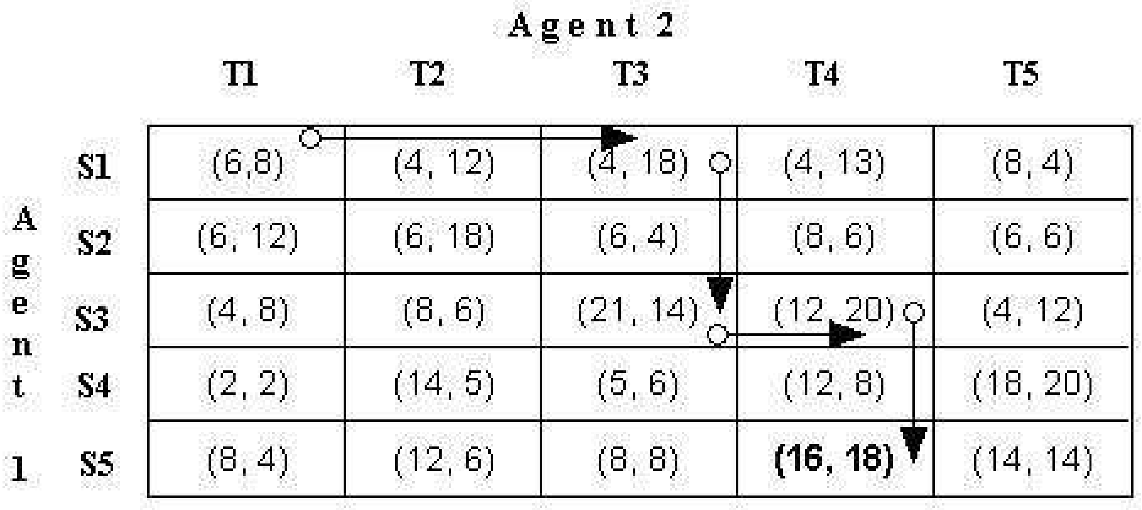


Figure 1.3: The best response functions of a two-player game where each player has infinitely many actions.

finding Nash equilibria: first calculate the best response function of each player, then find a profile a^* of actions for which $a_i^* \in \beta_i(a_{-i}^*)\ for\ all\ i \in N$.

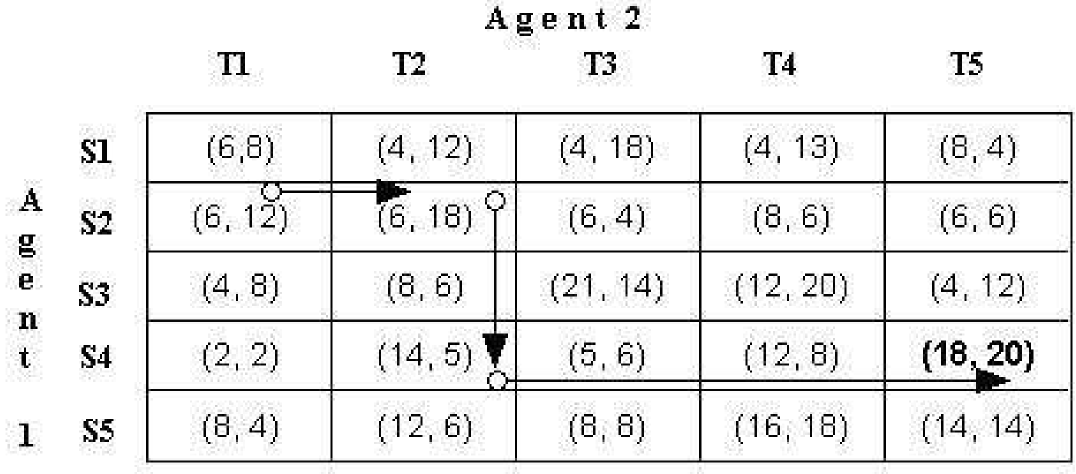
A strategic game may have no pure strategy Nash equilibrium, may have a single Nash equilibrium, or may have many Nash equilibria. Moreover, if the strategy space for each player is continuous, then each player has infinitely many actions and we cannot represent the game in a table like those used in Figure 1.1. In this case we construct and analyze the best response functions that can be represented by mathematical equations. In Figure 1.3, we plot each player's best response function and find the intersection points of the two player's best response function. Figure 1.3 shows the best response functions of a two-player game where each player has infinitely many actions. The first diagram on the left plots the best response functions of a two player game where the best response function for both the player is a straight line. In the game in this example, each player has a unique best response to every action of the other player, so that the best response functions are lines. The game has a unique Nash equilibria and is a point where both the straight lines intersect. The diagram on the right in Figure 1.3 is an example of the best response functions of a two-player game in which, the players have many best responses to some of the other player's actions. Thus her best response function is thick at some points. The action profiles corresponding to the intersection point of the players best response function are the multiple Nash equilibrium of the game. There are multiple Nash equilibria in this example because the best response functions cross each other more than once.



Agent 2

Agent 1	T1	T2	T3	T4	T5
S1	(6,8)	(4, 12)	(4, 18)	(4, 13)	(8, 4)
S2	(6, 12)	(6, 18)	(6, 4)	(8, 6)	(6, 6)
S3	(4, 8)	(8, 6)	(21, 14)	(12, 20)	(4, 12)
S4	(2, 2)	(14, 5)	(5, 6)	(12, 8)	(18, 20)
S5	(8, 4)	(12, 6)	(8, 8)	**(16, 18)**	(14, 14)

Figure 1.4: An example where applying the process of best response dynamics converges to a Nash equilibrium from the initial action profile (S1,T1).



Agent 2

Agent 1	T1	T2	T3	T4	T5
S1	(6,8)	(4, 12)	(4, 18)	(4, 13)	(8, 4)
S2	(6, 12)	(6, 18)	(6, 4)	(8, 6)	(6, 6)
S3	(4, 8)	(8, 6)	(21, 14)	(12, 20)	(4, 12)
S4	(2, 2)	(14, 5)	(5, 6)	(12, 8)	**(18, 20)**
S5	(8, 4)	(12, 6)	(8, 8)	(16, 18)	(14, 14)

Figure 1.5: Another example where applying the process of best response dynamics converges to a Nash equilibrium. The intial action profile and the Nash equilibrium are different from that of the previous example in Figure 1.4

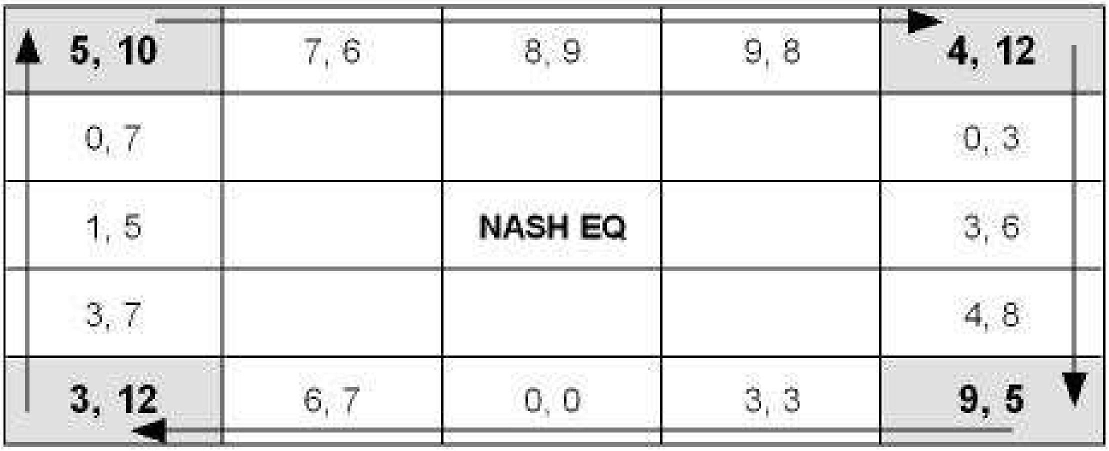
1.1.4 Best Response Dynamics

Best-response dynamics is an iterative process of determining a Nash equilibrium where each bidder iteratively select the best response to their opponent's best strategies. An example of how the process of iteratively computing the best response functions converges to a Nash equilibrium is depicted in Figure 1.4. Figure 1.4 shows a game in normal form where each player's action set contains 5 actions. Assume the search starts from the initial action profile $(S1, T1)$. The payoff to the players for this action profile is $(6, 8)$. The action profile $(S1, T1)$ is clearly not a Nash equilibrium as each player can benefit by choosing another action, if the other player keeps her strategy unchanged. Let us say it is player 2's turn to first find the best response. Player 2 finds $T3$ as the best response to player 1's action of $S1$. Now it is player 1's turn to find the best response to player 2's action of $T3$. This process of finding the best responses continues between the two players until a point in the space of outcomes is reached where both player's actions form best responses to one another. The point where the steady state is reached is the Nash equilibrium. For the example in Figure 1.4, the action profile $(S5, T4)$ is the Nash equilibrium reached as a process of applying the best response dynamics starting from the action profile $(S1, T1)$ with player 1's turn to first find the best response. A game can have multiple Nash equilibria and the Nash equilibrim reached by applying the process of best response dynamics depends on the starting or initial action profile . It also depends on the order in which players find the best responses. Figure 1.4 and Figure 1.5 refer to the same game but the Nash equilibria reached by applying the best response dynamics are different because of the starting point (cell of the payoff matrix).

It is important to note that the best response dynamics may not converge to a Nash equilibrium even if one exists. Figure 1.6 is an example of a normal-form game where applying the process of best response dynamics does not converges to a Nash equilibrium because of getting stuck in an endless cycle.

1.1.5 Combinatorial Auctions

Auctions involve trading of variety of different items, provide a useful mechanism for resource allocation problems and enable dynamic pricing. Various types of single and multi-unit auctions such as English, Dutch, Vickrey and First-Price Sealed-Bid auctions



5, 10	7, 6	8, 9	9, 8	**4, 12**
0, 7				0, 3
1, 5		**NASH EQ**		3, 6
3, 7				4, 8
3, 12	6, 7	0, 0	3, 3	**9, 5**

Figure 1.6: An example where applying the process of best response dynamics does not converges to a Nash equilibrium because of getting stuck in an endless cycle.

have been used as a trading mechanism for allocation of resources [3]. These auctions which involve selling of a single or multiple unit of a single type of item at a time have a limited capability that bidders cannot bid on combination or groups of items which may lead to inefficient outcomes because of the exposure problem [38] wherein a bidder is exposed to the risk that it may acquire unwanted items or may fail to acquire items for which it has the highest value. Many trading scenarios and problems like allocation of airport time slots [34], delivery routes, distributed scheduling, task assignment, allocation of segments of radio frequency spectrum and sale of furniture [5] involve agents that have non-additive values (valuation of a particular bundle of items may not equal to the sum of valuations of the individual items) for the resources being traded. For example the value of a property to a bidder is increased if another property or group of properties is won by that bidder because of a super-additive or synergistic effect [19]. Auctions that allow participants to bid on combination of items are called combinatorial auctions (CA). For items that exhibit complementarity and substitutability, combinatorial auctions can be used to reach an economically efficient allocation of goods and services and eliminate the exposure problem.

The gold standard for combinatorial auctions is the Generalized Vickrey Auction (GVA) [44]. Generalized Vickrey Auction is a sealed-bid combinatorial auction in which truthful bidding is the optimal strategy for the bidder. The Generalized Vickrey Auction has received wide attention in the literature and is considered to be a benchmark against which other combinatorial auctions are compared. Since the GVA is a single-shot auction, every bidder reports her valuation to the auctioneer who then computes final allocation and prices. The GVA is an economically efficient mechanism because it finds the socially optimal allocation by selecting the combination of bids that results in the maximum value. Computing an allocation that maximizes total value is called the combinatorial allocation problem (CAP). The GVA is incentive compatible as no bidder can unilaterally improve its outcome by deviating from reporting its true value. One of the drawbacks of GVA is that it requires the bidders to compute and reveal their values for all combinations of items. This drawback of the GVA is considered to be a fundamental problem as the agents have limited computation resources and are considered to be bounded rational. When there are large number of items and agents it is computationally demanding for every agent to reveal its true value for all possible combination of items. For this reason a variety of iterative combinatorial auctions have been suggested by the research community. The iterative combinatorial auctions does not require the agents to have complete information about their valuations up-front and can elicit information from the agents dynamically during the course of the auction. There has been a surge of recent research on combinatorial auctions and many iterative combinatorial auctions have been proposed including Ascending k-Bundle Auction [46, 47], Ascending Package Auction [2] and iBundle [31, 32]. These proposed iterative or progressive auctions differs from each other in the way by which the allocations and prices are computed and the type of information revealed to the bidders after each round of the auction. We have chosen these three auctions for our study.

1.1.6 Proxy Bidding

Proxy bidding is an automatic bidding process and is common in online auction sites like eBay. The proxy bidding system is very common in online auctions as it increases performance by reducing the amount of time spent by the bidders in participating in the auction. As shown in Figure 1.7, the proxy bidder sits between the bidder and the auction. Each bidder decides the maximum it is willing to pay for the item and conveys this amount to the proxy bidder. The proxy bidder then bids on behalf of the bidders. The proxy bidder will automatically place bids for the bidders up to the highest price the bidders have specified. The proxy bidder uses only as much of the maximum bid as is necessary to maintain the bidders position as a high bidder. The advantage of proxy bidding is that the bidder does not have to watch every minute of the auction. Once the auction is over, the bidders pay only the lowest possible winning bid, which may be less than the value they gave to their agent. The proxy bids are also kept confidential and are not disclosed to the seller or other bidders participating in the auction.

For example in Figure 1.7, bidder 1 specifies [(A 20), (B 30), (AB 60)] as the maximum amount that the proxy bidder should offer. The proxy bidding process is implemented in standard auctions at online sites like eBay, and the same concept can be extended to combinatorial auctions. We have implemented a proxy bidding system that can be tied to various combinatorial auctions. The proxy agent increases the bids for each bundle by the minimum bid increment only when the bidder has been outbid by another bidder.

1.1.7 Myopic bidding

In our study we assume bidders interact with the auction through a proxy agent and the proxy agent bids myopically. Myopic bidding is also called straightforward bidding. In each round, the myopic bidder bids on the bundle that gives it the highest surplus as if this were the last round of the auction. In iterative combinatorial auctions, at the end of each round the auctioneer announces the winning bids and the prices for each bundle. A myopic bidder behaves as if it can win any bundle that it is currently not winning by bidding δ more than the announced price where δ is the minimum bid increment. Let τ_b denote the price for bundle b. Let $\breve{b}$ denote agent $i's$ tentative allocation announced by the auctioneer in round t. The myopic agent's strategy is to bid on the bundle $b^{'}$ that maximizes its surplus at the given prices. The bundle, $b^{'}$, to bid on in the round $t+1$ is determined according to the following equation:

$$b^{'} = arg\ max_b \begin{cases} v_i(\breve{b}) \ - \ \tau_{\breve{b}} & if\ b = \breve{b}, \\ v_i(b) \ - \ (\tau_b \ + \ \delta) & otherwise \end{cases} \tag{1.4}$$

If the above solution gives strictly more surplus that the agent's current allocation, the agent will increase its bid on $b^{'}$ to $\tau_{b^{'}} \ + \ \delta$.

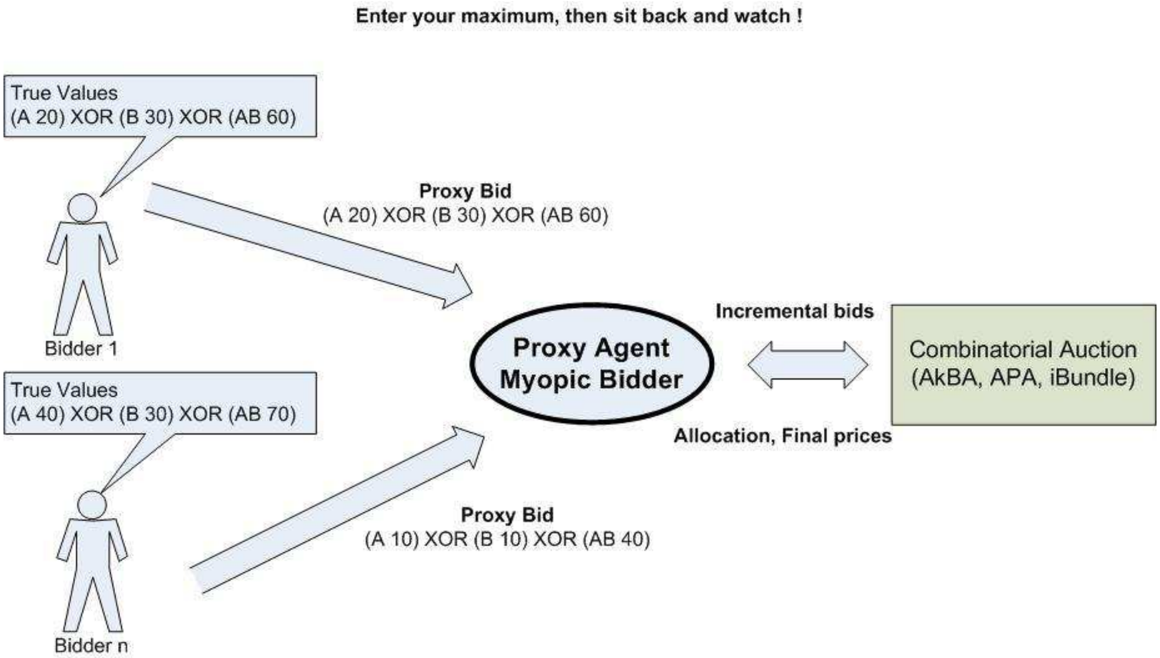

Enter your maximum, then sit back and watch !
True Values
(A 20) XOR (B 30) XOR (AB 60)
Bidder 1
Proxy Bid
(A 20) XOR (B 30) XOR (AB 60)
True Values
(A 40) XOR (B 30) XOR (AB 70)
Bidder n
Proxy Bid
(A 10) XOR (B 10) XOR (AB 40)
Proxy Agent
Myopic Bidder
Incremental bids
Allocation, Final prices
Combinatorial Auction
(AkBA, APA, iBundle)

Figure 1.7: Figure showing the interaction of the bidders with the combinatorial auction system through proxy agents.

1.1.8 XOR Bidding Language

In our system the proxy bid is expressed in terms of logical connectives. We use the exclusive-or (XOR) bidding language in which the bid $[(\tau_1, b_1)XOR(\tau_2, b_2)XOR...XOR(\tau_k, b_k)]$ has the meaning that the bidder will buy at most of the bundles $(b_1, b_2..b_k)$ at the stated price of $(\tau_1, \tau_2..\tau_k)$ for each other bundles where τ_i represents the maximum amount the bidder is willing to pay for the bundle b_i.

1.2 Combinatorial Auctions as Normal Form Games

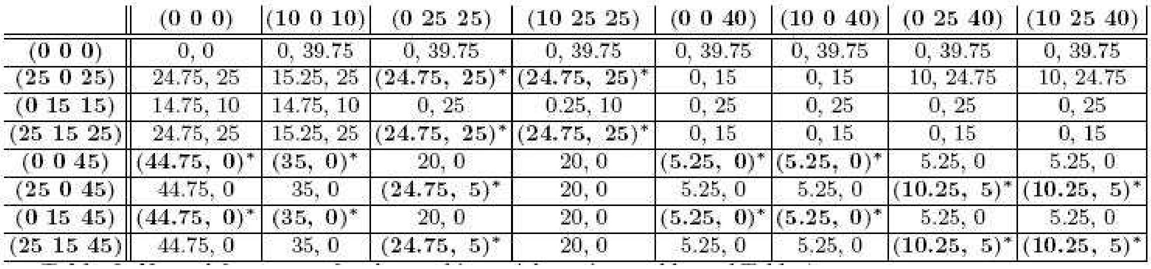
In our model, we assume that agents follow a myopic best response bidding strategy. The bid vector representing an agents maximum willingness to pay for a bundle constitutes a strategy as long as the bid vector satisfies the condition of free disposal. Free disposal is a standard assumption which means the value of a subset of a bundle is less than or equal to the value of the bundle. It is natural to expect the bidders in the auction act strategically and it is not necessary that agents reveal their true valuations for the bundles.

	A	**B**	**AB**
Agent 1	25	15	45
Agent 2	10	25	40

Table 1.3: Valuations of agents for the bundles for a two-agent two-item combinatorial auction problem

The agents have an incentive to misrepresent their values to obtain a higher surplus. We also assume that each bidder (decision maker) has perfect knowledge of the game and of his opposition; that is, he knows in full detail the rules of the game as well as the payoffs of all other bidders for all combination of joint actions. Table 1.3 shows a combinatorial auction problem with two agents and two items where agent 1's and agent 2's valuations on bundles exhibit complementarity.

Figure 1.8 is a normal form representation of the combinatorial auction in Table 1.3 with a restricted strategy space. The payoffs for each agent is determined by running the Ascending k-bundle auction [46, 47] for a restricted set of strategy profiles and using a bid increment of 0.25. Strategy profiles that are Nash Equilibrium are represented in parenthesis and bold letters. Agent 1 is the row player and Agent 2 is the column player.

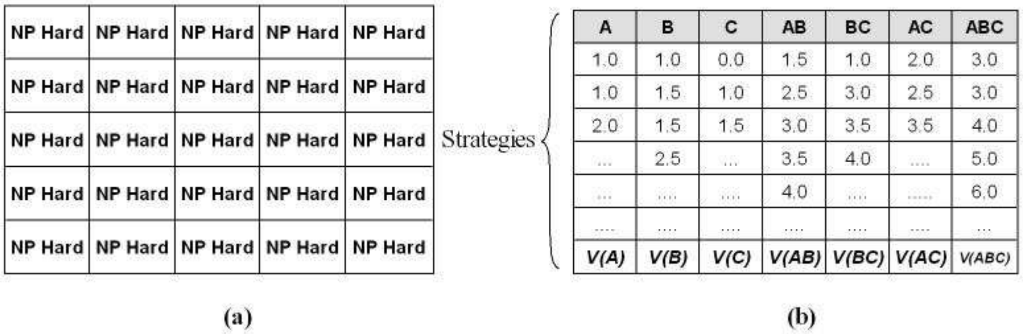


	(0 0 0)	(10 0 10)	(0 25 25)	(10 25 25)	(0 0 40)	(10 0 40)	(0 25 40)	(10 25 40)
(0 0 0)	0, 0	0, 39.75	0, 39.75	0, 39.75	0, 39.75	0, 39.75	0, 39.75	0, 39.75
(25 0 25)	24.75, 25	15.25, 25	**(24.75, 25)***	**(24.75, 25)***	0, 15	0, 15	10, 24.75	10, 24.75
(0 15 15)	14.75, 10	14.75, 10	0, 25	0.25, 10	0, 25	0, 25	0, 25	0, 25
(25 15 25)	24.75, 25	15.25, 25	**(24.75, 25)***	**(24.75, 25)***	0, 15	0, 15	0, 15	0, 15
(0 0 45)	**(44.75, 0)***	**(35, 0)***	20, 0	20, 0	**(5.25, 0)***	**(5.25, 0)***	5.25, 0	5.25, 0
(25 0 45)	44.75, 0	35, 0	**(24.75, 5)***	20, 0	5.25, 0	5.25, 0	**(10.25, 5)***	**(10.25, 5)***
(0 15 45)	**(44.75, 0)***	**(35, 0)***	20, 0	20, 0	**(5.25, 0)***	**(5.25, 0)***	5.25, 0	5.25, 0
(25 15 45)	44.75, 0	35, 0	**(24.75, 5)***	20, 0	5.25, 0	5.25, 0	**(10.25, 5)***	**(10.25, 5)***

Figure 1.8: Normal form game for the combinatorial auction problem of Table 1.3

1.3 Research problem and solution approaches

Figure 1.9 illustrates two challenges that makes it hard to compute the Nash equilibria of combinatorial auctions. As shown in panel (a) of Figure 1.9, one of the challenging computational issue making it hard to compute Nash equilibrium is that combinatorial auctions fall into a class of N-person games where it is computationally expensive to compute


Strategies

NP Hard	NP Hard	NP Hard	NP Hard	NP Hard
NP Hard	NP Hard	NP Hard	NP Hard	NP Hard
NP Hard	NP Hard	NP Hard	NP Hard	NP Hard
NP Hard	NP Hard	NP Hard	NP Hard	NP Hard
NP Hard	NP Hard	NP Hard	NP Hard	NP Hard

(a)

A	B	C	AB	BC	AC	ABC
1.0	1.0	0.0	1.5	1.0	2.0	3.0
1.0	1.5	1.0	2.5	3.0	2.5	3.0
2.0	1.5	1.5	3.0	3.5	3.5	4.0
...	2.5	...	3.5	4.0	....	5.0
...	....	....	4.0	....	.....	6.0
....	....	....	....	....	....	...
V(A)	*V(B)*	*V(C)*	*V(AB)*	*V(BC)*	*V(AC)*	*V(ABC)*

(b)

Figure 1.9: Panel (a): Figure illustrating a normal from game where NP-hard optimization problems are required to be solved to determine the values in every cell. Panel (b): Figure illustrating the very large size of strategy space of a bidder participating in a combinatorial auction.

the payoffs of the players as a result of the joint actions. As shown in panel (b) of Figure 1.9, another problem that makes it hard to compute Nash equilibrium of combinatorial auctions is the very large size of the strategy space and the resultant normal form game. The size of the strategy space is of the order of k^{2^n} where n denotes the number of items and k denotes the number of discrete bid values allowed for each bundle.

The research question that we address in this dissertation is: ***How do we efficiently find Nash equilibrium for games of very large size and for which it is computationally expensive to compute the payoffs ?***

The study approaches the problem from four different angles. The ultimate goal is to reduce the amount of time required to compute Nash equilibria in combinatorial auctions. The four different solution approaches I study to addresses the research problem are:

Approach 1 To compute pure strategy Nash equilibria of a game without computing the whole pay-off matrix. This reduces the amount of time required to compute the payoff matrix thereby reducing the total amount of time required to compute the Nash equilibrium

Approach 2 To determine an approximate Nash equilibrium.

Approach 3 To take advantage of the structure and pattern of the payoff matrix to identify Nash equilibria without actually running the auction and creating a payoff matrix.

Approach 4 To reduce the amount of time required to determine the payoffs for a joint action thereby reducing the total amount of time required to compute Nash equilibria.

Following is an outline of the dissertation. This is Chapter 1 and each of the remaining five chapters are devoted to one of the four solution approaches and a list of dissertation contributions. In Chapter 2, we present a technique based on best-response dynamics and tabu search to compute pure-strategy Nash equilibria (Approach 1). In Chapter 3, we present an application of metaheuristic techniques like genetic algorithms and tabu search to explore the space of bidding strategies in order to find the best response strategy (Approach 2). In Chapter 4, we present a geometric approach for determining Nash equilibria of combinatorial auctions (Approach 3). Finally in Chapter 5, we present a linear programming approach to directly compute the outcome of a specific type of auction (Approach 4). A high level overview of my dissertation contributions are presented in Chapter 6.

Chapter 2

Tabu Search for Finding Pure Strategy Nash Equilibria in Very Large Normal Form Games

In this chapter, we present a new method for computing pure strategy Nash equilibria for a class of N-person games where it is computationally expensive to compute the payoffs of the players from the joint actions. Previous algorithms to compute Nash equilibria are based on mathematical programming and analytical derivation and require a complete payoff matrix before computing the Nash equilibria. However, determining a payoff matrix can itself be computationally intensive. One example of the problem arises from combinatorial auctions which requires solving many hard optimization problems to determine the payoffs of the bidder. Many other real world market scenarios and multi-agent decision making situations fall into the category. This chapter proposes an approach, based on best response dynamics and tabu search, that resolves the constraint of having a complete payoff matrix upfront, and instead computes the payoffs only when it is required. The proposed method can find pure strategy Nash equilibria in a multi-player by computing the values of the cells of a payoff matrix at runtime. We test the algorithm on several classes of standard and random games, and present empirical results that show the algorithm performs well. We also present empirical results of the runtime behavior of the algorithm on varying the distribution of the game. We present a list of performance indicators for evaluating the performance and efficiency of this type of algorithm.

2.1 Introduction

Computing Nash equilibria of a game is a hard problem and several algorithms have been proposed over the years to solve this problem [22]. The current state-of-the-art algorithms for computing Nash equilibria are the Lemke-Howson algorithm [20] for two-player games, the Govindan-Wilson algorithm [16] and an algorithm based on simplical subdivision [17] for n-player finite games. Several other algorithms for solving finite games are implemented in Gambit [23], which is a library of game theory software and tools for the construction and analysis of finite extensive and normal form games. The appropriate algorithm for computing the Nash equilibria for a game depends on a number of factors, such as, whether you want to find pure strategy equilibria or mixed strategy equilibria, or whether you want to find just one equilibrium or find all the equilibria.

The underlying assumption in current algorithms is the availability of a complete payoff matrix. The first task that a user who wants to compute Nash equilibria will do is to describe a game in normal or extensive form and store it in a text file in a prescribed format for the algorithm to operate on. However, determining a payoff matrix can itself be computationally expensive. For example , if it requires an hour to determine the values of each cell of the payoff matrix in a normal form game, then it can take many days to just enter the entire game matrix for a 2 player game in which each player has 10 actions. Despite the small size of the game, determining Nash equilibria becomes time consuming. This happens because the cost of computing the payoffs dominates the cost of computing the Nash equilibria. Another difficulty arises when there are large number of players and actions. In this situation, the file that serves as an input data to the algorithm for computing Nash equilibria can be very large, easily of the order of mega-bytes. For example, in a game with 6 players each with 10 actions to chose from, there are more than one million entries in the normal form game. Computing the payoffs and storing this type of game as a text file can be very time consuming and can consume many mega-bytes of disk space.

One example of the problem arises from combinatorial auctions [2, 5, 32, 38, 47]. Combinatorial auctions can be represented as noncooperative games in which bidders act strategically. We can model combinatorial auctions as a multi-player game where the bids submitted by the bidders represent the actions and the surplus to the bidders at the end of the auction represents the payoffs. In order to determine the payoffs to each bidders, we need to run the auction, thus determining the outcome of Combinatorial Auctions requires solving many hard optimization problems. It is a well recognized issue that in general combinatorial auctions are NP-hard to clear [5, 42, 8]. For combinatorial auctions computing the payoff for each cell in the normal form game is NP-hard and the size of the normal form game grows exponentially with the number of bidders and items. Even for a small and moderate size combinatorial auction problem, it is not possible to determine Nash equilibria using current tools. In order to relieve the burden of first running the auction for every strategy combination and then finding Nash equilibria, we propose a novel technique based on tabu search and best response dynamics that computes the payoffs only when it is required.

Many other real world market scenarios and multi-agent decision making situations are areas to which the current work is applicable. One such example is the Trading Agent Competition (TAC) [45]. TAC is a test-bed developed by researchers at University of Michigan to develop and experiment with various protocols and strategies of multi-agent systems. TAC describes two games, one a supply-chain management scenario and the other an E-Commerce trading scenario. In both type of games several software agents compete against each other to maximize their individual utility. Each TAC game takes around 15-55 minutes (depending on the type of TAC game) to run and report the payoffs of each participating agent at the end of the game. When viewed as a normal form game, it can be quite time consuming to compute the game matrix for a small to moderate size game even though computing the pure strategy Nash equilibria can be trivial task.

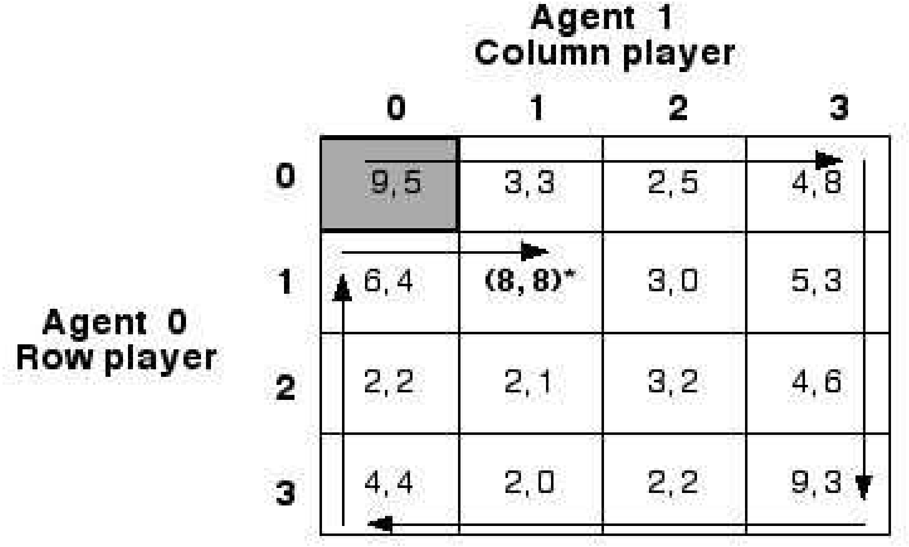
This work is an effort to develop solution methods for finding the pure strategy Nash equilibria where methods based on mathematical programming and analytical derivations cannot be used. Any algorithm that does not require to have a complete payoff matrix for computing the pure strategy Nash equilibria can be useful for these types of applications. It is important to note that the proposed algorithm should not be viewed as another method for computing Nash equilibria of a general finite N-person game. The current work is a step in a different direction and is advantageous for games in which it is computationally expensive to compute the complete payoff matrix upfront. Our main result is that for a class of games where it computationally expensive to compute the payoffs, vast computational savings can be realized by determining the payoffs only when it is required.

Following is an outline for the rest of the chapter. Section 2.2 gives an overview of tabu search. Best response dynamics and tabu search are the two main ingredients of the proposed algorithm. In section 2.3, we describe the underlying concepts behind the algorithm and present a formal description of the algorithm. Section 2.4 is devoted to experimental setup and results. Finally, section 2.5 are the conclusions.

2.2 Tabu Search

Tabu search has been widely used to solve approximately complex combinatorial optimization problems encountered in a variety of real-life applications. Tabu search was first presented by Glover [11, 12] and additional efforts of formalization are reported by Hansen [18] and de Werra & Hertz [6]. It is a local (neighborhood) search technique that can escape local optima. It makes use of adaptive memory to keep track of information obtained in the previous part of the run. Each time a move is made, it is placed on a list called the tabu-list. When considering a next move, it is deemed unchoosable, or tabu, if it is on the tabu-list [11, 12]. Old moves are typically removed from the tabu-list after some number of iterations. Other methods that have been applied to solve a wide variety of optimization problems are simulated annealing and evolutionary approaches using genetic algorithms. Unlike tabu search, both simulated annealing and genetic algorithms are memory-less methods in the sense that they do not make use of memory to record information related to solutions visited during the search process.

There are two types of adaptive memory used in tabu search: explicit memory and attributive memory. Explicit memory records complete solutions visited during the search. Explicit memory records exact solutions and is used to guide the search to avoid visiting solutions more than once. Since the complete solution is recorded, explicit memory structure may have excessive memory requirements depending on the memory required to store a solution and the number of solutions required to be stored in the tabu list. The total memory requirement increases with the increase in the length of the tabu list and the increase in the size of the solution. The amount of memory required by explicit memory version of the algorithm can be limited by setting an appropriate value of the length of the



Agent 0 Row player \ Agent 1 Column player	0	1	2	3
0	9,5	3,3	2,5	4,8
1	6,4	(8,8)*	3,0	5,3
2	2,2	2,1	3,2	4,6
3	4,4	2,0	2,2	9,3

Figure 2.1: An example of best response dynamics and tabu search showing the use of adaptive memory in preventing cycles during the search process.

tabu-list. A good value of the length of tabu-list depends on the nature of the problem that we are trying to solve and can be determined empirically.

Another form of adaptive memory that tabu search makes use of is called attributive memory. Attribute-based memory, like explicit memory is used to avoid visiting solutions more than once. Rather than recording the exact solutions, attribute-based memory records information about solution attributes that change in moving from one solution to another. For example, in a graph or network setting, attributes can consists of nodes or arcs that are added, dropped or repositioned by the moving mechanism. In production scheduling, the index of jobs may be used as attributes to inhibit the method from following certain search directions. In our experiments we test both explicit memory and attributive memory and compare the results obtained.

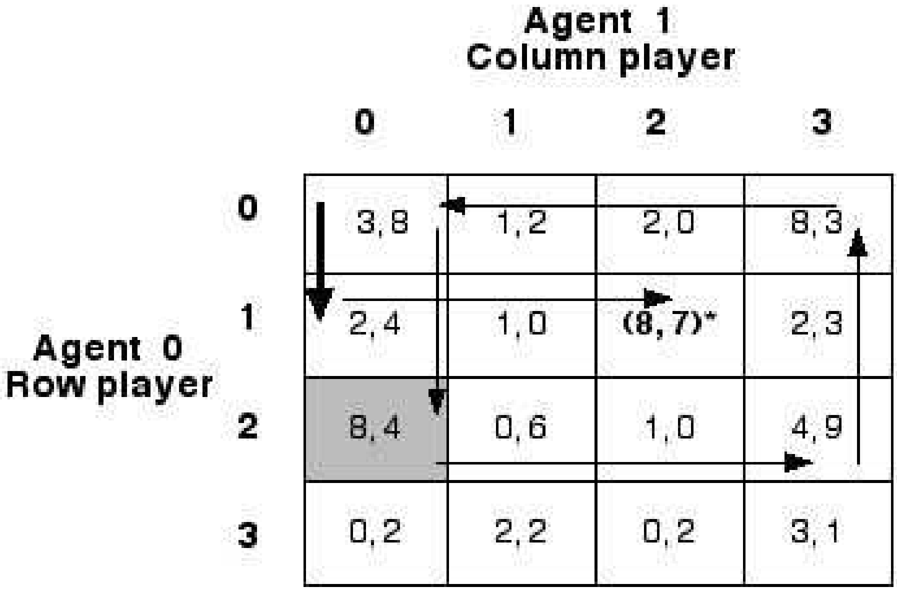
2.3 The Algorithm

Tabu search begins in the same way as ordinary local or neighborhood descent search, proceeding iteratively from one solution to another until a chosen termination criteria is satisfied. Each point (solution) in the solution space has an associated neighborhood and each neighborhood solution is reached from the preceding solution by a move operation.

One difference between the other descent methods and tabu search is the use of adaptive memory in tabu search which keeps track of the exploration process. The efficiency of the exploration process in tabu search is improved by keeping track of not only the local information (like the current value of the objective function) but also of some information related to the exploration process.

Figure 2.1 illustrates a two player game in normal form and shows how the process of applying best response dynamics and tabu search converges to a Nash equilibrium. Using Figure 2.1 we illustrate the use of adaptive memory in preventing cycles in the search process. Let the initial action profile (starting solution for the search process) be $(0, 0)$. The payoff to the agents for this action profile is $(9, 5)$. Let agent 0 be the first to find the best response. Agent 0 is not able to find the best response to agent 1's action of 0 because if agent 1 chooses to play action 0, then the payoff maximizing action for agent 0 is the action 0 yielding a surplus of 9. Figure 2.3 shows the management of the tabu list with explicit memory. At the beginning of the search process, the tabu list is empty. The solution (0,0) is added to the tabu list of agent 0 because a transition from any other solution to solution (0,0) will return the search to a previously explored solution and will result in a cycle. Now it is agent 1's turn to find the best response to player 0's action of 0. As shown in Figure 2.1 and Figure 2.3, the process of finding the best responses continues between the two players and the tabu lists are updated. When the search process reaches the action profile $(3, 0)$ and it is agent 0's turn to find the best response, the use of the tabu list comes into the picture. Even though action 0, resulting in surplus of 9, is the best response for agent 0, it is not chosen because it is in the tabu list. Instead, the next best action, 1, is chosen. Selecting action 0 would have resulted in a cycle. Continuing further, the search process reaches the action profile $(1, 1)$. The action profile $(1, 1)$ is a point in the space of outcomes where both player's actions form best responses to one another. The solution where both the row player and column player choose action 0 is marked as tabu for the row player (shown as shaded cell in the Figure 2.1).

Another point of departure between the simple descent methods and tabu search is that in simple descent methods moves are permitted to the neighbor solutions that improve the current objective function value and ends when no improving solution is found whereas in tabu search non improving moves are also permitted. The simple descent method can get stuck on a local optimum which might not be a global optimum, while tabu search can escape from local optimum. Figure 2.2 and 2.4 shows temporary acceptance of new inferior



		Agent 1 Column player			
		0	1	2	3
Agent 0 Row player	0	3,8	1,2	2,0	8,3
	1	2,4	1,0	(8,7)*	2,3
	2	8,4	0,6	1,0	4,9
	3	0,2	2,2	0,2	3,1

Figure 2.2: An example where tabu search process takes a non-improving move to come out of the local optima.

solutions, in order to avoid paths already investigated. Agent 0 chooses a non-improving move to move from $(0, 0)$ to $(1, 0)$ instead of $(2, 0)$. Figure 2.4 shows the functioning of the algorithm with attribute based memory. The actions are used as an attribute and are tabu for a limited period as specified by the tabu tenure. In the example of Figure 2.4 we use the tabu tenure of 2.

2.3.1 Formal description of the algorithm

Consider an n player game in normal form where N is the set of players. For each player $i \in N$ there is a nonempty set A_i (the set of actions available to player i). The number of actions available to each player is $|A_i|$. Every possible combination of joint actions is a point in the search space and thus the size of the search space will be $\prod_{i=1}^{N} |A_i|$. Let $x \in X$ be a solution in the search space and X is a set of all solutions. The element x is an action profile consisting of one action from each player. Let x_i denote the action of player i for the action profile x and x_{-i} denote the actions of all other players for the action profile x. For x^* to be a Nash equilibrium it must be that no player i can profitably

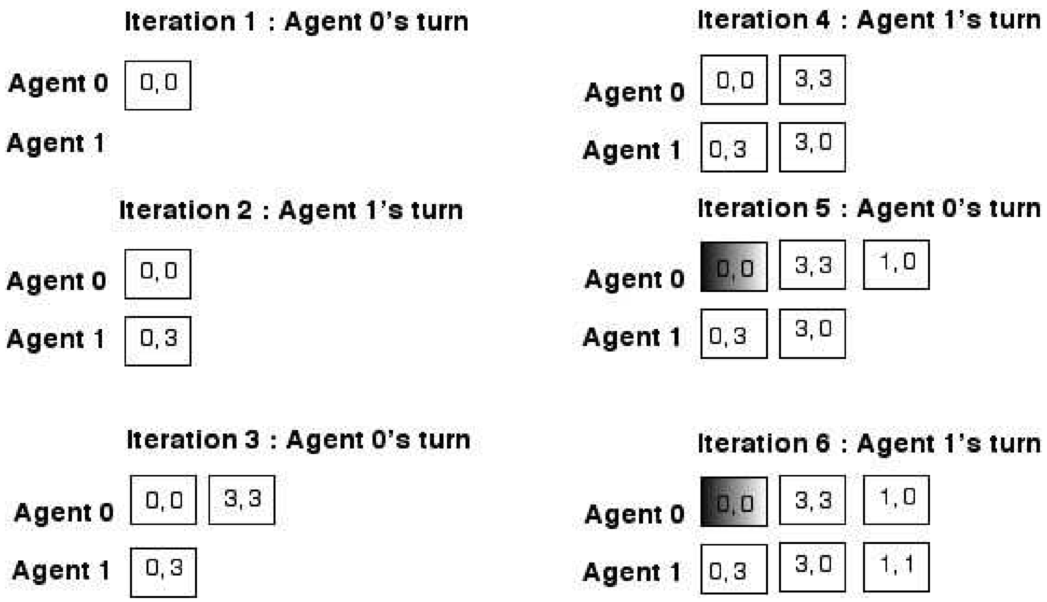

Iteration 1 : Agent 0's turn
Agent 0
0,0
Agent 1
Iteration 2 : Agent 1's turn
Agent 0
0,0
Agent 1
0,3
Iteration 3 : Agent 0's turn
Agent 0
0,0
3,3
Agent 1
0,3
Iteration 4 : Agent 1's turn
Agent 0
0,0
3,3
Agent 1
0,3
3,0
Iteration 5 : Agent 0's turn
Agent 0
0,0
3,3
1,0
Agent 1
0,3
3,0
Iteration 6 : Agent 1's turn
Agent 0
0,0
3,3
1,0
Agent 1
0,3
3,0
1,1

Figure 2.3: The status of the tabu list (explicit memory) after each iteration for the search process shown described in Figure 2.1.

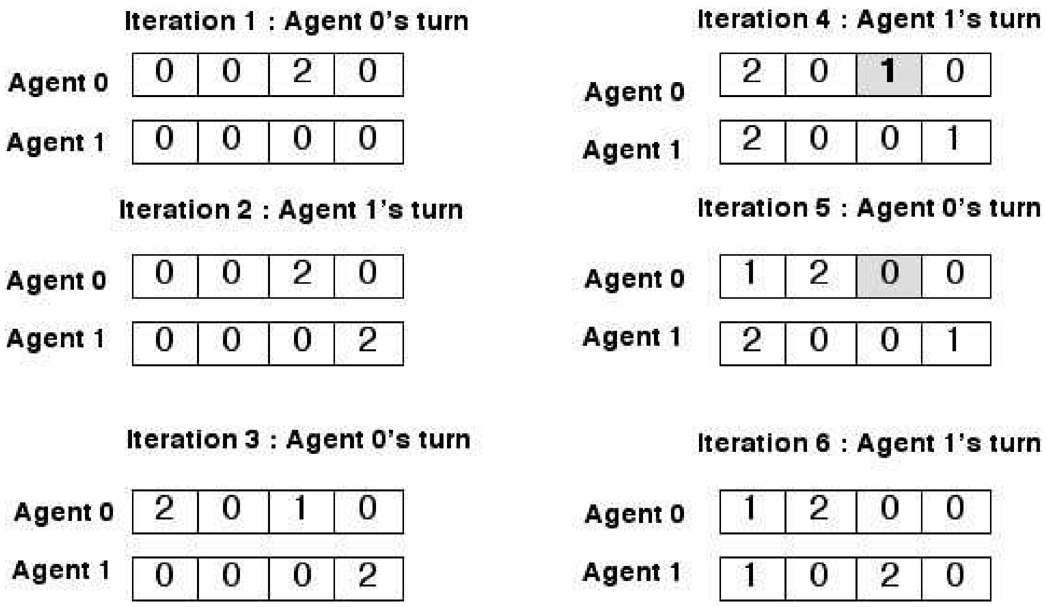

Iteration 1 : Agent 0's turn
Agent 0
0 0 2 0
Agent 1
0 0 0 0
Iteration 2 : Agent 1's turn
Agent 0
0 0 2 0
Agent 1
0 0 0 2
Iteration 3 : Agent 0's turn
Agent 0
2 0 1 0
Agent 1
0 0 0 2
Iteration 4 : Agent 1's turn
Agent 0
2 0 1 0
Agent 1
2 0 0 1
Iteration 5 : Agent 0's turn
Agent 0
1 2 0 0
Agent 1
2 0 0 1
Iteration 6 : Agent 1's turn
Agent 0
1 2 0 0
Agent 1
1 0 2 0

Figure 2.4: The status of the tabu list (attribute based memory) after each iteration for the search process shown in Figure 2.2.

deviate, given the actions of the other players i.e. the following condition holds:

$$(x^*_{-i}, x^*_i) \succeq_i (x^*_{-i}, x_i) \ \ for \ all \ \ x_i \in A_i, \tag{2.1}$$

The steps of the algorithm are as follows:

[1] Initialization Randomly generate a starting solution $x^{current} \in X$.

Initialize the tabu list T_i for each player i . The length of the tabu list is L and initially all the tabu lists are empty for the explicit memory type.

For attribute based memory the length of tabu list for each player is equal to the number of actions available to him. Initially, all the attributes have a value of 0 (i.e, no action is marked as tabu). The maximum value for an element in the tabu list is equal to the tabu tenure t.

[2] Finding best response This step is performed once for player i. The solution obtained as a result of applying the best response for player i serves as the initial solution for the next player's search of finding the best response. Determine $\beta_i(x^{current})$, the best response of player i. If the player is not able to find a better response then we skip the rest of the steps and move on to the next player.

If the player is able to find a better response then we denote $x^{trial} \in X$ as the solution resulting from changing the strategy from x_i in $x^{current}$ to $\beta_i(x^{current})$.

IF explicit memory:
Check to see if the solution x^{trial} is in the tabu list T_i of player i (i.e, $x^{trial} \in T_i$). If the solution x^{trial} is not in the tabu list then skip rest of step [2] and move on to the next step of **Update and termination**.

SET flag = true
While (flag) {

```
If !($x^{trial} \in T_i$), Then flag = false
Chose the next best solution $x^{'}$ and set $x^{trial} = x^{'}$.
}
```

IF attribute based memory:

Let the index of the strategy for player i in the strategy profile x^{trial} be j. Let the value of an element in the tabu list T_i for index j be T_i^j. Check to see if the solution x^{trial} is in the tabu list T_i of player i (i.e, $T_i^j > 0$). If the solution x^{trial} is not in the tabu list then skip rest of step [2] and move on to the next step of **Update and termination**.

IF attribute-based memory:

```
SET flag = true
While(flag) {
If !($T_i^j > 0$), Then flag = false
Chose the next best solution $x^{'}$ and set $x^{trial} = x^{'}$.
}
```

The resulting solution x^{trial} can lead to an improvement in the surplus for player i. If $surplus_i(x^{trial}) > surplus_i(x^{current})$ then we say that the search accepts a improving move else the search moves to a non-improving solution.

[3] Update and termination Set $x^{current} = x^{trial}$ chosen from the previous step.

If explicit memory:

Update the tabu list T_i by adding $x^{current}$ to the front of the list and removing a solution from the back of the list if there exists one to maintain the length of T_i to L.

If attribute based memory:

The tabu list are updated by changing the values of the tabu tenure for each action. $T_i^j = T_i^j + 1$ and $\forall_{k \neq j},\ T_i^k = T_i^k - 1$.

```
If $T_i^k < 0$ Then SET $T_i^k = 0$.
SET terminate = false
IF $(\forall_i,\ x^{current} = \beta_i(x^{current}))$ AND $(\forall_i,\ !(x^{current} \in T_i)$ {
terminate = true
}

If (terminate=true), then stop
else go to step [2] finding best response.
```

2.4 Experimental Results

To evaluate the algorithm, we ran a series of experiments. Before discussing the results, we present a list of criteria for evaluating the performance and efficiency of this type of algorithm. The amount of time required to converge to a Nash equilibrium is definitely one of the most important performance indicators, but just recording the absolute time is not sufficient as it is dependent on the type of the problem and cannot be used as a common benchmark for comparing across problems of different types. For example, if we say that it takes on average 20 minutes to converge to a Nash equilibrium for a problem of type 1 and 40 minutes for a problem of type 2, we are not taking into account the time required to compute each payoff. It might turn out that the time required to find solution in the problem of type 2 is high because it is more computationally expensive to compute the payoff values than for problem of type 1, even though the actual number of steps required to converge and the number of cells visited are fewer. We present two evaluation criteria that are independent of the problem types and are directly proportional to the speed of convergence or the total amount of time required to find the solution. The two criteria are as follows:

1. **Average percent of solution space explored**
   We propose that the average number of solutions explored, or the percentage of the search space explored in order to converge to a Nash equilibrium, is the most important performance indicator for the algorithm. The main emphasis for the search

algorithm is to find a Nash equilibrium by minimum evaluation of the game matrix as the algorithm is specifically designed for situations where it is computationally expensive to obtain a payoff matrix.

Average percent of solution space explored = (average number of solutions visited)/(size of the solution space)

The minimum value of the percentage solution space explored =

$$\frac{1 + \sum_{i=1}^{N}(|A_i| - 1)}{\prod_{i=1}^{N}|A_i|} \tag{2.2}$$

2. **Average Number of steps to converge**
   When an agent transitions from one solution to another during the search process, it it counted as one step. We record the average number of steps to converge to a Nash equilibrium. The minimum number of steps is zero and will occur for the case where the starting solution itself is a Nash equilibrium.

We also record the maximum number of steps to converge and the maximum percentage of strategy space explored to keep track of the worst case performance of the algorithm. Tabu search parameters like the number of tabu conditions encountered and the number of times non-improving moves made are also recorded to study the effect of tabu search control parameters like the type of memory used and the tabu tenure on the effectiveness of the search

For testing our algorithm we generated normal form games using GAMUT, a suite of game generators designated for testing game-theoretic algorithms [29]. The proposed algorithm requires a normal form game as input and we used different parameterization options of GAMUT to generate random games and some well known games in normal form. Figures 2.5, 2.6, 2.7 and 2.8 show the experimental results of applying the algorithm for random games generated by GAMUT. We generated many random games of 5 players and 10 actions. The size of the solution space is 10^5. Keeping the size of the solution space constant, we vary the the number of Nash equilibria that exists. We run the algorithm 10^5 times starting once from each solution and track the average number of steps to converge and the average amount of the solution space explored for each game. The performance of the algorithm depends heavily on the number of Nash equilibria that exists. As shown

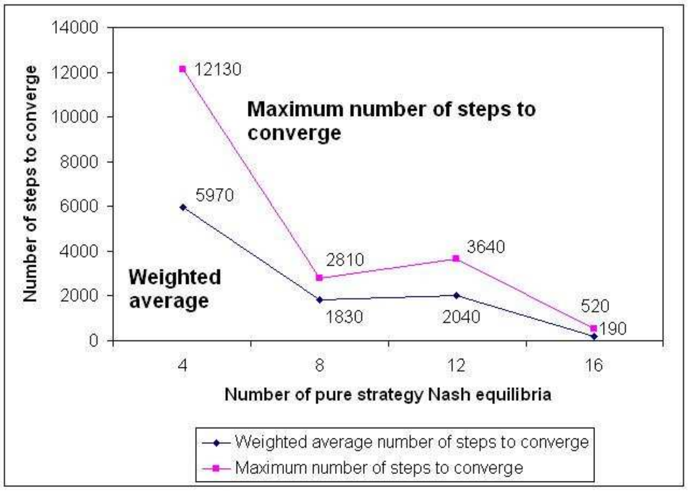


Figure 2.5: Effect of varying the number of pure strategy Nash equilibria on the average and maximum number of steps to converge for explicit memory version of the algorithm.

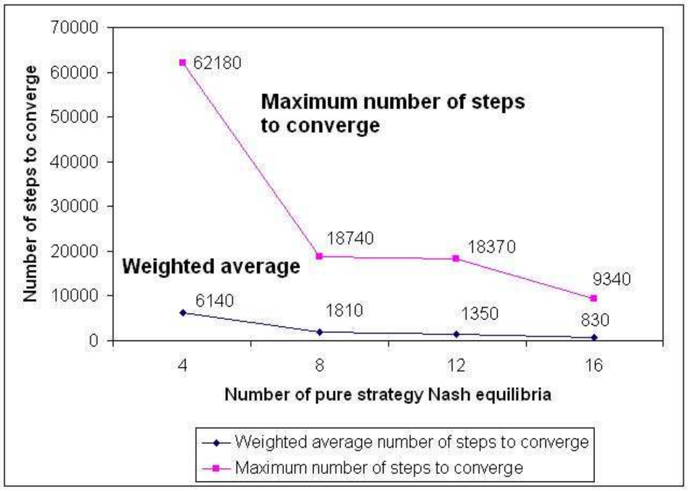


Figure 2.6: Effect of varying the number of pure strategy Nash equilibria on the average and maximum number of steps to converge for attribute based memory version of the algorithm.

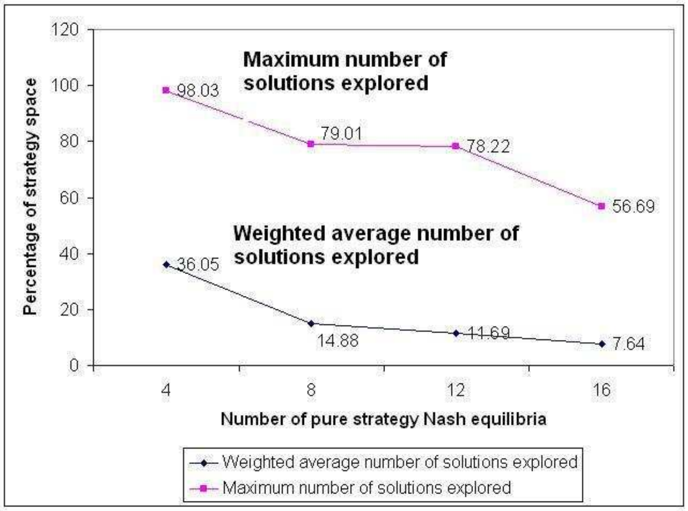


Figure 2.7: Effect of varying the number of pure strategy Nash equilibria on the average and maximum number of solutions explored as a percentage of strategy space for attribute based memory version of the algorithm.

in Figures 2.5, 2.6, 2.7 and 2.8, the algorithm converges faster as the number of Nash equilibria in a game increases. We observe that the worst case performance for the explicit memory version of the algorithm is better than the attribute based memory version of the algorithm. On the other hand, on average the attribute based memory version of the algorithm performs well.

Figure 2.9 shows the performance of the algorithm for four different input sizes of the Traveler's Dilemma, Minimum Effort and Covariant games. The y-axis represents the average percentage of the strategy space explored. The number of experimental runs were equal to the size of the solution space. Each point in the search space was used as a starting solution. For example, in the 11-player, 5-action game, the number of times the algorithm was run to compute the Nash equilibrium was $5^{11} = 48828125$. We observe that the algorithm's performance depends on the distributions of the payoffs and the input sizes. For example, keeping the input size the same, the algorithm exhibits a highly different behavior between the Covariant game and the Traveler's Dilemma game. The difference between the runtime behavior is less visible for the Traveler's Dilemma and the Minimum Effort game.

For every run of the algorithm we record the number of times the search encounters a solution marked as tabu and the number of times the search accepts non-improving moves.

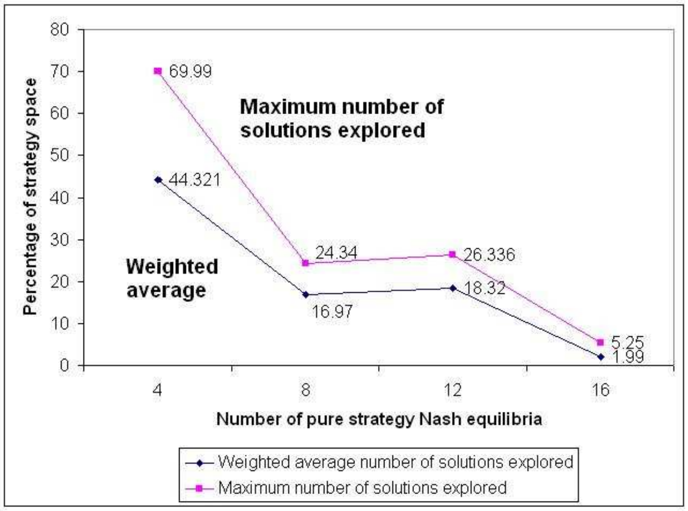


Figure 2.8: Effect of varying the number of pure strategy Nash equilibria on the average and maximum number of solutions explored as a percentage of strategy space for explicit memory version of the algorithm.

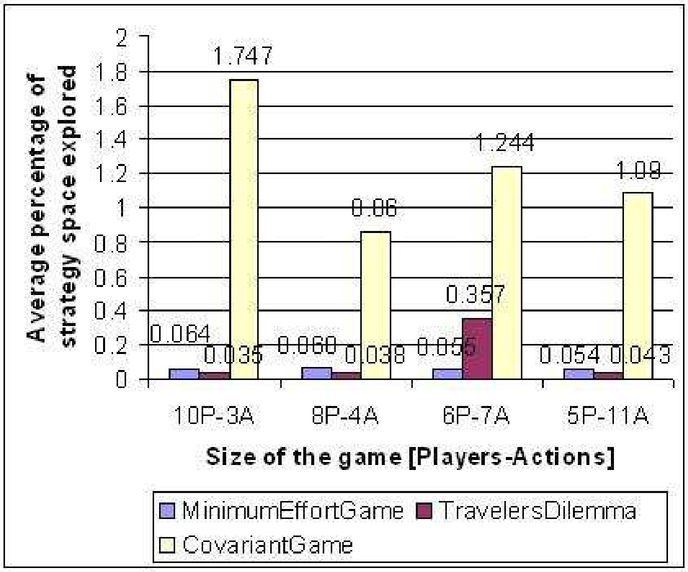


Figure 2.9: Effect of the type of the game and the size of the game on the average percentage of strategy space explored to find the Nash equilibrium.

Equilibrium action profile	**36473**	**64598**	**80337**	**91904**
Number of times found	2550	96240	1050	160
Number of times found (%)	2.5	96.2	1.0	0.1
Average number of steps to converge	8484.0	5915.1	5763.9	2789.9
Maximum number of steps to converge	11477	12137	10116	77
Average number of solutions explored	55848.1	44081.0	41810.9	21905.4
Average number of solutions explored (%)	55.8	44.0	41.8	21.9
Maximum number of solutions explored	68116	69990	63385	53564
Maximum number of solutions explored (%)	68.1	69.9	63.3	53.5
Average Number of tabu conditions	817.3	398.3	433.0	154.2
Average Number of non-improving moves	103.8	54.1	59.6	22.0

Table 2.1: Simulation results of running the explicit memory version of the algorithm on a 5-Player 10-Action random game consisting of four Nash equilibria.

Equilibrium action profile	**02835**	**19745**	**36200**	**42559**	**46903**	**49660**
Num Times	30	1210	740	40	830	95270
Num Times (%)	0.03	1.1	0.7	0.04	0.8	95.2
Av. steps	1.5	1114.1	1067.2	2.1	1032.9	2085.3
Max Num of steps	2	2861	3065	4	2729	3649
Av solutions explored	52	9857.0	9614.7	59.5	9797.3	18698.7
Av solutions explored (%)	0.05	9.8	9.6	0.06	9.8	18.7
Max Num solutions explored	55	25031	26336	82	23926	30693
Max Num explored (%)	0.05	25.0	26.3	0.08	23.9	3.0
Av Num tabu cond	0	35.3	27.3	0.0	18.7	58.9
Av Num non-improving	0	4.4	3.3	0.0	1.2	5.8

Table 2.2: Simulation results of running the explicit memory version of the algorithm on a 5-Player 10-Action random game consisting of six Nash equilibria

Table 2.1 and 2.2 present simulation results of running the explicit memory version of the algorithm on a 5-player, 10-action random normal form game consisting of four and six Nash equilibria respectively. We notice that there are several instances where the algorithm encounters solutions marked as tabu and moves to an alternate solution (possibly non-improving) required to break a cycle in best response dynamics.

We model combinatorial auctions as multi-player complete information games and assume that the bidders participating in the auction uses a proxy agent that follows a myopic best response bidding strategy. Myopic best response bidding is a simple bidding strategy in which in each round, the myopic bidder bids on the bundle that gives it the highest surplus as if this were the last round of the auction. We applied the proposed algorithm on normal form games generated by running the Ascending Package Auction and Ascending k-Bundle Auction [2, 47]. We notice that the performance of the algorithm depends on the type of the auction and the valuation profile of the bidders. We applied the algorithm on a number of problem instances in which we varied the number of players and number of actions available to each player. The results of the simulations so far shows that the average number of solution space explore varies from 13% to 39%.

2.5 Conclusion

We looked at a relatively unexplored area of finding pure strategy Nash equilibria using heuristic search techniques. These approaches are applicable in situations where the size of the strategy space is very large and where it is computationally expensive to compute the payoffs. We presented a list of criterion for evaluating the performance and efficiency of this type of algorithm and argue that the ratio of the number of cells of the normal form game visited to the total number of cells is an important metric to measure the effectiveness of the algorithm. Although it is difficult to produce theoretical guarantees about the performance of this type of algorithm, our empirical tests on standard, random and combinatorial auctions games show that the algorithm can significantly reduce the computational cost.

Chapter 3

Metaheuristic Techniques for Finding Nash Equilibria In Combinatorial Auctions

As mentioned in Chapter 1, one of the problems making it computationally expensive to find Nash equilibria of combinatorial auctions is the large size of bidder's strategy space. The large size of strategy space results in an increases in the number of cells of the normal form game thereby making an exhaustive search infeasible. Chapter 1 describes best response dynamics and explain its usefulness in finding a Nash equilibrium. In Chapter 2, we described a novel algorithm inspired from tabu search and best response dynamics to find an equilibrium of a normal form game by evaluating payoff matrix at runtime. The algorithm uses an exhaustive search (enumeration) to find the best response strategy at each iteration of the best response dynamics. The difficulty arises when the size of the strategy space becomes very large, making it computationally expensive to find the best response strategy by enumeration. When enumeration over the entire space of solutions is not possible, heuristics like genetic algorithms and tabu search can be used to find a good solution by sacrificing completeness in return of efficiency. Metaheuristic search algorithms such as genetic algorithms and tabu search have been applied successfully to a number of optimization problems in many engineering disciplines [15, 37]. The work presented in this chapter describes our design and experiences in applying metaheuristic search techniques to find the best response strategy at each step of the best response dynamics process. Figure 3.1 depicts the application of heuristic techniques in the overall process of finding a Nash equilibrium using best response dynamics. Figure 3.1 also shows the limitation of using heuristic technique: the algorithm converges to a candidate equilibrium with payoffs (r,c) which is not a Nash equilibrium. The row player can unilaterally deviate and improve his surplus from r to r^* and similarly the column player can unilaterally deviate and improve his surplus from c to c^*. The best solution is not reached because some of the solutions were unevaluated during the search process as shown by the unshaded cells corresponding to the rows and columns of the equilibrium found. As shown in Figure 3.1, the application of heuristic techniques resulted in a solution that is not a Nash equilibrium but close to the ideal solution in the sense that each player has a small incentive to deviate from the solution found. The solution found can be arbitrary close to the best solution as no proof or theoretical guarantee is provided of the solution quality. It is important to note that a solution is not a Nash equilibrium if any player has an incentive to deviate irrespective of the magnitude of the incentive.

To carry out search by means of metaheuristic techniques, several elements of the problem and search strategy must be defined: a model of the problem and a representation of possible solutions, transformation operators that are capable of changing an existing solution into an alternative solution and a strategy for searching the space of possible solutions using the representation and transformation operators [28]. We designed two algorithms inspired from genetic algorithms and tabu search to explore the space of strategies with the objective of finding a strategy that maximizes the surplus to a bidder. We configure the parameters of metaheuristics to adapt to the problem of finding the best response strategy and present how it can be helpful in finding Nash equilibria of combinatorial auctions.

Following is an outline of the chapter. Section 3.1 presents a mathematical formulation of finding best response strategy as a combinatorial optimization problem. Section 3.2 presents a binary string representation of bidding strategies so that metaheuristic techniques can be applied. A metaheuristic technique is a top-level general strategy and the actual implementation of the technique depends on the context within which the technique will be used. In Sections 3.3 and 3.4, we describe our adaptation of tabu search and ge-

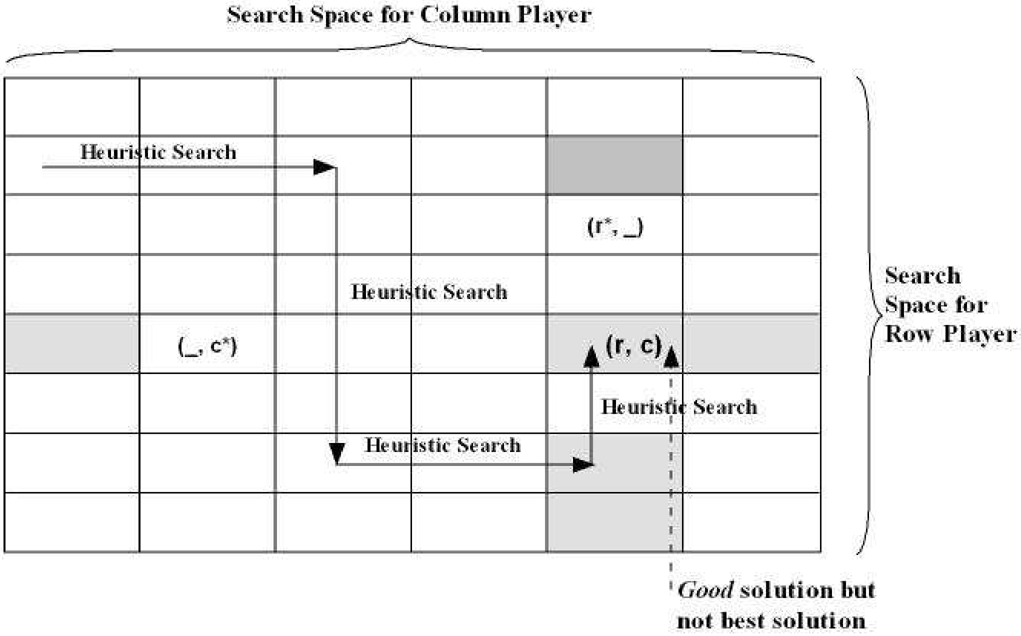

Search Space for Column Player
Heuristic Search
(r*, _)
Heuristic Search
(_, c*)
(r, c)
Heuristic Search
Heuristic Search
Search Space for Row Player
Good solution but not best solution

Figure 3.1: Figure depicting the application of metaheuristic search at each step of the best response dynamics.

netic algorithm to tackle the specific problem of searching the space of bidding strategies in combinatorial auctions. In Section 3.5, we present experimental results. Finally Section 3.6 and 3.7 are the conclusions and future work.

3.1 Best response strategy as combinatorial optimization problem

In this section, we use mathematical notation to formulate the problem of finding the best response strategy as a combinatorial optimization problem. Let us denote the set of agents by $I = \{1, ..., n\}$. Agent $i's$ strategy space is S_i. In our model, we assume a discrete strategy space. The size of the strategy space is $|S_i|$ and is finite. Let $b_i^j \in S_i$ denote the proxy bid vector (strategy) for agent i, and j denote the index of a strategy in the strategy set S_i. Finding a best response strategy can be formulated as an optimization problem. Agent $i's$ task is to find the surplus maximizing strategy b_i^* from the set S_i given the strategies of all other agents. Let the surplus of agent i be denoted by the function

$f(b_i^j, b_{-i})$ where b_i^j represents the proxy bid vector of agent i with an index j in the strategy set S_i and b_{-i} represents the proxy bid vector of all other agents. The combinatorial optimization problem is a maximization problem where agent i wishes to find a bidding strategy b_i^* from the set S_i such that:

$$f(b_i^*, b_{-i}) \; \geq \; f(b_i^j, b_{-i}), \; \forall \; b_i^j \in S_i \tag{3.1}$$

Nash equilibrium of the strategic form game is a strategy profile $b^* \in S$ such that every agent is playing a best response to the strategy choices of his opponents. Formally, we can say that b^* is a Nash equilibrium if $(\forall i \in I)$ $b_i^* \in \beta_i(b_{-i}^*)$ where $\beta_i(b_{-i}^*)$ denotes the best response of agent i given the best responses of all other agents. Hence, finding the best response strategy involves solving one maximization problem and finding the Nash equilibrium to the game using best response dynamics involves solving many maximization problems.

3.2 A binary string representation of bidding strategies

It is important to encode the solution and solution space in a format so that heuristic search techniques can be applied. In this section, we present our design of transforming a bidding strategy into a vector of binary variables in a form applicable to metaheuristic search. The bidding strategy for an agent participating in the auction is represented as a candidate solution for the metaheuristic search and the objective is to efficiently explore the space of bidding strategies and find a surplus maximizing strategy. We represent candidate solutions as a binary string encoding the proxy bid vector. The binary string determines the fraction of the agent's value submitted as a proxy bid to the auctioneer. Each agent i strategy denoted as c_i is translated into the proxy bid vector b_i. $c_i(b)$ is a string of $0's$ and $1's$ representing the strategy of agent i for bundle b. G is the set of all the bundles i.e. $b \in G$. The length of $c_i(b)$ is equal to r (resolution). Similarly, $b_i(b)$ denotes the bid of agent i for bundle b . Let v_i denote the vector representing the true value of agent i for all the bundles and let $v_i(b)$ denote the value of agent i for bundle b. In order to calculate the proxy bid vector we first need to calculate the fraction vector f_i where $f_i(b)$ denotes the fraction computed by diving the decimal number generated from $c_i(b)$ by the maximum decimal number with r bits. For example if $c_b = 10$ then $f_i(b) = (10/11)_2 = 2/3 = 0.667$.

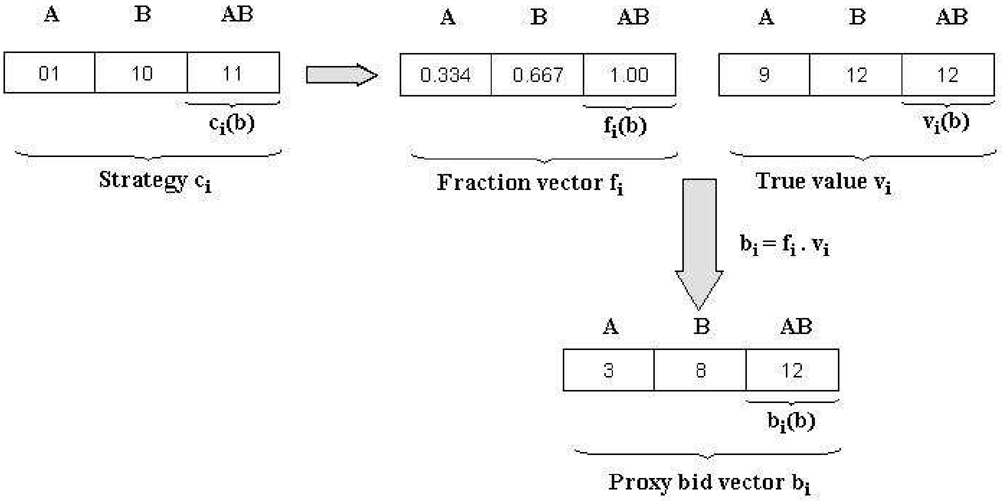


Figure 3.2: Example illustrating the encoding scheme to map the strategy (string of 0's and 1's) to a proxy bid vector.

The proxy bid vector is a vector multiplication of the fraction vector f_i with the true value vector v_i i.e. $b_i = f_i \; . \; v_i$. Figure 3.2 shows an example computation for the proxy bid vector from the strategy vector and true valuation vector. The length l of the strategy c_i is a function of the number of items m and the resolution. The length l is equal to $r(2^m)$.

In Figure 3.2 we see that the length of the candidate solution is a function of the resolution and number of items. All unique combinations of binary variables of length l constitute a strategy and hence the size of the strategy space is 2^l. The decision variable constituting the solution vector are discrete. As we increase the resolution the size of the strategy space increases exponentially. The strategy space also increases exponentially with the increase in the number of items. The number of candidate solutions is very large and grows exponentially with the problem size so that simple enumeration scheme are rendered impractical.

3.3 Genetic Algorithms

In this section we apply genetic algorithm (GAs) as an optimization tool for solving the problem of finding the best response strategy and Nash equilibrium. Genetic algorithms are a rapidly growing area of artificial intelligence, inspired by Darwin's theory of evolution.

It was invented by John Holland at University of Michigan in the 1960s and has been shown to work very well on some types of discrete combinatorial optimization problems. They are less susceptible to getting stuck at local optima than gradient search methods. We used genetic algorithms to search for the best response strategy and Nash equilibrium. Our problem requires searching through a huge number of possibilities in a search space whose structure is not well known and GA's have shown to perform well under such situations.

Figure 3.3 and 3.4 provides a high level overview of genetic algorithms. Figure 3.3 shows that GA's are modelled loosely on the principles of the evolution via natural selection, employing a population of individuals that undergo selection in the presence of variation-inducing operators such as mutation and recombination (crossover). A fitness function is used to evaluate individuals, and reproductive success varies with fitness. As shown in Figure 3.3, a genetic algorithm starts by randomly generating an initial population. It then computes and saves the fitness for each individual (solution) in the current population. Based on the fitness of the population, the algorithm stochastically selects individuals from the population to produce offspring via genetic operators like mutation and crossover.

3.3.1 Genetic operators

In the following section we describe and explain the most important and common elements of GAs: populations of chromosomes (Figure 3.5), crossover as a genetic operator to produce new offspring (Figure 3.6), and random mutation of new offspring (Figure 3.7). Genetic algorithms assume that high-quality parent candidate solutions from different regions in the space can be combined via genetic operators to produce high-quality offspring candidate solutions. Figure 3.5 shows the chromosomes in a GA population taking the form of bit strings. The chromosome is divided into genes (single bits) that encode a particular element of the candidate solution.

Population

A population of individuals are maintained within search space for a GA, each representing a possible solution to a given problem, as shown in the Figure 3.5.

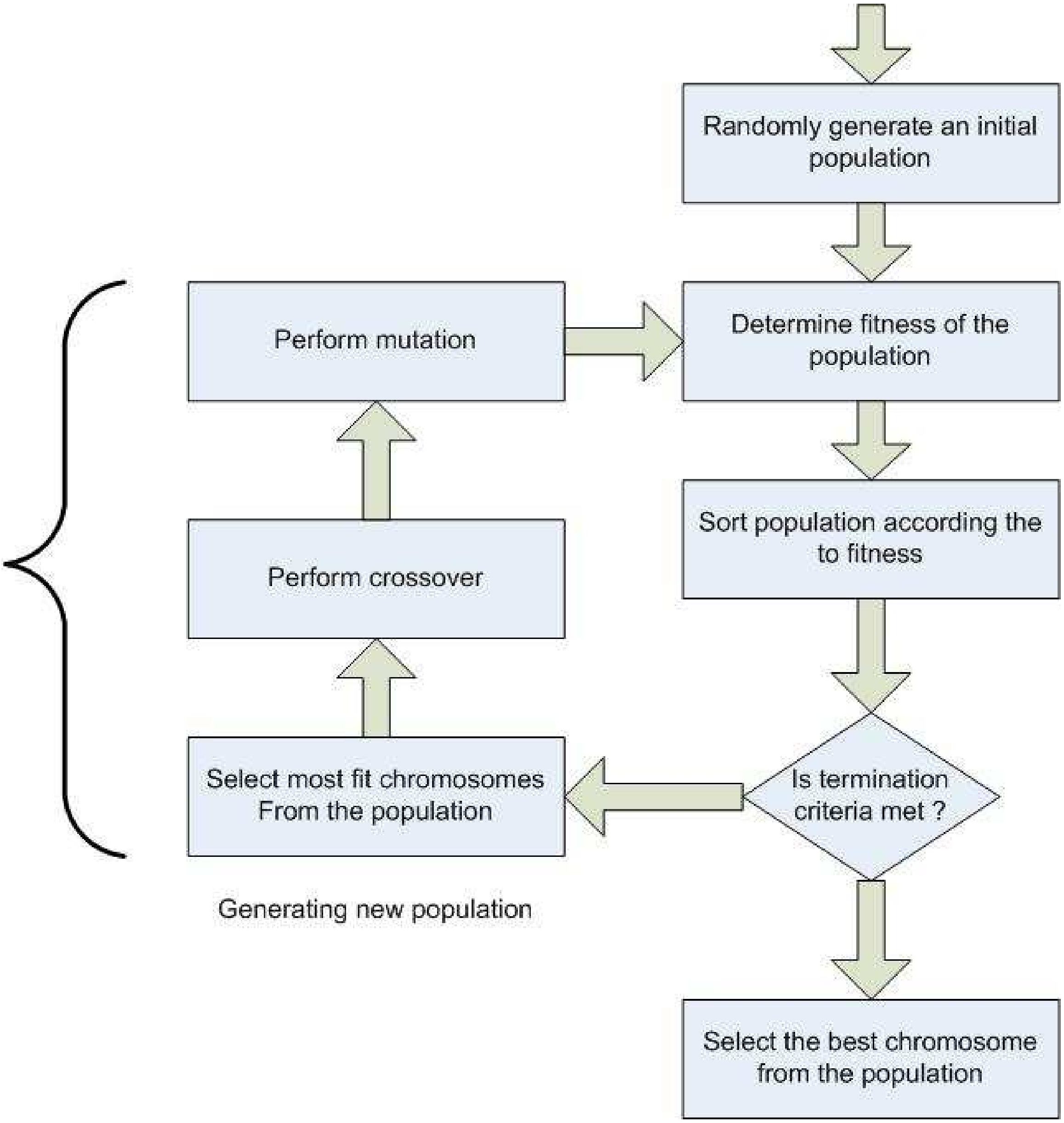

Randomly generate an initial population
Determine fitness of the population
Sort population according the to fitness
Is termination criteria met ?
Select the best chromosome from the population
Select most fit chromosomes From the population
Perform crossover
Perform mutation
Generating new population

Figure 3.3: Figure showing GA as a method for moving from one population of chromosomes (strings of ones and zeros) to a new population by using natural selection together with the genetics-inspired operators of crossover and mutation.

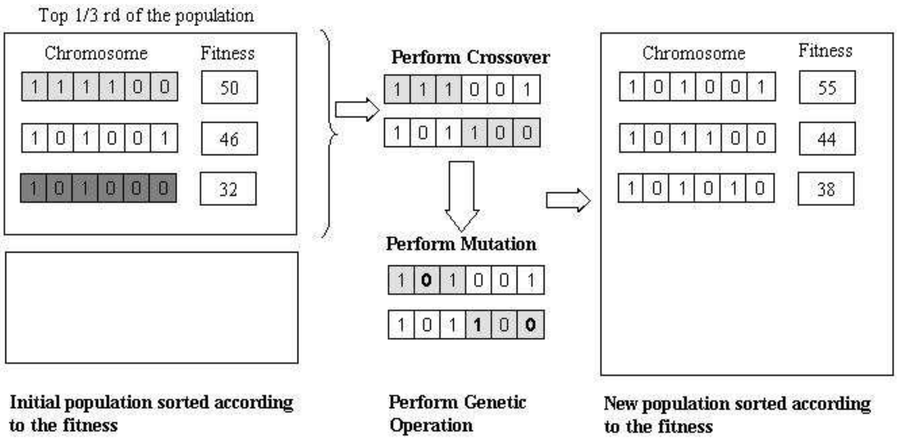
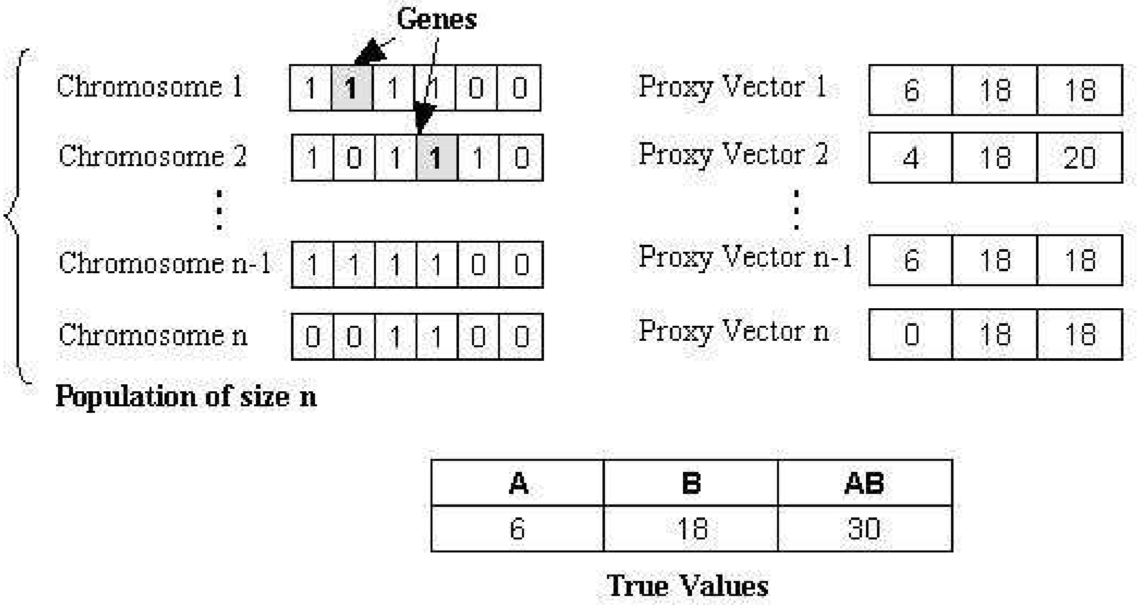


Figure 3.4: Figure showing the high-level overview of GA



A	B	AB
6	18	30

Figure 3.5: Figure showing a population of chromosomes encoded as a bit string that refers to a candidate solution to a problem. The chromosome is divided into genes (single bits) that encode a particular element of the candidate solution. Each chromosome maps to a proxy bid vector.

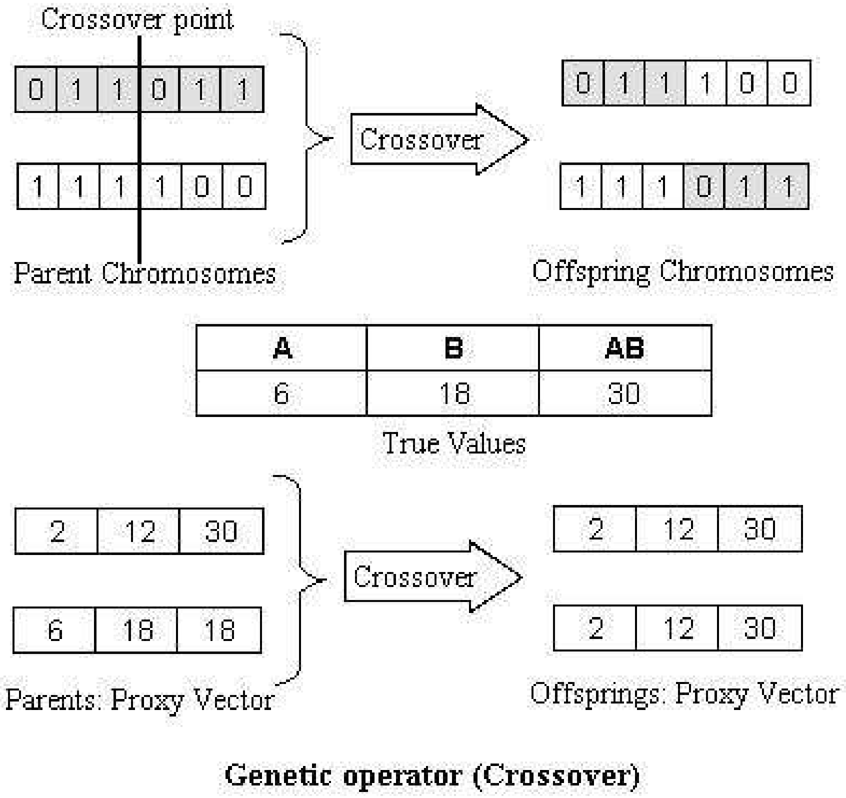


Figure 3.6: Figure showing a crossover operation

Crossover

Crossover in biological terms refers to the blending of chromosomes from the parents to produce new chromosomes for the offspring. The GA selects two strings at random from the mating pool. Then a random splicing or crossover point is chosen in a string, the two strings are spliced and the spliced regions are mixed to create two (potentially) new strings. These child strings are then placed in the new population.

Mutation

Mutation is a genetic operator that alters one ore more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With these new gene values, the genetic algorithm may be able to arrive at

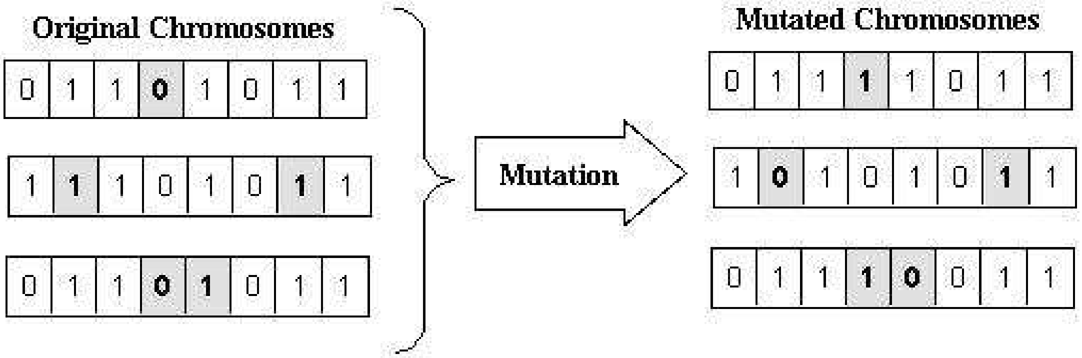


Figure 3.7: This operator randomly flips some of the bits in a chromosome. For example, the string 01101011 might be mutated in its fourth position to yield 01111011. Mutation can occur at each bit position in a string with some probability, usually very small (e.g., 0.001).

better solution than was previously possible and which is not represented in the current population. Mutation is an important part of the genetic search as it helps to prevent the population from stagnating at local optima. The GA has a mutation probability, m, which dictates the frequency with which mutation occurs. For each bit in each string in the mating pool, the GA checks to see if it should perform a mutation. If it should, it randomly changes the element value to a new one. In our binary strings, 1s are changed to 0's and 0's to 1's. Mutation is performed after the crossover operation.

3.4 Tabu Search

Figure 3.8 shows a high level overview of tabu search. Tabu search begins in the same way as ordinary local or neighborhood descent search, proceeding iteratively from one solution to another until a chosen termination criteria is satisfied. Each point in the search space has an associated neighborhood reached by a move operation. The point of departure between the simple descent methods and tabu search is that, in simple descent methods, moves are permitted to the neighbor solutions that improve the current objective function value and ends when no improving solution is found. In tabu search as in simulated annealing non improving moves are also permitted. The simple descent method can get stuck on a local optimum which in most cases might not be a global optimum. On the

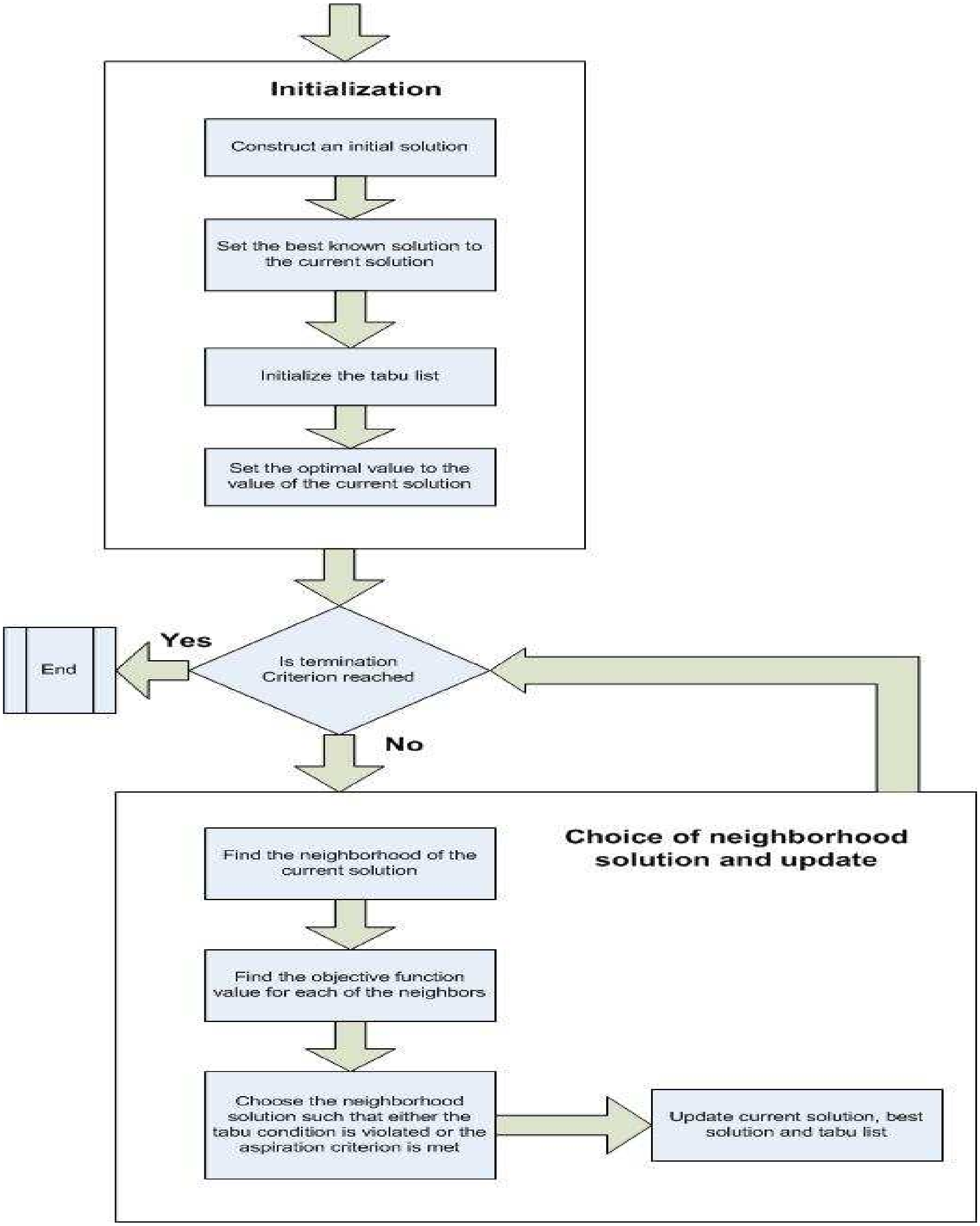

Initialization
Construct an initial solution
Set the best known solution to the current solution
Initialize the tabu list
Set the optimal value to the value of the current solution
Yes
End
Is termination Criterion reached
No
Choice of neighborhood solution and update
Find the neighborhood of the current solution
Find the objective function value for each of the neighbors
Choose the neighborhood solution such that either the tabu condition is violated or the aspiration criterion is met
Update current solution, best solution and tabu list

Figure 3.8: A high level overview of tabu search.

other hand tabu search which allows non improving moves can cross the boundaries of local optimum. Another difference between the other descent methods and tabu search is use of adaptive memory in tabu search which keeps track of the exploration process. The efficiency of the exploration process in tabu search is improved by keeping track of not only the local information (like the current value of the objective function) but also of some information related to the exploration process. It is thus based on procedures designed to cross boundaries of local optimality and explores forbidden regions by systematically imposing and releasing constraints.

3.4.1 Tabu search parameters

The important components of tabu search are the definition of neighborhood function, structure of the memory, aspiration criteria, tabu tenure and termination criteria. The way we define these components or parameters and their values have a strong impact on the quality of the search process.

Neighborhood function

The neighborhood function is applied at every iteration of tabu search to construct a set of solutions from reachable current solution. Let x be a feasible solution belonging to a set of feasible solutions X. A neighborhood function can be defined as $N(x)$ where $j \in N(x)$ is a neighbor of x. The definition of a neighborhood function is a crucial factor in tabu search and has a strong influence on the search procedure [1]. The choice of a neighborhood function influences the trajectory followed in moving from one solution to the next [14]. If a solution x is better than any other solution in its neighborhood then x is a local optimum with respect to its neighborhood. A solution can be a local optimum if we apply one neighborhood function but the same solution might not be a local optimum if we apply another neighborhood function. Applying different neighborhood function to the same problem provides different search trajectories. Tabu search is a flexible framework, and depending on the nature of the problem and the representation of the solution, it allows us to define neighborhood function in several different ways. In our experiments we have used two different neighborhood functions. The first is called as *toggle-bit* and the second *k-deviation*. We assume neighborhoods to be symmetric i.e. x' is a neighbor of x if and

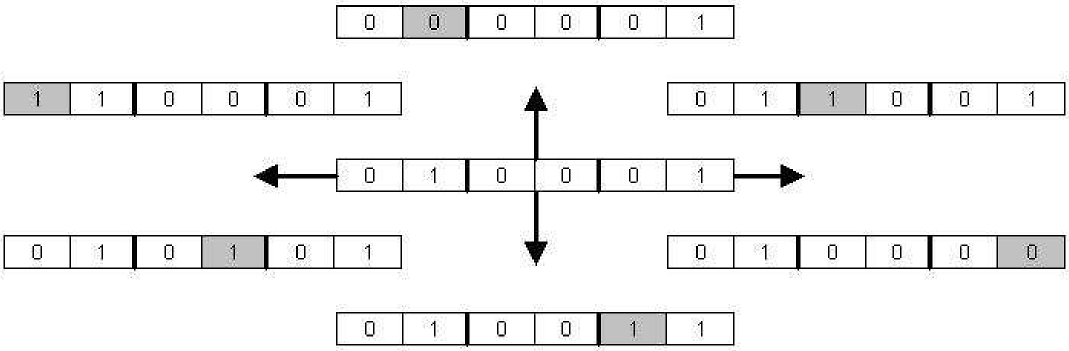


Figure 3.9: Current solution and its neighboring solutions that can be reached by applying the toggle-bit neighborhood function. The shaded cell in the neighborhood solutions specifies the bit that was toggled.

only if x is a neighbor of x'.

Toggle-bit neighborhood function

When we apply toggle-bit neighborhood function, the neighborhood of a solution is a set of solutions that can be reached by toggling one bit in the solution. Figure 3.9 illustrates the results of applying toggle-bit neighborhood function on the binary vector (010001). Since the solution in Figure 3.9 is represented by 6 bits, there are 6 neighbors. The bit that has been toggled is shown by shading the background cell of the neighbor in the figure 3.9.

Figure 3.10 shows the proxy vectors translated from the solutions in Figure 3.9. The shaded cell in Figure 3.10 shows the value of the bundle that was changed as the result of applying the toggle-bit neighborhood function.

K-deviation neighborhood function

We define the k-deviation neighborhood function is a systematic way of finding neighbors of a solution in the proxy vector encoding by increasing and decreasing the value

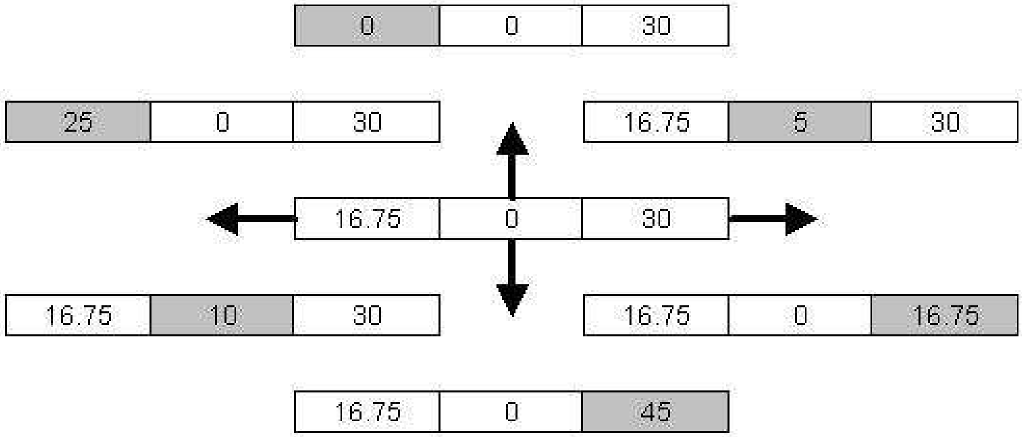


Figure 3.10: Proxy vectors translated from the solutions in figure 3.9.

of the agent for each bundle by a fixed value. Figure 3.11 shows the current solution and its neighboring solutions that can be reached by applying the k-deviation neighborhood function. In the k-deviation neighborhood function we increase and decrease the value of the agent for each bundle by a constant amount equal to k. In Figure 3.11 the value of k is 1. The current solution is (010001) and the resolution is 2. Since the resolution is 2 the value for each bundle is represented by 2 bits. We take these 2 bits and increase its value by 1 to find the upper neighbors. Similarly, to find the lower neighbors we decrease the value of these 2 bits by k. In the example in Figure 3.11, the upper and lower neighbors of the solution (010001) if we take the right most two bits as the agent's value for a bundle will be (010010) and (0100000). Unlike the toggle-bit neighborhood function, more than one bit can be changed to get the neighboring solutions. In the example in Figure 3.11, the value of k is 1. The shaded cells in the neighborhood solutions specifies the bit(s) that were changed as a result of applying the neighborhood function.

Figure 3.12 shows the proxy vectors translated from the solutions in figure 3.11. The shaded cell in figure 3.12 shows the value of the bundle that was changed as the result of applying the toggle-bit neighborhood function.

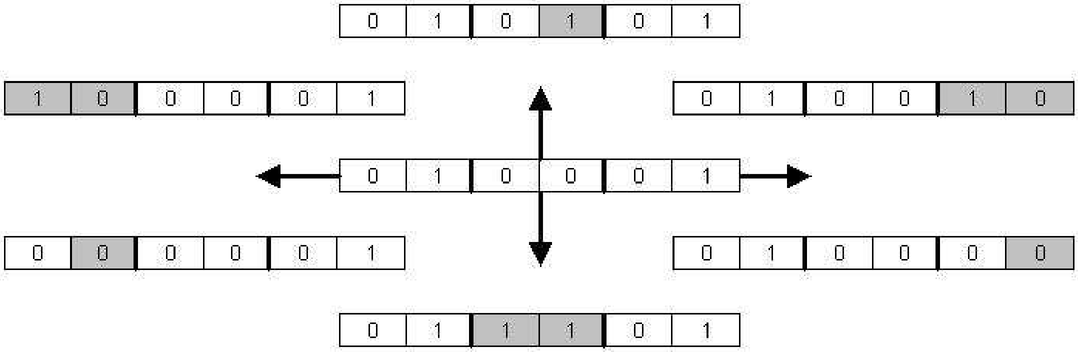


Figure 3.11: Current solution and its neighboring solutions that can be reached by applying the k-deviation neighborhood function. The shaded cells in the neighborhood solutions specifies the bit(s) that were changed as a result of applying the neighborhood function.

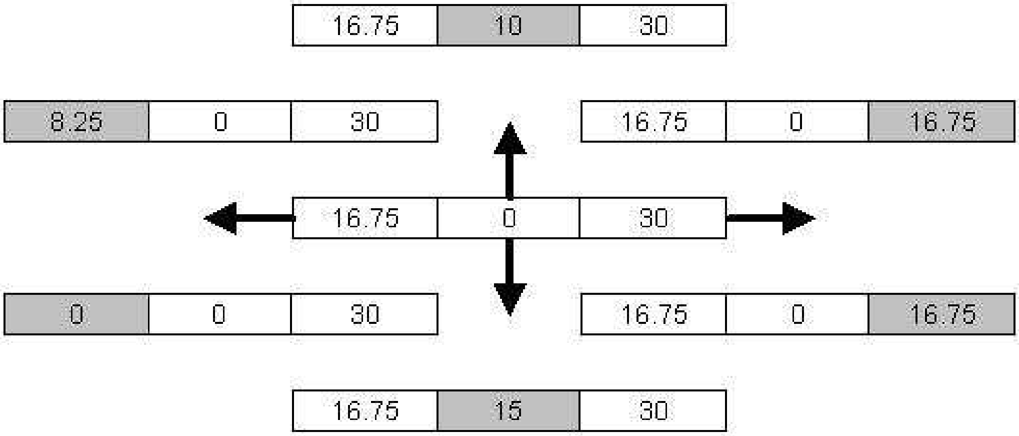
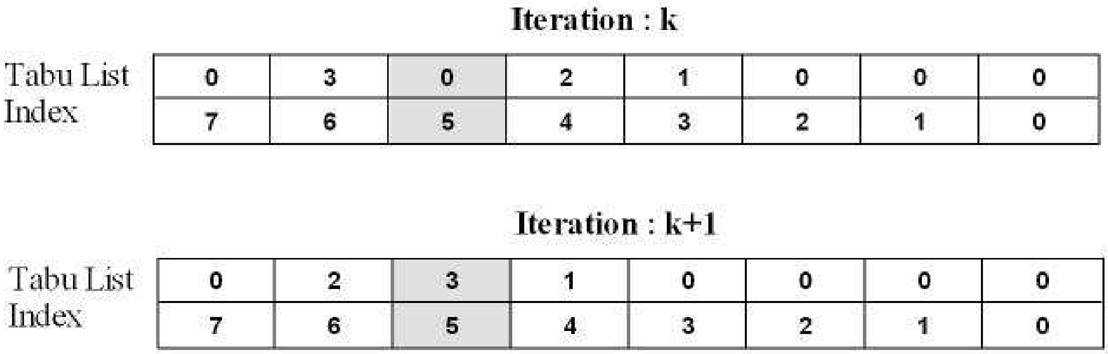


Figure 3.12: Proxy vectors translated from the solutions in figure 3.12.



Iteration : k

Tabu List	0	3	0	2	1	0	0	0
Index	7	6	5	4	3	2	1	0

Iteration : k+1

Tabu List	0	2	3	1	0	0	0	0
Index	7	6	5	4	3	2	1	0

Figure 3.13: Figure showing the functioning of attribute based memory. The neighborhood function used is toggle-bit. The attribute of the solution are the bit indices.

Memory mechanism

Tabu search makes use of memory to classify a subset of the moves in a neighborhood as forbidden (or tabu). The notion of exploiting certain forms of flexible memory to control the search process is the central theme underlying tabu search [35].

There are two types of adaptive memory used in tabu search: explicit memory and attributive memory. As mentioned in Chapter 2, explicit memory records exact solutions and is used to guide the search to avoid visiting solutions more than once. Since the complete solution is recorded, explicit memory structure can have excessive memory requirements depending on the memory required to store a solution and the number of solutions required to be stored in the tabu list. The total memory requirement increases with the increase in the length of the tabu list and the increase in the size of the solution. The amount of memory required by explicit memory version of the algorithm can be limited by setting an appropriate value of the length of the tabu-list. A good value of the length of tabu-list depends on the nature of the problem that we are trying to solve and can be determined empirically.

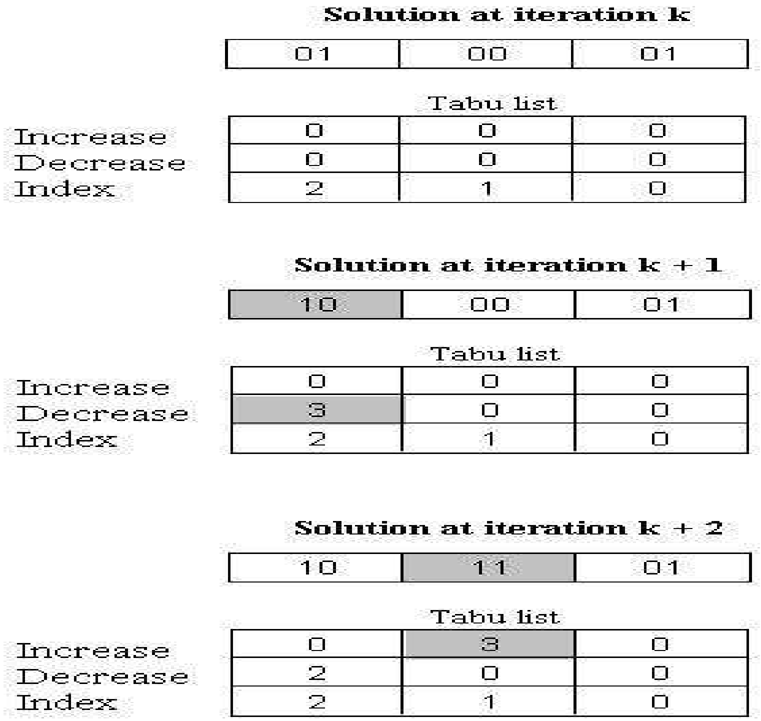
Another form of adaptive memory that tabu search makes use of is called attributive memory. The attribute based memory like explicit memory is also used to guide the search to avoid visiting solutions more than once but rather than recording the exact solutions, attribute based memory records information about solution attributes that change in moving from one solution to another.

Figure 3.13 demonstrates the functioning of attribute based memory in our implementation. As shown in the example of Figure 3.13, the attribute of the solution when we apply the toggle bit neighborhood function are the bit indices. In order to prevent the search from toggling the same bit tried in the recent past, potentially reversing the effects of previous moves, we will classify as tabu all indexes of the solution that have been used previously, for a certain number of most recent moves. The index of the bit will be kept tabu for a duration of a certain number of iterations called the tabu tenure. Figure 3.13 shows the content of a tabu list of size eight for two iterations. At iteration k the content of the tabu list is $(0, 3, 0, 2, 1, 0, 2, 0)$ which means that the bit at index 6 will be tabu for 3 iterations. Similarly, the bit at index 1 will not be allowed to flip for 2 more iterations. At the next iteration $k+1$, we see that a better solution is reached by toggling bit 5. Since bit 5 was not a tabu, we proceed by toggling bit 5 and setting the tabu tabu tenure of bit 5 to 3 and decreasing the tabu tenure of all other bits by 1.

The structure of the attribute based memory in tabu search is dependent on the type of the neighborhood function used. Figure 3.14 shows the functioning of attribute based memory when the neighborhood function used is k-deviation. There are two attributes of the solutions in this case. One attribute is the index of the bundle and the other attribute is whether the value at that index is increased or decreased. As shown in Figure 3.14 there are two tabu list maintained for each bundle. One keeping track of the value increments and the other keeping track of the decrements. Lets say that iteration at $k+1$ as shown in the figure 3.14, a better solution is found by increasing the value of the bundle at index 2. As a result of this the search process moves to iteration $k+1$ by updating the tabu list that is keeping track of the decrements. The value of the decrement tabu list at index 2 is set to the tabu tenure of 3 in order to prevent the search from increasing its value again on the next 3 iterations. Similar, logic can be applied to the next iteration $k+2$.

Aspiration criterion

Aspiration criteria is an important component of tabu search and is used to determine when the tabu restrictions can be overridden. If a neighbor is found to be better than the best solution found so far, it can be selected as a move even when it is tabu if it



Solution at iteration k

01	00	01

Tabu list

Increase	0	0	0
Decrease	0	0	0
Index	2	1	0

Solution at iteration k + 1

10	00	01

Tabu list

Increase	0	0	0
Decrease	3	0	0
Index	2	1	0

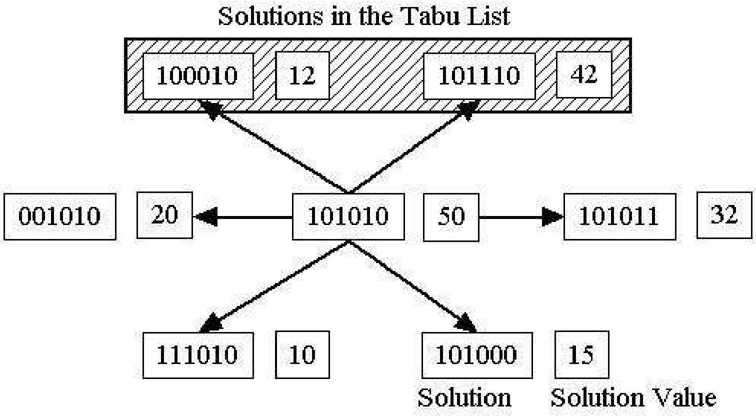
Solution at iteration k + 2

10	11	01

Tabu list

Increase	0	3	0
Decrease	2	0	0
Index	2	1	0

Figure 3.14: Figure showing the functioning of attribute based memory. The neighborhood function used is k-deviation. There are two attributes of the solutions. One attribute is the index of the bundle and the other attribute is whether the value at that index was increased or decreased.



Aspiration Value	Next Solution	Solution Value
0.8	101110	42
0.9	101011	32

Figure 3.15: Figure showing the affect of aspiration criteria during the search process. The figure shows a solution and its neighboring solutions that can be reached by applying the toggle-bit neighborhood function. The figure also shows the value of each solution, the contents of the tabu list and sample aspiration values.

meets the aspiration criteria.

The affect of aspiration criteria can be shown with the help of an example in Figure 3.15. Figure 3.15 shows a solution and its neighbors that can be reached by applying the toggle-bit neighborhood function. In the table on the right side of each solution is its value. For example the value of solution (101010) is 50. Figure 3.15 also shows the contents of the tabu list. The value of the current solution is 50 and the best neighboring solution, with value is 42 is in the tabu list. Normally, if the solution is in the tabu list it cannot be selected but the aspiration condition overrides that. In the example if the aspiration value is 0.8 then any solution whose value is greater that 80 percent of the current value will be selected if it is in the tabu list and is the best neighboring solution. Figure 3.15 shows the current solution and next solution under two different aspiration values.

Termination criterion

Just like the neighborhood function, the memory technique, and the aspiration criteria, the termination criteria also influences the search procedure and the results. Following are the two most common termination criteria used.

- The number of iterations is fixed to some value k.
- The number of iterations since the last improvement of the best solution is larger than a specified number k.

3.4.2 Steps of tabu search

The many ways to define the neighborhood function, memory mechanism, aspiration criteria and termination criteria provide a lot of parameters with which to fine tune the search. We have implemented four variations to test in our experiments. The two types of adaptive memory that we implement are expicit memory and attribute-based memory and the two types of neighborhood functions that we implement are toogle-bit and k-deviation. This results in four variations of the tabu search algorithm. The following sections presents a formal description of the algorithm along with the four variations.

Algorithm

Initialization Randomly generate a starting solution $x^{current} \in X$. The number of bits in the solution is equal to $2^{(number\ of\ goods)} * resolution$ and the size of the search space will be $2^{(number\ of\ bits\ in\ the\ solution)}$.

Record the current best known solution by setting $x^{best} = x^{current}$ and the $best\_surplus = surplus(x^{best})$.

IF explicit memory : Initialize the tabu list T_r. The length of the tabu list is equal to the tabu tenure r and insert $x^{current}$ as all the elements of the tabu list.

IF attribute-based memory AND toggle-bit neighborhood function : Initialize the tabu list T_r. The length of the tabu list is equal to the number of bits in the solution. Set each element of the tabu list to 0.

IF attribute-based memory AND k-deviation neighborhood function : There are two tabu lists denoted as T_r^{inc} and T_r^{dec}. Initialize the tabu list T_r^{inc} and T_r^{dec}. The length of each of the tabu lists is equal to the number of bundles (i.e, $2^{number\ of\ goods}$). Set each element of the tabu list to 0.

Choice of neighbor Determine $N(x^{current})$, the neighbors of the current solution.

IF toggle-bit neighborhood function : The neighborhood of a solution is a set of solutions that can be reached by toggling one bit in the solution. The total number of neighbors is equal to the number of bits in the solution (i.e, $2^{(number\ of\ goods)} * resolution$).

IF k-deviation neighborhood function : The neighborhood of a solution is a set of solutions that can be reached by deviating (increasing and decreasing) the value of proxy bid for each bundle by k. The total number of neighbors is equal to two times the number of bundles (i.e, $2^{(number\ of\ bundles)+1}$).

Find the objective function value for each of the neighbors $x^{trial} \in N(x^{current})$. The objective is to maximize the function, $f(x)$, which denotes the surplus to the agent as a result of choosing the strategy x.

IF explicit memory : Choose the best $x^{trial} \in N(x^{current})$ such that either the tabu condition ($x^{trial} \in T_r$) is violated or the aspiration criteria ($surplus(x^{trial}) > best\_surplus$) is met.

IF attribute-based memory AND toggle-bit neighborhood function : Choose the best $x_i^{trial} \in N(x^{current})$ such that either the tabu condition $T_r(i) > 0$ is violated or the aspiration criteria ($surplus(x_i^{trial}) > best\_surplus$) is met where $T_r(i)$ represents the value in the tabu list for index i.

IF attribute-based memory AND k-deviation neighborhood function : Choose the best $x_i^{trial}(inc/dec) \in N(x^{current})$ such that either the tabu condition $T_r^{inc/dec}(i) > 0$ is violated or the aspiration criteria ($surplus(x_i^{trial}(inc/dec)) > best\_surplus$) is met where $T_r(i)$ represents the value in the tabu list for index i.

Update and termination Set $x^{current} = x^{trial}$ chosen from the previous step.

Set $x^{best} = x^{current}$ and $best\_surplus = surplus(x^{current})$ if $surplus(x^{current}) > best\_surplus$.

	A	**B**	**AB**
Agent 1	25	15	45
Agent 2	20	20	45
Agent 3	10	25	40

Table 3.1: An example combinatorial auction problem with three agents and two items

IF explicit memory : Update the tabu list T_r by inserting $x^{current}$ and removing the solution that crossed its tabu tenure. The new solution is added to the front of the list and the solution from the back of the list is removed.

IF attribute-based memory AND toggle-bit neighborhood function : Update the tabu list T_r by setting the value of $T_r(i)$ to the tabu-tenure r and decreasing the value of $T_r(-i)$ by 1.

IF attribute-based memory AND k-deviation neighborhood function : Update the tabu list T_r by setting the value of $T_r^{inc/dec}(i)$ to the tabu-tenure r and decreasing the value of $T_r^{inc/dec}(-i)$ by 1.

If a termination condition is met, then stop. Else go to the previous step: Choice of neighbor.

3.5 Results

To test the metaheuristic algorithms, we designed a set of problems and classified them into different types depending on the size of the strategy space and the size of the outcome space. Table 3.2 is a list of 13 problem types in increasing order of the size of the outcome space. The size of the outcome space is $|S_i|^n$ where $|S_i|$ is the size of the strategy space for agent i and n denotes the total number of agents. In our model, we assume that the size of strategy space for each agent is the same. The size of the strategy space is 2^l where l denotes the encoding length. The problem size grows exponentially with the number of agents, items and resolution. Table 3.1 is an example problem with three agents and two items. All the agents in the example of Table 3.1 have complementary preferences (valuation of a particular bundle of items is greater than the the sum of valuations of the individual items) and two of the agents are part of the optimal allocation. For a fixed number of items and agents, there are several possible variations in constructing a problem

Problem Size	**Num Agents**	**Resolution**	**Num Items**	**Encoding Length**	**Strategy Space**	**Outcome Space**
1	2	1	2	3	8	10^2
2	2	2	2	6	64	$4 * 10^3$
3	2	1	3	7	128	$2 * 10^4$
4	2	3	2	9	512	$2 * 10^5$
5	3	2	2	6	64	$2 * 10^5$
6	3	3	2	9	512	$2 * 10^8$
7	2	2	3	14	$1.6 * 10^4$	$2 * 10^8$
8	2	3	3	21	$2 * 10^6$	$4 * 10^{12}$
9	3	2	3	14	$1.6 * 10^4$	$4 * 10^{12}$
10	4	2	3	14	$1.6 * 10^4$	$4 * 10^{16}$
11	3	3	3	21	$2 * 10^6$	$8 * 10^{18}$
12	4	3	3	21	$2 * 10^6$	$2 * 10^{25}$
13	4	2	4	30	10^9	10^{36}

Table 3.2: Classification of problem types based on the number of agents, items and resolution

by changing the valuation profile of the agents. For example, we can construct a problem where the agents have sub-additive[1] preferences and every item is allocated to a different agent. We designed several problems by varying the number of agents, the number of items, the resolution and the valuation profiles of the agents to make sure that the results are not an artifact of a particular problem type.

Figure 3.16 and 3.17 are plots of the amount of time (in seconds) and the amount of memory allocated (in MB) to search the space of bidding strategies of one agent and find the surplus maximizing strategy using tabu search algorithm. For the graphs in Figure 3.16 and 3.17, we used an explicit memory version of the algorithm with the toggle bit neighborhood function. We applied the algorithm on several problems with random starting solutions and averaged out the results. The experiments were conducted on a single Apple Macintosh machine. The processor was a 1 GHz PowerPC G4 with 1024MB memory. The development environment was Macintosh Common Lisp version 5.0 on a Mac OS X version 10.2.8. It takes the tabu search algorithm approximately 30 minutes to do a heuristic search on the space of bidding strategies of size of the order of 10^9.

Figure 3.18 and 3.19 are simulation results obtained by applying the search based on genetic algorithm. We observed that the absolute running time required by genetic

[1] valuation of a particular bundle of items is less than the the sum of valuations of the individual items

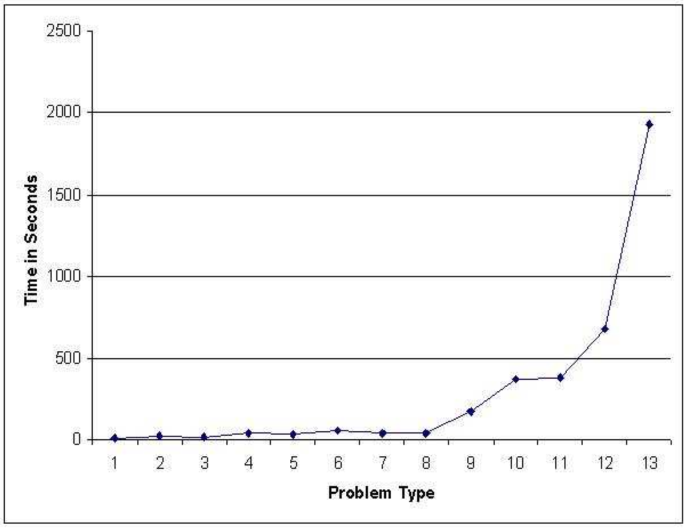


Figure 3.16: Convergence speed of tabu search as a function of problem size.

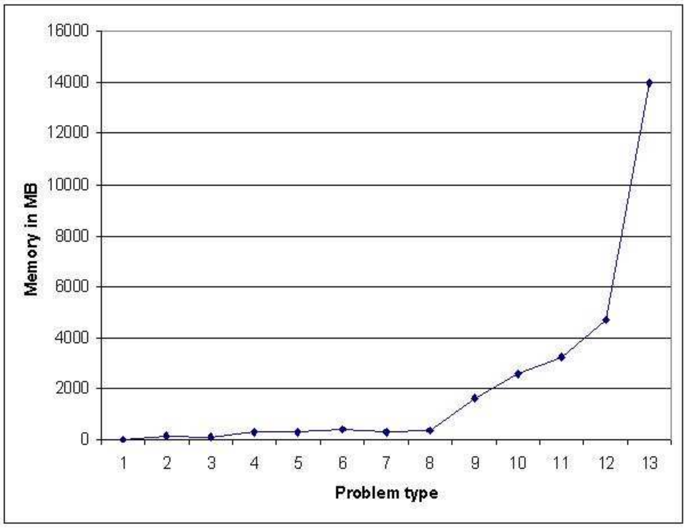


Figure 3.17: Memory Allocated by tabu search algorithm as a function of problem size.

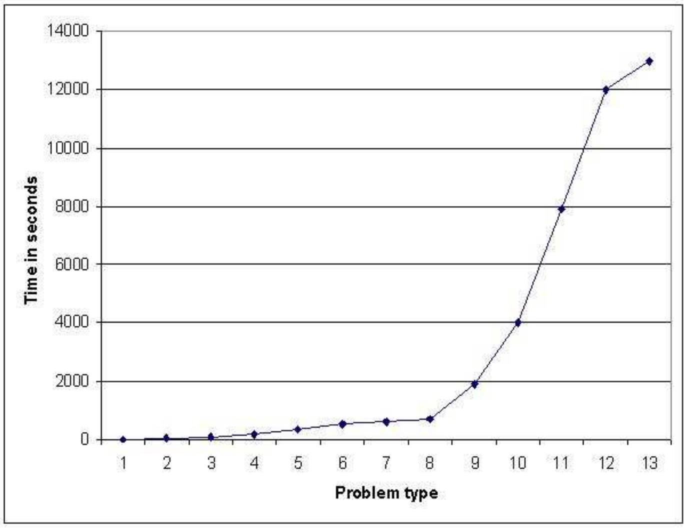


Figure 3.18: Convergence speed of genetic algorithm as a function of problem size.

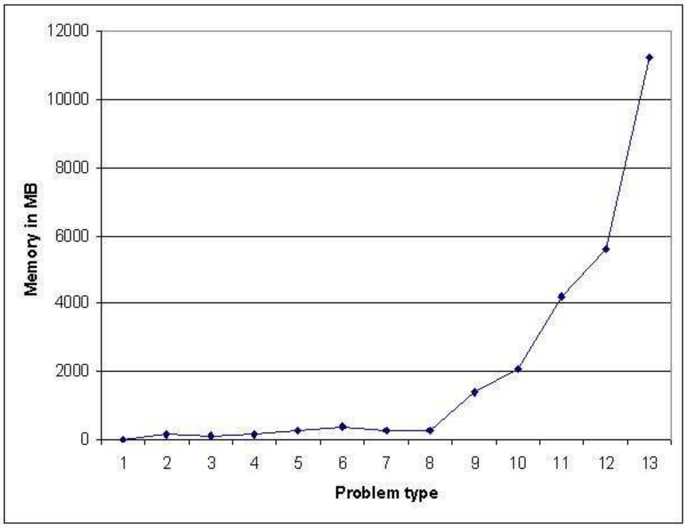


Figure 3.19: Memory Allocated by genetic algorithm as a function of problem size.

algorithm is more than what is required by tabu search. There are several factors that have an effect on the total amount of time required by the program to run, such as the efficiency of algorithm implementing the metaheurisct technique or the way memory is managed or released within a program. We believe that the overall time can be reduced further by improving the efficiency of the algorithm implementing the heuristic search. The results in Figures 3.16, 3.17, 3.18 and 3.19 provides a general idea of an increasing trend of the amount of time and memory required as the size of the problem grows. It is important to note that even though the results indicate that tabu search is a superior metaheuristic technique than genetic algorithm for the problem that we are trying to solve, we do not make that conclusion. Further experiments needs to be carried out in order to do a comprhensive comparison between the two techniques.

It is important to note that the actual time taken also depends on other factors like the bid increment and the type of the auction. Increasing the bid increment results in a faster convergence of the auction and hence can reduce the overall time. Therefore, we also recorded the number of rounds taken to converge to the solution which is independent of the configuration of the machine used, available memory and parameters like bid increment. Figure 3.20 is a plot of the average and maximum number of rounds taken to converge to an equilibrium by tabu search (explicit memory and toggle-bit neighborhood function). Searching the space of bidding strategy once for each agent while keeping the strategies of all other agents fixed is referred to as a single round. Applying the process of best response dynamics can take several such rounds to converge to a Nash equilibrium. The actual number of rounds taken to converge varies with the type of the problem, the starting solution and the auction type. But we can estimate the maximum and average number of rounds to converge for a problem of fixed size. We observe that the average and maximum number of rounds to converge to a equilibrium remains the same till the problem of type 8 and then increases. The largest problem to which we applied the algorithm was of type 13 (refer to Figure 3.2) and it takes on an average 8.3 rounds to converge to a solution. For the results in Figure 3.20 we used an explicit based memory version of the algorithm with toggle-bit neighborhood function. The number of rounds elapsed for termination also increased with the problem size. As shown in the Figure 3.20, we observed that the number of rounds since the last improvement of the best solution for any agent becomes jumps from 2 to 4 as we increase the problem size. We did not observe any noticeable difference between the k-deviation and toggle-bit neighborhood functions. We parameterized the

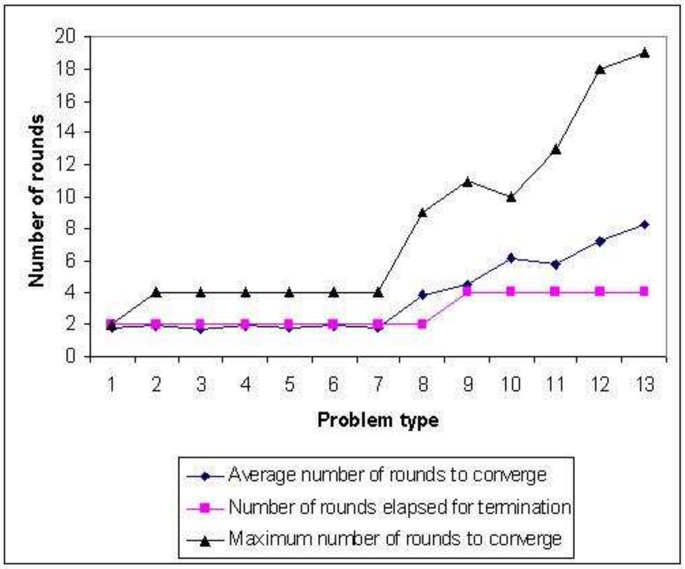


Figure 3.20: Simulation results of applying tabu search on various problem types.

tabu search algorithm and used explicit as well as attribute-based memory version of the algorithm but our experience shows no noticable difference between the memory types and the neighborhood function. The path taken to converge to a equilibrium depends on the type of memory and neighborhood function used by the algorithm but we did not observe any consistant evidence of one type of adpaptive memory or neighborhood function being better than the other. We plan to do further tests on the effect of tabu search parameters on the search trajectory and convergence time.

We carried out experiments to see the effect of the number of tabu search iterations or the number of moves required by tabu search in each run on the quality of solution found. An increase in the number of moves results in an increased coverage of the search space but may not improve the quality of the solution found. If the number of moves are very small then we are just searching a very small portion of the search space. We conducted simulations to find an optimal value of the number of tabu search iterations for a specific problem type. Figure 3.21 shows the experimental results of changing the *numIter* (number of neighborhood moves) parameter on the number of rounds require to converge to a Nash equilibrium. The experiment was performed on problems of various sizes. We observe that for problems of type 9 and 10 the value of optimal numebr of iterations is 4. Setting

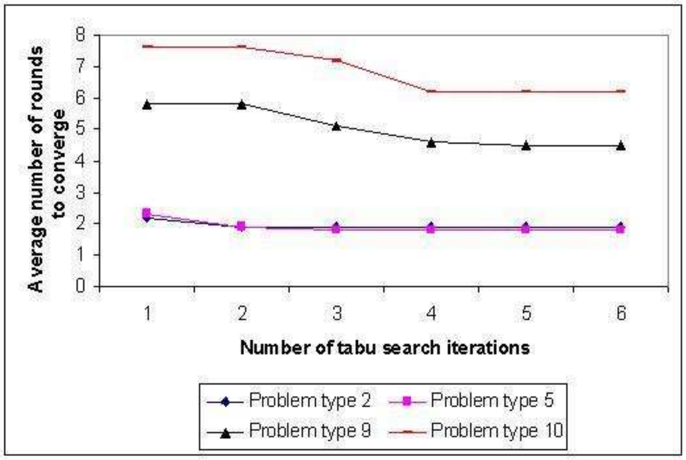


Figure 3.21: Effect of number of neighborhood moves parameter on the number of rounds require to converge

the number of iterations to more than 4 has no additional benefit and does not make the algorithm converge faster whereas a value less than 4 leaves some part of the strategy space uncovered.

We verified the results obtained from tabu search using enumeration. Based on our simulation results we conclude that the quality of the solution found depends on tabu search parameters like the termination criteria and the number of neighborhood moves. We performed experments to find an optimal value of these parameters for a specific problem type. Once the optimal value of these parameters are set, we observed that the candidate equilibrium found is a Nash equilibrium. We verified our results using enumeration and tested it for problems of size up to 13. In future we plan to verify the results for problems of size more than 4 agents and 4 items. Figures 3.22 and 3.23 show the amount of strategy space explored by tabu search to find the best response strategy for problems of various size. For example, we notice that for problems of type 9 and 10 approximately 600 strategies are evaluated from a space of 16000 strategies to find an optimal strategy. We verified this using an exhaustive search and also noticed that there are several optimal solutions and not just one. Figure 3.22 and 3.23 shows the size of the strategy space as a sum of evaluated and unevaluated strategies.

We applied genetic algorithm to problems of various sizes. The path taken to converge to a Nash equilibrium depends on the GA parameters like the population size, number of generations and mutation probability. We conducted experiments to determine

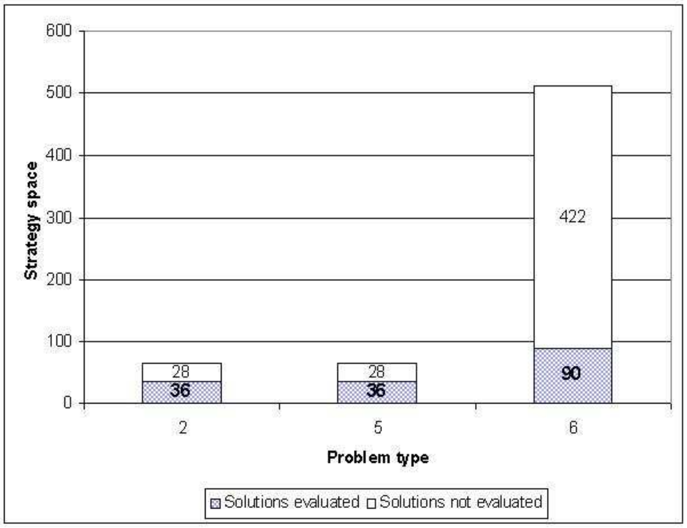


Figure 3.22: Amount of strategy space explored by tabu search to find the best response strategy for problem size 2, 5 and 6.

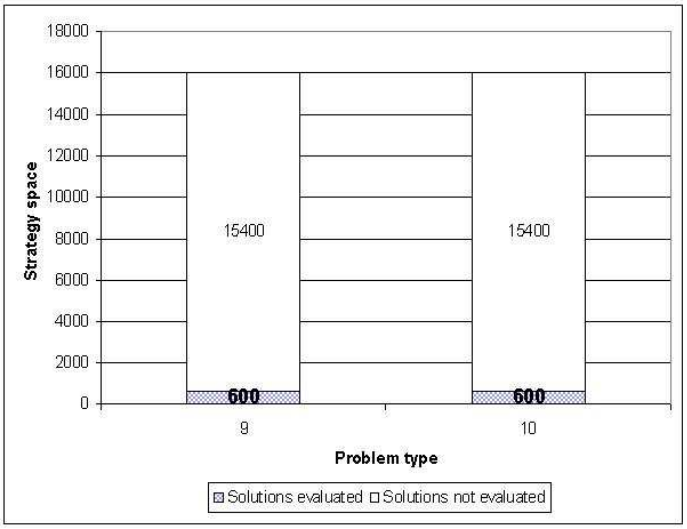


Figure 3.23: Amount of strategy space explored by tabu search to find the best response strategy for problem size 9 and 10.

the optimum population size and the number of generations for a problem of a particular type. We tried a range of population sizes and observed that a population size of 20 works best for problems of type 9 or less. For problems of size 9 to 13, the population size increases as the size of the strategy space increases. We tried a range of population sizes in multiples of 10 before settling on a size that works best with a problem of a particular size.

3.6 Conclusion

We present an application of metaheuristic techniques for solving complex optimization problems in the field of combinatorial auctions and game theory. Tabu search and genetic algorithms prove to be a good approximate technique to solve the optimization problem of finding the best response strategy in combinatorial auctions. The limitation of this approach is that both tabu search and genetic algorithms are heuristic search techniques and are not guaranteed to find the optimal solution. No clean proof of convergence is known but our experiments show that the technique can be used to find good bidding strategies in cases where it is computationally expensive to find an optimal bidding strategy by enumeration. Our experiments show a significant reduction in the amount of search space that needs to be explored.

3.7 Future work

There are several issues for future research. Investigating the application of other search algorithms such as ant colonies, simulated annealing, scatter search and their hybrids. Analyzing the effect of the search process and performance gains by tuning the various genetic algorithm and tabu search parameters. We also plan to conduct further tests of the algorithms on other possibly more complex and larger combinatorial auction problem instances.

Chapter 4

Geometric Approach for Finding Nash Equilibria in Combinatorial Auctions

In this chapter, we present a geometric approach for finding Nash equilibrium of combinatorial auctions. In the previous two chapters, we formulated the problem of finding a Nash equilibrium as a search through the space of best responses. The search algorithms that we presented so far work on discrete strategy space. The techniques were based on the assumption that combinatorial auctions can be formulated as a normal form game and the algorithm requires that we run the auction to determine the payoff matrix. In this chapter, we present a technique to compute the Nash equilibrium without running the auction and without representing auctions as normal form games for every instance of a combinatorial auction problem. The proposed technique requires analyzing normal form game representations of combinatorial auction problems of a particular type and then determines a closed form solution. Once a general closed form solution of a particular problem size is determined then we do not need to construct a payoff matrix for every instance of a problem for finding a Nash equilibrium. Nash equilibria can be determined by setting the appropriate parameters of the closed form solution. The technique that we present in this chapter is based on an analytical approach and works on continuous strategy space. The analytical technique that we refer to as the geometric approach exploits the structure of the game and computes the Nash equilibrium. The advantages of the analytical approach is that we have a closed form solution and the technique can be computationally more efficient than a search on a discrete solution space. The drawback of the analytical approach is that currently we can apply it to a limited set of problems and it works for problems of smaller size whereas the search based approach is computationally intensive but can be applied to a problem of arbitrary size. We think that both the search based and the analytical approach are useful depending on the nature of the problem we are trying to solve. If we can take advantage of the structure of the game then it is certainly useful to have a closed form solution but in cases where there not much structure that can be exploited, the search based approach on discrete solution space can serve as a useful tool to compute the Nash equilibrium. The advantage of the discrete search is that we can apply it to arbitrary problems without understanding the outcome space. On the other hand, the geometric approach that we present in this chapter has the potential to give us exact solutions, but it requires a much deeper understanding of the combinatorial auctions in order to apply it. Once we have that, however, we don't need to solve a large number of instances to find the solution. We observed that in combinatorial auctions, the payoffs to bidders as a result of joint strategies have a structure that can be exploited in determining Nash equilibria.

Before introducing the geometric approach let us clarify what we mean by an analytical solution with the help of an example based on minimum-effort game. The minimum effort game [43] is an N-person game in which players are members of a group. Each player i chooses an effort level $e_i,\ i = 1, ..., n$. Each player's payoff equals the difference between the minimum effort level of any of the group members (including the player himself) and the cost of that player's own effort. This game has often been interpreted as modeling the team production problem in which the minimum effort level in a team is the key determinant of the team's output. There is an individual cost of per unit of effort. Thus the payoff is the minimum effort multiplied by a constant amount minus the cost of one's own effort. Equation 4.1 gives the individual's payoff as a function of his own choice as well as the choice of other players. Each player chooses an effort from the interval $[0, \overline{e}]$. In equation (4.1), we assume that $a > b$. Efforts are required to be greater than or equal to a specified

	(1)	**(2)**	**(3)**	**(4)**	**(5)**
(1)	**(5, 5)***	5, 4	5, 3	5, 2	5, 1
(2)	4, 5	**(10, 10)***	10, 9	10, 8	10, 7
(3)	3, 5	9, 10	**(15, 15)***	15, 14	15, 13
(4)	2, 5	8, 10	14, 15	**(20, 20)***	20, 19
(5)	1, 5	7, 10	13, 15	19, 20	**(25, 25)***

Table 4.1: Normal from representation of a minimum effort game.

lower bound, and to be less than or equal to an upper bound.

$$p_i(e_1, ..., e_n) \quad = \quad a * \min_{j=1,...,n} \{e_j\} - b * e_i, \; i = 1, ..., n \tag{4.1}$$

Table 4.1 is an example of a minimum-effort game represented in normal form. The normal form game of Table 4.1 has two players and the action space of each player is a discrete set of five effort levels. The payoff matrix is computed by using equation (4.1) and setting the values of $a = 6$ and $b = 1$. The game in Table 4.1 has five Nash equilibria in pure strategies. The pure-strategy Nash equilibria are symmetric and are Pareto-ranked. The uniform profile with every player using effort level 5 is the Pareto efficient one. The profile with every player playing effort level 1 is the worst Nash equilibrium in terms of the social welfare. Table 4.2 shows the payoffs of an individual player in an another example of a minimum-effort game with more than two players and with values of the constants a and b of the payoff equation 4.1 equal to 10 and 1, respectively. Table 4.2 shows the payoffs of any one player with respect to his own effort and the minimum effort of the group. The payoffs in Table 4.2 can be used to construct a game in normal form and the Nash equilibria can be easily determined. An interesting question is that is it possible to take advantage of the properties of the payoff function to determine a closed form solution. Once we have closed form solution then it can be applied to any game of the same type.

By examining the payoff matrix in Table 4.2, we observe that irrespective of the number of players and the cost of the minimum effort (as long as $a > b$ in equation 4.1), any common effort constitutes a Nash equilibrium. An important feature of the payoff function is that a unilateral increase in effort will not affect the minimum effort and results in a reduced payoff. A unilateral decrease reduces the minimum by more than the cost savings.

	(8)	**(7)**	**(6)**	**(5)**	**(4)**	**(3)**	**(2)**	**(1)**
(8)	**72***	62	52	42	32	22	12	2
(7)	-	**63***	53	43	33	23	13	3
(6)	-	-	**54***	44	34	24	14	4
(5)	-	-	-	**45***	35	25	15	5
(4)	-	-	-	-	**36***	26	16	6
(3)	-	-	-	-	-	**27***	17	7
(2)	-	-	-	-	-	-	**18***	8
(1)	-	-	-	-	-	-	-	**9***

Table 4.2: Payoffs of the row player with respect to his own effort and the minimum effort of the group in a minimum effort game.

The dilemma for an individual is that better outcomes require higher effort but entail more risk. The minimum effort game has a specific structure that can be identified and taken advantage of and a closed form solution can written without actually creating a normal form game and computing the payoffs for every strategy combination. Moreover, the set of equilibria is unaffected by changes in the number of participants, the size of the action space or the cost of the effort. The magnitude of the payoff to each player depends on factors like the effort cost, number of players and the action space but the best responses do not depend on these factors. The payoff structure of the minimum-effort game is such that it produces a continuum of pure-strategy Nash equilibria. These equilibria are Pareto-ranked because all individuals prefer the equilibrium with the highest effort levels for all.

The minimum effort game is a simple and well studied game and has an analytical solution to determine the Nash equilibria. Many other games, including multi-player prisoner's dilemma, traveling salesman, arms race, Cournot duopoly, first-price and second-price sealed-bid auctions have known analytical solutions [30]. The main contribution of this chapter is a technique to analytically determine the Nash equilibria of combinatorial auctions. In the following sections, we illustrate our technique using single item auctions and then extend it to combinatorial auctions.

4.1 Applying geometric approach to single item auctions

Consider a first-price sealed bid auction where the item is allocated to the highest bidder and the winner pays the price she bids. The first-price sealed bid auction can be

	(0)	**(1)**	**(2)**	**(3)**	**(4)**	**(5)**	**(6)**	**(7)**
(0)	10, 0	0, 6	0, 5	0, 4	0, 3	0, 2	0, 1	0, 0
(1)	9, 0	9, 0	0, 5	0, 4	0, 3	0, 2	0, 1	0, 0
(2)	8, 0	8, 0	8, 0	0, 4	0, 3	0, 2	0, 1	0, 0
(3)	7, 0	7, 0	7, 0	7, 0	0, 3	0, 2	0, 1	0, 0
(4)	6, 0	6, 0	6, 0	6, 0	6, 0	0, 2	0, 1	0, 0
(5)	5, 0	5, 0	5, 0	5, 0	5, 0	5, 0	0, 1	0, 0
(6)	4, 0	4, 0	4, 0	4, 0	4, 0	4, 0	$\mathbf{(4, 0)}^*$	0, 0
(7)	3, 0	3, 0	3, 0	3, 0	3, 0	3, 0	3, 0	$\mathbf{(3, 0)}^*$
(8)	2, 0	2, 0	2, 0	2, 0	2, 0	2, 0	2, 0	2, 0
(9)	1, 0	1, 0	1, 0	1, 0	1, 0	1, 0	1, 0	1, 0
(10)	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0

Table 4.3: Normal from representation of First-Price Sealed-Bid Auction. The valuation of bidder is 10 and the valuation of bidder 2 is 7. Bidder 1 is the row player and Bidder 2 is the column player.

formulated as a normal form game in which the n bidders represent the n players. The set of possible actions of each player is the set of possible bids. The payoff of player i is $v_i - b_i$ if b_i is higher than every other bid. Let us assume that if there is a tie for the highest bid then the bidder having a lower index wins. For example if there is a tie between bidder 2 and bidder 5 then the item is allocated to bidder 2. The payoff for all other players who are not winner is 0. In our model, we assume that bidders won't bid more than their true valuations. Table 4.3 is the payoff matrix for a 2 player first-price sealed-bid auction problem where $v_1 = 10$ and $v_2 = 7$ and the bidders can submit bids in increments of 1, starting from 0, up to their valuations.

Every possible combination of strategies (one strategy from each bidder) and the corresponding payoffs to each bidder represents an outcome of the auction. In our example, the strategy space for each bidder has a single dimension, and the outcome space is two dimensional. There are 70 possible outcomes and we divide the space of the outcomes into two regions. Region R_1 includes outcomes where the item is allocated to bidder 1 (i.e, outcomes which satisfies the conditions of $b_1 \geq b_2$). The surplus to bidder 1 in region R_1 is $v_1 - b_1$ and the surplus to bidder 2 is 0. Similarly, region R_2 includes outcomes where the item is allocated to bidder 2 (i.e, outcomes which satisfies the conditions of $b_2 > b_1$). For any point within the region R_2, the surplus for bidder 2 is $v_2 - b_2$ and the surplus for bidder 1 is 0. Let the space of outcomes be denoted by Θ. The regions are mutually exclusive and exhaustive and $R_1 \cup R_2 = \Theta$. As shown in Figure 4.1, the outcome space is separated into regions by the line $b_1 = b_2$. The regions plotted on panel (a) of two dimensional Figure of 4.1 follows the same pattern irrespective of the size of the strategy space and the valuations of the bidder. The regions will exhibit a similar pattern for more than two bidders and for continuous strategy space. Panel (b) of Figure 4.1, shows the best response strategies of both the bidders. Best response of bidder 1 is to always bid b_2. Best response of bidder 2 is to outbid bidder 1 by the minimum amount permitted by the strategy space up to his valuations. Let $\beta_i(b_{-i})$ denote the best responses of bidder i as a function of the all other agent's strategy. Equations 4.2, 4.3 and 4.4 is the best response function of the two bidders.

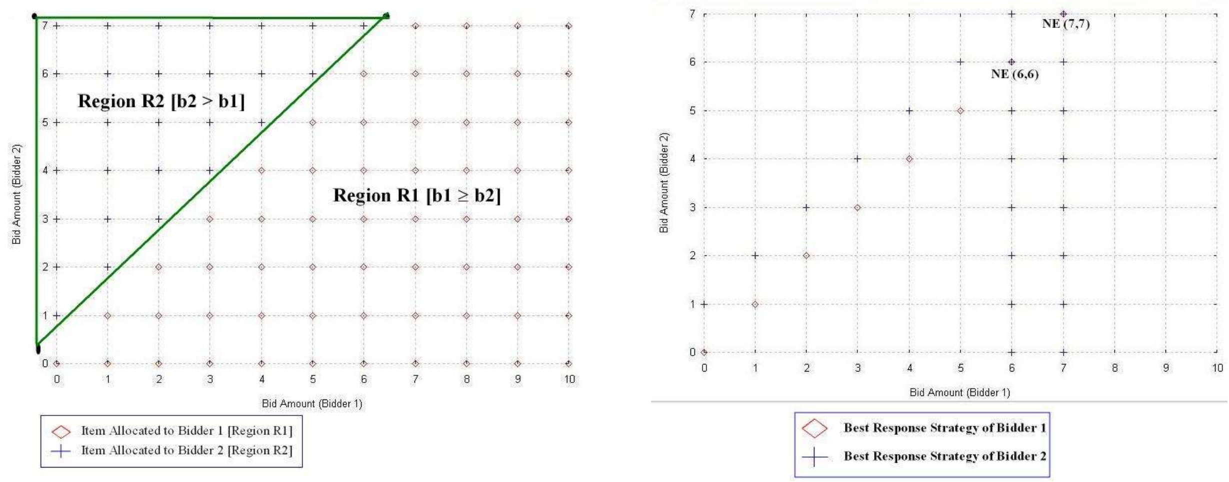
$$\beta_1(b_1) \quad = \quad b_2 \tag{4.2}$$

$$\beta_2(b_2) \quad = \quad b_1 \, + \, 1 \;\; if \;\; b_1 \; \leq \; (v_2 - 1) \tag{4.3}$$

$$\beta_2(b_2) \quad = \quad [0, v_2] \;\; if \;\; b_1 \; \geq \; v_2 \tag{4.4}$$

As we can see from equations 4.2, 4.3 and 4.4, the best response functions of bidders participating in a first-price sealed-bid auction can be written in a closed form. Once the best response functions are written in a closed form, then they can be plotted and visualized or can be solved as a system of simultaneous equations to determine the Nash equilibrium. This leads us to study best response functions of bidders in combinatorial auctions. It is a well understood that the intersection of best response functions results in Nash equilibria. In this chapter, we present preliminary results of deriving closed form best response functions of bidders in combinatorial auctions. We also propose a novel technique which we refer to as the geometric approach to determine the Nash equilibrium. Following is a description of the technique with the help of the same example of first price sealed bid auction.

Figure 4.2 is a plot of the utility function of each bidder with payoffs defined in Table 4.3. The x-axis represents the bids of bidder 1 and the y-axis represents the bids of bidder 2. The z-axis represents the utility. It is evident from Figure 4.2 that the utility functions are not random and have a structure that can be explained. The utility function of bidder 1 in region $R1$ is the equation of a plane with a constant slope, and the utility of bidder 1 is 0 in region $R2$. The utility of bidder 1 is highest at the xy-coordinates $(0, 0)$ and it gradually decreases as we traverse the boundaries from $(0, 0)$ to $(10, 0)$ and $(0, 0)$ to $(7, 7)$. The utility function of bidder 2 is also a plane with a constant slope. The utility of bidder 2 is highest at the xy-coordinates $(0, 1)$ and it gradually decreases as we traverse from $(0, 1)$



(a) (b)

Figure 4.1: Panel (a): Regions in the outcome space of first price sealed bid auction with 2 bidders. , Panel (b): Best response strategies of the bidders and Nash equilibria as a result of the intersection of the best response strategies.

to $(0, 7)$ and $(0, 1)$ to $(6, 7)$. Figure 4.3 illustrates how we can exploit the structure of the utility functions to determine the Nash equilibrium. For all the outcomes that fall in the region $R2$, the item is allocated to bidder 2. From any point within the region $R2$, bidder 1 can move to region $R1$. With reference to Figure 4.3, bidder 2 can move horizontally by changing its bid amount and bidder 1 can move vertically by changing the value of its bid $b1$. The surplus of bidder 1 in region $R2$ is 0 and the surplus for bidder 1 in region $R1$ is always greater than 0. Since, bidder 1 can always move from $R2$ to $R1$ and can be better off, any outcome within region $R2$ will never be a Nash equilibrium. In this way, we have identified a portion of the outcome space that will never contain a outcome constituting a Nash equilibrium. Region $R1$ can be further partitioned into two regions $R1-L$ and $R1-R$, such that $R1-L \cup R1-R = R1$. The property of region $R1-L$ is that bidder 2 can move from any point within the region $R1-L$ to region $R2$ by increasing its bid amount and improve its surplus. Using the basic definition of Nash equilibrium, the set of strategies and the corresponding payoffs in region $R1-L$ do not constitute the Nash Equilibrium because bidder 2 can benefit by changing its strategy while bidder 1 keeps its strategies unchanged. Since $v_2 < v_1$, in $R1-R$, bidder 2 cannot move vertically by increasing its bid to obtain a

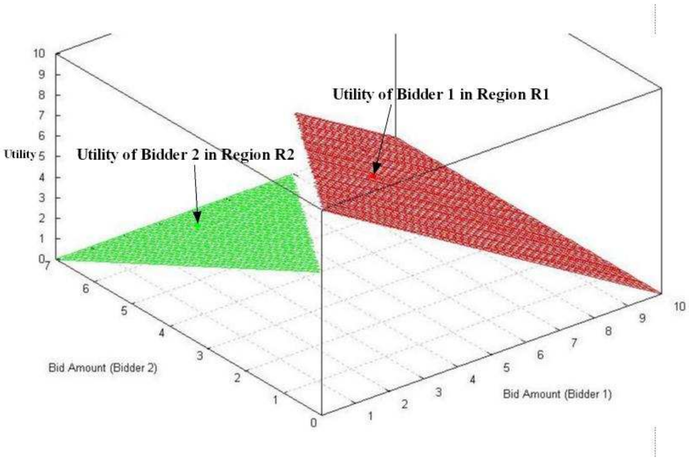


Figure 4.2: Utility function of the bidders for first-price sealed-bid auction.

positive surplus. As shown in Figure 4.2, the utility of bidder 1 in region $R1$ is such that, keeping b_1 constant, bidder 2 gets the maximum surplus at the boundary line separating the two regions. The line $b_1 = b_2$ dominates any point in region $R1$ with respect to bidder $1's$ surplus. Due to the specific structure of the utility function of bidder 1, only outcomes at the boundary line can be a Nash equilibrium. The result of the analysis leads us to the conclusion that there are two Nash equilibria in our example: either both agents bid 6 or both bid 7.

Based on our analysis, we can provide an analytical solution to the Nash equilibrium of first price sealed bid auction with two bidders. Let $v_1 < v_2$ and let δ denote the resolution of the discrete strategy space. The size of the strategy space for each bidder i is $\frac{v_i}{\delta}\ + 1$. There will be two Nash equilibria for the game with strategy profile (v_2, v_2) and $(v_2 - \delta, v_2 - \delta)$. We can extend the same analysis to multiple bidders and continuous strategy space. In all equilibria, the item is allocated to the bidder who values it most highly. Assuming that there are n players and the player's valuations of the object are all different and all positive. For convenience, we number the players 1 through n in such a way that $v_1 > v_2 > ...v_n > 0$. Applying the geometric approach allows us to conclude $(v_2, v_2, v_3, ..., v_n)$ to be a Nash equilibrium.

Consider another example of a second-price sealed-bid auction where the item is allocated to the highest bidder and the winner pays the price equal to the second highest

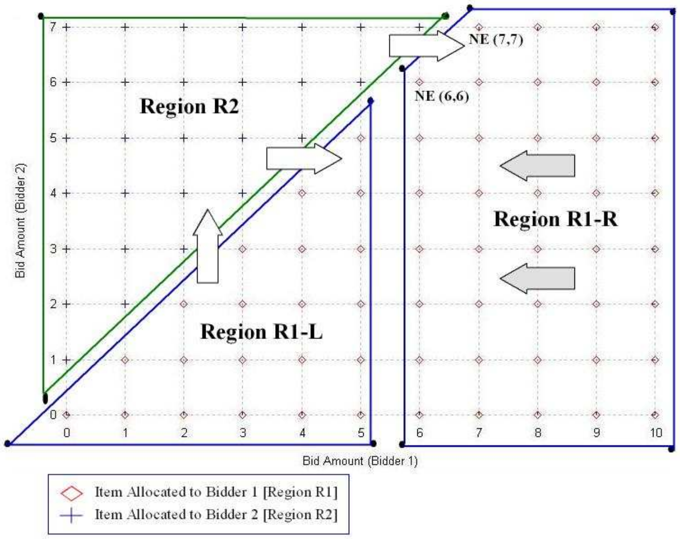
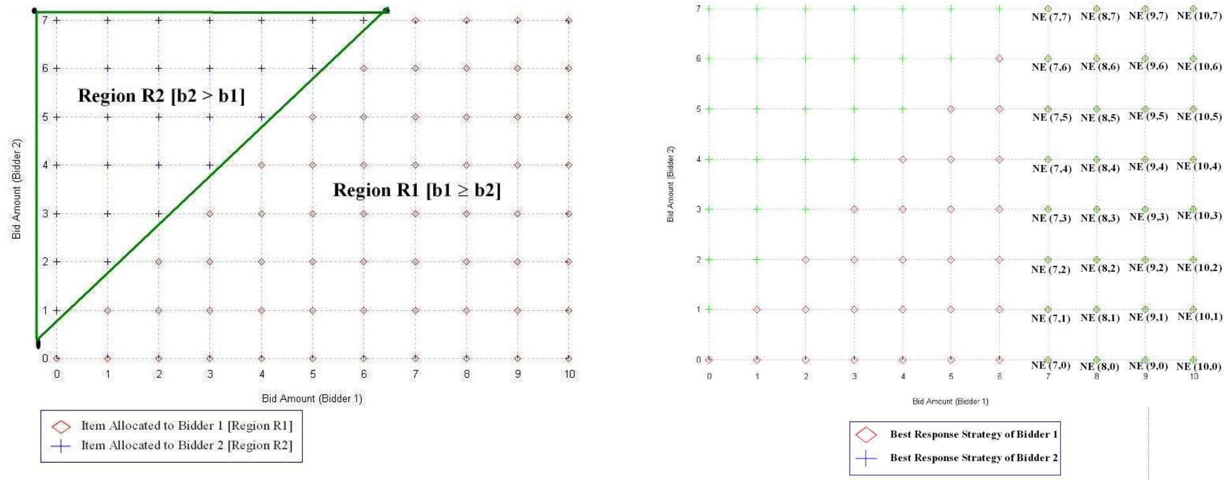


Figure 4.3: Results of applying geometric approach to the problem of first-price sealed-bid auction for computing Nash equilibria.

	(0)	**(1)**	**(2)**	**(3)**	**(4)**	**(5)**	**(6)**	**(7)**
(0)	10, 0	0, 7	0, 7	0, 7	0, 7	0, 7	0, 7	0, 7
(1)	10, 0	9, 0	0, 6	0, 6	0, 6	0, 6	0, 6	0, 6
(2)	10, 0	9, 0	8, 0	0, 5	0, 5	0, 5	0, 5	0, 5
(3)	10, 0	9, 0	8, 0	7, 0	0, 4	0, 4	0, 4	0, 4
(4)	10, 0	9, 0	8, 0	7, 0	6, 0	0, 3	0, 3	0, 3
(5)	10, 0	9, 0	8, 0	7, 0	6, 0	5, 0	0, 2	0, 2
(6)	10, 0	9, 0	8, 0	7, 0	6, 0	5, 0	4, 0	0, 1
(7)	$\mathbf{(10,0)}^*$	$\mathbf{(9,0)}^*$	$\mathbf{(8,0)}^*$	$\mathbf{(7,0)}^*$	$\mathbf{(6,0)}^*$	$\mathbf{(5,0)}^*$	$\mathbf{(4,0)}^*$	$\mathbf{(3,0)}^*$
(8)	$\mathbf{(10,0)}^*$	$\mathbf{(9,0)}^*$	$\mathbf{(8,0)}^*$	$\mathbf{(7,0)}^*$	$\mathbf{(6,0)}^*$	$\mathbf{(5,0)}^*$	$\mathbf{(4,0)}^*$	$\mathbf{(3,0)}^*$
(9)	$\mathbf{(10,0)}^*$	$\mathbf{(9,0)}^*$	$\mathbf{(8,0)}^*$	$\mathbf{(7,0)}^*$	$\mathbf{(6,0)}^*$	$\mathbf{(5,0)}^*$	$\mathbf{(4,0)}^*$	$\mathbf{(3,0)}^*$
(10)	$\mathbf{(10,0)}^*$	$\mathbf{(9,0)}^*$	$\mathbf{(8,0)}^*$	$\mathbf{(7,0)}^*$	$\mathbf{(6,0)}^*$	$\mathbf{(5,0)}^*$	$\mathbf{(4,0)}^*$	$\mathbf{(3,0)}^*$

Table 4.4: Normal from representation of second-price sealed-bid auction. The valuation of bidder is 10 and the valuation of bidder 2 is 7. Bidder 1 is the row player and Bidder 2 is the column player.



(a) (b)

Figure 4.4: Panel (a): Regions in the outcome space of second price sealed bid auction with 2 bidders. , Panel (b): Best response strategies of the bidders and Nash equilibria as a result of the intersection of the best response strategies.

bid. The second-price sealed bid auction can be formulated as a normal form game in which the n bidders represent the n players. The set of actions, or strategies, of each player is the set of possible bids. Each player's valuations of the object is represented by v_i and each player i submits a sealed bid b_i. If player i's bid is higher that every other, she obtains the object at a price equal to the second highest bid, say b_j, and hence receives the payoff $v_i - b_j$. We assume the same tie breaking rule (i.e, if there is a tie for the highest bid then the bidder having a lower index wins). The payoff for all other players who are not winner is 0. Table 4.4 is the payoff matrix for a 2 player first-price sealed-bid auction problem where $v_1 = 10$ and $v_2 = 7$ and the bidders can submit bids in increments of 1 starting from 0 up to their valuations.

Panel (a) of Figure 4.4 illustrates the two regions based on the whether the item is allocated to bidder 1 or bidder 2. For all the outcomes in region $R1$, the item is allocated to bidder 1 and the surplus of bidder 1 is $v_1 - b_2$. Similarly, for all the outcomes in region $R2$, the item is allocated to bidder 2 and the surplus of bidder 2 is $v_2 - b_1$, whereas the surplus of bidder 1 is 0. Panel (b) of Figure 4.4 is a plot of the best response functions of the

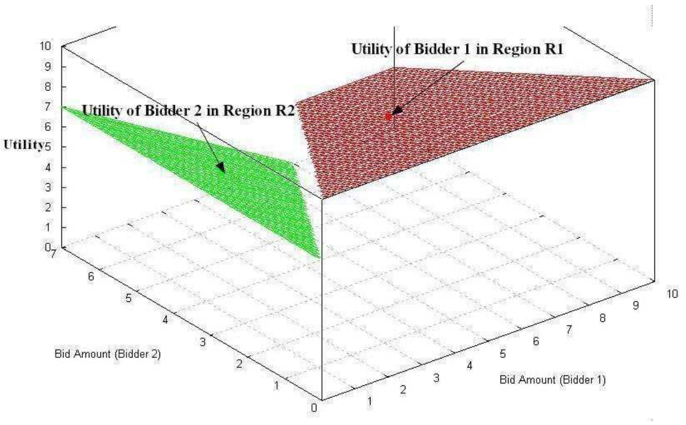


Figure 4.5: Utility function of the bidders for second-price sealed-bid auction.

bidders and the Nash equilibria as a result of the intersections of the best response functions.

Figure 4.5 is a plot of the utility function of the bidders and Figure 4.6 shows the results of applying the geometric approach for computing Nash equilibria. Bidder 1 can unilaterally move from region $R2$ to region $R1$ by increasing its bid and hence any outcome in region $R2$ is not a Nash equilibria. Region $R1$ is further partitioned into region $R1 - L$ and $R1 - R$ such that $R1 - L \cup R1 - R = R1$. Bidder 2 can unilaterally move from region $R1 - L$ to region $R2$ by increasing its bid and hence any outcome in region $R1 - L$ is not a Nash equilibria. Regions $R2$ and $R1 - L$ are ruled out as they do not satisfy the conditions of Nash equilibrium. For any outcome in the region $R1 - R$, bidder 2 can not unilaterally move to a different region because all bids by bidder 1 in region $R1 - R$ are greater than or equal to the valuation of bidder 2. Within region $R1 - R$ and keeping the strategy of bidder 2 fixed, all strategies for bidder 1 results in the same surplus. Hence, every outcome within the region $R1 - R$ is a Nash equilibrium. The technique can be applied to multiple players and with a continuous bidding strategy. The game has many Nash equilibria and one of which is that each player bid is equal to her valuation of the object.

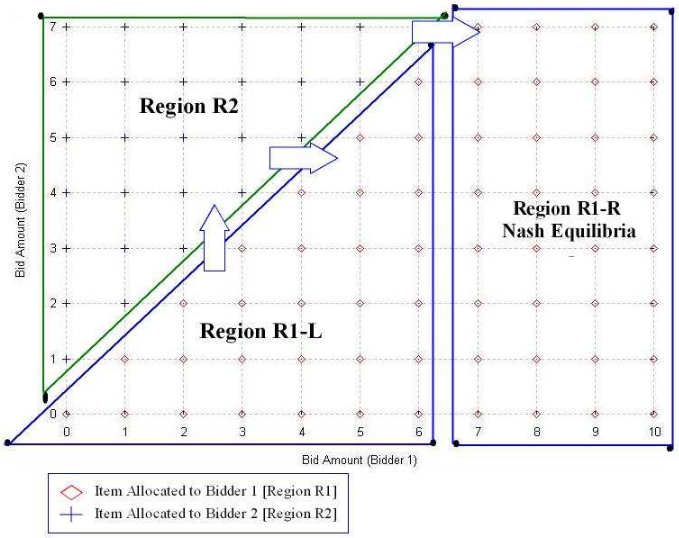


Figure 4.6: Results of applying geometric approach to the problem of second-price sealed-bid auction for computing Nash equilibria.

4.2 2-agent 2-item combinatorial auctions

In the previous section, we applied the geometric approach to determine pure strategy Nash equilibria of single item first-price and second-price sealed bid auctions. In this section, we apply the geometric approach for determining Nash equilibrium for a general 2-agent 2-item combinatorial auction problem. In the later sections of this chapter, we present problems with more than two agents and two items. We model combinatorial auctions as multi-player complete information game and assume that the bidders participating in the auction follow a myopic best-response bidding strategy. As in the previous chapters, we follow the model in which a bid vector representing the bidder's maximum willingness to pay over all the possible combination of items submitted to the proxy agent constitutes a strategy. We adopt an approach that takes advantage of the patterns and structure of the payoff matrix. The approach partitions the space of outcomes into regions and we derive equations for the utility of bidders in each region and the conditions under which bidders have no incentive to deviate from one region to another. We solve a general two agent and two item combinatorial auction problem using the technique and present the results for one instantiation of the problem.

For a 2-agent and 2-item problem, there are four possible feasible allocations. The

	Allocations			
	A	**B**	**Regions**	**Utility**
R_{11}	Agent 1	Agent 1	$b_1(AB) \geq b_1(A) + b_2(B)$ $b_1(AB) \geq b_1(B) + b_2(A)$ $b_1(AB) \geq b_2(AB)$	$u_1(R_{11}) = v_1(AB) - b_2(AB)$ $u_2(R_{22}) = 0$
R_{12}	Agent 1	Agent 2	$b_1(A) + b_2(B) \geq b_1(B) + b_2(A)$ $b_1(A) + b_2(B) \geq b_1(AB)$ $b_1(A) + b_2(B) \geq b_2(AB)$	$u_1(R_{12}) = v_1(A) - [b_2(AB) - b_2(B)]$ $u_2(R_{12}) = v_2(B) - [b_1(AB) - b_1(A)]$
R_{21}	Agent 2	Agent 1	$b_1(B) + b_2(A) \geq b_1(A) + b_2(B)$ $b_1(B) + b_2(A) \geq b_1(AB)$ $b_1(B) + b_2(A) \geq b_2(AB)$	$u_1(R_{21}) = v_1(B) - [b_2(AB) - b_2(A)]$ $u_2(R_{21}) = v_1(A) - [b_1(AB) - b_1(B)]$
R_{22}	Agent 2	Agent 2	$b_2(AB) \geq b_1(A) + b_2(B)$ $b_2(AB) \geq b_1(B) + b_2(A)$ $b_2(AB) \geq b_1(AB)$	$u_1(R_{22}) = 0$ $u_2(R_{22}) = v_2(AB) - b_1(AB)$

Table 4.5: Division of the outcome space for 2-agent and 2-item problem. The four regions corresponds to the four possible allocations.

four feasible allocations are: $(AB \rightarrow Agent\,1)$, $(AB \rightarrow Agent\,2)$, $(A \rightarrow Agent\,1,\ B \rightarrow Agent\,2)$ and $(A \rightarrow Agent\,2,\ B \rightarrow Agent\,1)$ and hence there are 4 base regions. The strategy space for each bidder is three dimensional (one dimension for each of the bundle excluding the null bundle). Since there are two bidders and 3 bundles , the space of outcomes is six dimensional. Each axis of the outcome space ranges from 0 to the maximum valuations of the bidder for the bundle corresponding the the axis. For example, in the 2-agent 2 -item problem, the six axis of the 6-dimensional outcome space are $b_1(A),\ b_1(B),\ b_1(AB), b_2(A),\ b_2(B) and\, b_2(AB)$. The value of the axis $b_1(A)$ ranges from 0 to $v_1(A)$ and similarly the value of the other axis also range from 0 to the agent's valuations for the bundle.

Table 4.5 illustrates the division of the 6-dimensional outcome space for 2-agent and 2-item problem. The four regions correspond to the four possible allocations. We adopt a notation where R is denoted by the region and the subscript of R is a n-digit number. The n in the subscript of R corresponds to the set of n items. The value of each digit in the n-digit number corresponds to the agent id to which the item is allocated. For example a region R_{12} means the set of outcomes where item A is allocated to agent 1 and item B is allocated to agent 2. For a 3-Agent 3-Item problem region R_{213} signifies item A is allocated agent 2, item B to agent 1 and item C to agent 3. Table 4.5 shows the equations for the utility of each agent within every region for the Ascending Package Auction [2]. We derived the equations for the utility within every region by observing the outcome of several runs of the auction with varying input (valuations). The equations for the utility functions in Table 4.5 holds true as long as the values of the agents satisfy the free disposal property. Free disposal means that the value of union of any two bundles is greater than or equal to the value of any component bundles. This is standard assumption in the literature on combinatorial auctions. An interesting and useful phenomenon we observed is that the utility of an agent in a region can be written in a closed form and is dependent on a small set of variables. For instance, the utility of agent 1 in region R_{11} is $v_1(AB) - b_2(AB)$ and is only dependent on $b_2(AB)$ and not on other variables like $b_1(A)$ or $b_1(B)$. One of the fundamental assumption for our geometric approach technique to work is to be able to write closed form utility functions of agents within a region.

The next step is to determine the change in utility of an agent if it moves from one

Source	**Destination**	**Change in utility**	**Constraints**
R_{11}	R_{12}	$\triangle u_1(R_{11} \rightarrow R_{12}) = b_2(B) - [v_1(AB) - v_1(A)]$	$[v_1(AB) - v_1(A)] \geq b_2(B)$
R_{11}	R_{21}	$\triangle u_1(R_{11} \rightarrow R_{21}) = b_2(A) - [v_1(AB) - v_1(B)]$	$[v_1(AB) - v_1(B)] \geq b_2(A)$
R_{11}	R_{22}	$\triangle u_1(R_{11} \rightarrow R_{22}) = b_2(AB) - v_1(AB)$	$v_1(AB) \geq b_2(AB)$
R_{12}	R_{11}	$\triangle u_1(R_{12} \rightarrow R_{11}) = [v_1(AB) - v_1(A)] - b_2(B)$	$b_2(B) \geq [v_1(AB) - v_1(A)]$
R_{12}	R_{21}	$\triangle u_1(R_{12} \rightarrow R_{21}) = [v_1(B) - v_1(A)] - [b_2(B) - b_2(A)]$	$[b_2(B) - b_2(A)] \geq [v_1(B) - v_1(A)]$
R_{12}	R_{22}	$\triangle u_1(R_{12} \rightarrow R_{22}) = [b_2(AB) - b_2(B)] - v_1(A)$	$v_1(A) \geq [b_2(AB) - b_2(B)]$

Table 4.6: Change in the utility of Agent 1 when it moves from one region to another

Source	Destination	Change in utility	Constraints
R_{21}	R_{11}	$\triangle u_1(R_{21} \rightarrow R_{11}) = [v_1(AB) - v_1(B)] - b_2(A)$	$b_2(A) \geq [v_1(AB) - v_1(B)]$
R_{21}	R_{12}	$\triangle u_1(R_{21} \rightarrow R_{12}) = [v_1(A) - v_1(B)] - [b_2(A) - b_2(B)]$	$[b_2(A) - b_2(B)] \geq [v_1(A) - v_1(B)]$
R_{21}	R_{22}	$\triangle u_1(R_{21} \rightarrow R_{22}) = [b_2(AB) - b_2(A)] - v_1(B)$	$v_1(B) \geq [b_2(AB) - b_2(A)]$
R_{22}	R_{11}	$\triangle u_1(R_{22} \rightarrow R_{11}) = v_1(AB) - b_2(AB)$	$b_2(AB) \geq v_1(AB)$
R_{22}	R_{12}	$\triangle u_1(R_{22} \rightarrow R_{12}) = v_1(A) - [b_2(AB) - b_2(B)]$	$[b_2(AB) - b_2(B)] \geq v_1(A)$
R_{22}	R_{21}	$\triangle u_1(R_{22} \rightarrow R_{21}) = v_1(B) - [b_2(AB) - b_2(A)]$	$[b_2(AB) - b_2(A)] \geq v_1(B)$

Table 4.7: Change in the utility of Agent 1 when it moves from one region to another

region to another. Keeping the bids of all other agents the same, an agent can change the auction allocation by changing its proxy bid. For example, agent 1 can move from region R_{11} to region R_{22} by bidding lower on $b_1(AB)$ such that the condition $b_1(AB) < b_2(AB)$ becomes true. For a problem with 2-agents and 2-items there are 6 dimensions in the outcome space and any one agent has 3 degrees of freedom to transition from one region to another. Utilizing the utility equations within every region from Table 4.5 we can find out the change in utility of an agent as it moves from one region to the other. Tables 4.6, 4.7, 4.8 and 4.9 enumerate all the possible transitions of each agent and the change in the utility. For every transition, Tables 4.6, 4.7, 4.8 and 4.9 also lists the constraints under which the change in utility is positive. For example, agent 1 will not be better off by moving from region R_{11} to R_{12} if $[v_1(AB) - v_1(A)] \geq b_2(B)$. If the condition $[v_1(AB) - v_1(A)] \geq b_2(B)$ is true then agent 1 does not have an incentive to unilaterally deviate from region R_{11} to region R_{12}. Once the utility functions of agents for each region are known then it is straight forward to compute the change in utility for any transition. The change in utility for agent i from a source region R_S to the destination region R_D denoted by $\triangle u_i(R_S \rightarrow R_D)$ is simply the difference of $u_i(R_D)$ and $u_i(R_S)$. It is also straight forward to compute the constraints under which the agents have or do not have an incentive to deviate from one region to another. The constraints under which agent i has an incentive to move from region (R_S) to (R_D) can be obtained by solving the equation $u_i(R_D) - u_i(R_S) \geq 0$. Tables 4.6, 4.7, 4.8 and 4.9 enumerate all the constraints for a general 2-agent 2-item problem.

For a 2-agent 2-item problem there are four possible regions. From any single region, there are three possible regions for each agent to move to. By combining all the constraints from Table 4.6 and 4.7 and a similar table for the other agents, we can determine all the Nash equilibria. Following the definition of Nash equilibrium, we determine a subset of the outcomes of a region where no agent has an incentive to unilaterally deviate. For example, the set of Nash equilibria in the region R_{11} contains outcomes that satisfies the six constraints: $u_1(R_{12}) - u_1(R_{11}) \geq 0$, $u_1(R_{21}) - u_1(R_{11}) \geq 0$, $u_1(R_{22}) - u_1(R_{11}) \geq 0$, $u_2(R_{12}) - u_2(R_{11}) \geq 0$, $u_2(R_{21}) - u_2(R_{11}) \geq 0$ and $u_2(R_{22}) - u_2(R_{11}) \geq 0$. Table 4.10 shows the results of applying the geometric approach to a general 2-Agent 2-Item problem. Nash equilibria of a 2-Agent 2-Item problem is the set $N_{11} \cup N_{12} \cup N_{21} \cup N_{22}$.

Following is a high level description of the technique in two steps.

Step 1 : The algorithm begins by first identifying all the possible feasible allocations for n

Source	**Destination**	**Change in utility**	**Constraints**
R_{11}	R_{12}	$\triangle u_2(R_{11} \rightarrow R_{12}) = v_2(B) - [b_1(AB) - b_1(A)]$	$[b_1(AB) - b_1(A)] \geq v_2(B)$
R_{11}	R_{21}	$\triangle u_2(R_{11} \rightarrow R_{21}) = v_2(A) - [b_1(AB) - b_1(B)]$	$[b_1(AB) - b_1(B)] \geq v_2(A)$
R_{11}	R_{22}	$\triangle u_2(R_{11} \rightarrow R_{22}) = v_2(AB) - b_1(AB)$	$b_1(AB) \geq v_2(AB)$
R_{12}	R_{11}	$\triangle u_2(R_{12} \rightarrow R_{11}) = [b_1(AB) - b_1(A)] - v_2(B)$	$v_2(B) \geq [b_1(AB) - b_1(A)]$
R_{12}	R_{21}	$\triangle u_2(R_{12} \rightarrow R_{21}) = [v_2(A) - v_2(B)] - [b_1(A) - b_1(B)]$	$[b_1(A) - b_1(B)] \geq [v_2(A) - v_2(B)]$
R_{12}	R_{22}	$\triangle u_2(R_{12} \rightarrow R_{22}) = [v_2(AB) - v_2(B)] - b_1(A)$	$b_1(A) \geq [v_2(AB) - v_2(B)]$

Table 4.8: Change in the utility of Agent 2 if it moves from one region to another

Source	Destination	Change in utility	Constraints
R_{21}	R_{11}	$\triangle u_2(R_{21} \rightarrow R_{11}) = [b_1(AB) - b_1(B)] - v_2(A)$	$v_2(A) \geq [b_1(AB) - b_1(B)]$
R_{21}	R_{12}	$\triangle u_2(R_{21} \rightarrow R_{12}) = [v_2(B) - v_2(A)] - [b_1(B) - b_1(A)]$	$[b_1(B) - b_1(A)] \geq [v_2(B) - v_2(A)]$
R_{21}	R_{22}	$\triangle u_2(R_{21} \rightarrow R_{22}) = [v_2(AB) - v_2(A)] - b_1(B)$	$b_1(B) \geq [v_2(AB) - v_2(A)]$
R_{22}	R_{11}	$\triangle u_2(R_{22} \rightarrow R_{11}) = b_1(AB) - v_2(AB)$	$v_2(AB) \geq b_1(AB)$
R_{22}	R_{12}	$\triangle u_2(R_{22} \rightarrow R_{12}) = b_1(A) - [v_2(AB) - v_2(B)]$	$[v_2(AB) - v_2(B)] \geq b_1(A)$
R_{22}	R_{21}	$\triangle u_2(R_{22} \rightarrow R_{21}) = b_1(B) - [v_2(AB) - v_2(A)]$	$[v_2(AB) - v_2(A)] \geq b_1(B)$

Table 4.9: Change in the utility of Agent 2 if it moves from one region to another

agents and m items. For a problem with n agents and m items, the space of outcome is d dimensional where $d = n(2^m - 1)$. Each agent's bid for a bundle represents one axis in the d-dimensional space. The range for each axis is from 0 to the value of the agent for that bundle. Each feasible allocation represents a region in the d-dimensional space and contains the set of bids that results in that particular allocation. Within each region, the sum of the bids for the allocated item is more than the sum of bids for any other feasible allocations. The regions are mutually exclusive and exhaustive. After the regions are identified, the next task is to determine the equations for prices of the bundles which result in a particular allocation within each region. This involves analytically deriving the market clearing bundle prices. Once the price for each winning bundle is determined the utility of each agent can be determined in a straightforward manner. An agent's utility is simply its value for the allocation f minus the price it pays for the bundle it receives as part of the allocation i.e $u_i(f) = v_i(f) - p_i(f)$.

Step 2:Once the space of outcomes is divided into regions and the utility of each agent within a region is computed, the next step is to compute the change in the utility of each agent as the agent moves from region to the other. Also, determine the conditions under which the agent has no incentive to deviate from one region to the other. Finally, determine the Nash Equilibria as the subset of each region by simplifying the equations obtained from the first two steps.

4.2.1 Example problem with 2 Agents and 2 Items

Using geometric approach, we provided a solution to a general 2-agent and 2-item problem. Once we have the solution, it can be applied to any instantiation of a 2-agent 2-item problem. Table 4.11 shows a combinatorial auction problem with two agents and two items. After simplifying the results obtained for a general 2-agent 2-item problem from Table 4.10, we can directly get the solution for the problem defined in Table 4.11. The Nash equilibria for the problem in Table 4.11 is listed in Table 4.12.

For the combinatorial auction problem defined in Table 4.11, any point in the region R_{21} can not be a Nash Equilibrium because the constraint $b_2(A) \geq [v_1(AB) - v_1(B)]$ defined in Table 4.6 and 4.7 for the region R_{21} can never be true. The value of $v_1(AB) - v_1(B)$ is 30 and the maximum value of $b_2(A)$ is $v_2(A)$ which is equal to 10. Agent 1 can always deviate from region R_{21} to R_{11} and increase its utility. Another reason

Nash	**Agent 1 Constraints**	**Agent 2 Constraints**
$N_{11} \subseteq R_{11}$	$[v_1(AB) - v_1(A)] \geq b_2(B)$ $[v_1(AB) - v_1(B)] \geq b_2(A)$ $v_1(AB) \geq b_2(AB)$	$[b_1(AB) - b_1(A)] \geq v_2(B)$ $[b_1(AB) - b_1(B)] \geq v_2(A)$ $b_1(AB) \geq v_2(AB)$
$N_{12} \subseteq R_{12}$	$b_2(B) \geq [v_1(AB) - v_1(A)]$ $[b_2(B) - b_2(A)] \geq [v_1(B) - v_1(A)]$ $v_1(A) \geq [b_2(AB) - b_2(B)]$	$v_2(B) \geq [b_1(AB) - b_1(A)]$ $[b_1(A) - b_1(B)] \geq [v_2(A) - v_2(B)]$ $b_1(A) \geq [v_2(AB) - v_2(B)]$
$N_{21} \subseteq R_{21}$	$b_2(A) \geq [v_1(AB) - v_1(B)]$ $[b_2(A) - b_2(B)] \geq [v_1(A) - v_1(B)]$ $v_1(B) \geq [b_2(AB) - b_2(A)]$	$v_2(A) \geq [b_1(AB) - b_1(B)]$ $[b_1(B) - b_1(A)] \geq [v_2(B) - v_2(A)]$ $b_1(B) \geq [v_2(AB) - v_2(A)]$
$N_{22} \subseteq R_{22}$	$b_2(AB) \geq v_1(AB)$ $[b_2(AB) - b_2(B)] \geq v_1(A)$ $[b_2(AB) - b_2(A)] \geq v_1(B)$	$v_2(AB) \geq b_1(AB)$ $[v_2(AB) - v_2(B)] \geq b_1(A)$ $[v_2(AB) - v_2(A)] \geq b_1(B)$

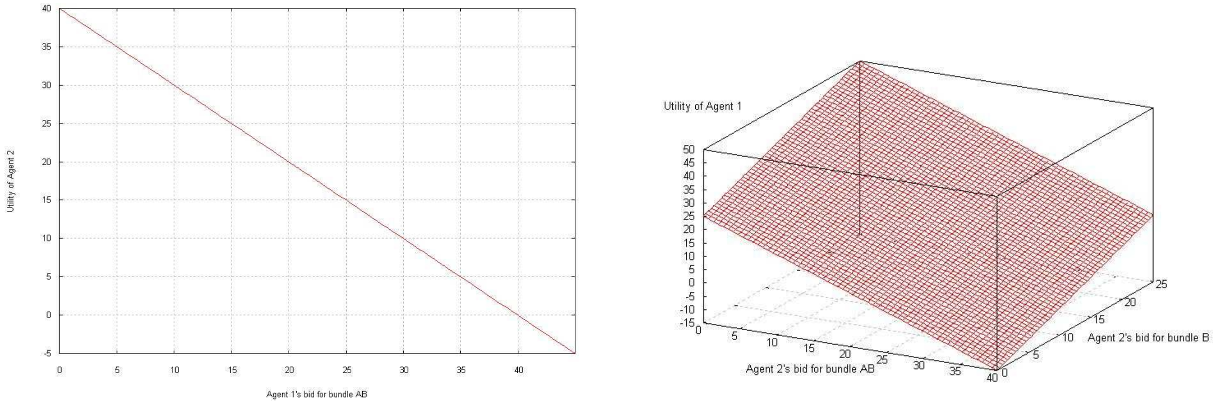
Table 4.10: Nash equilibria in 2-Agent 2-Item problem

	A	**B**	**AB**
Agent 1	25	15	45
Agent 2	10	25	40

Table 4.11: Example problem where both the agents have complementary preferences

Nash	**Region**
N_{11}	$0 \leq b_1(A) \leq b_1(AB) - 25$ $0 \leq b_1(B) \leq 15$ $40 \leq b_1(AB) \leq 45$ $0 \leq b_2(A) \leq 10$ $0 \leq b_2(B) \leq 20$ $0 \leq b_2(AB) \leq 40$
N_{12}	$15 \leq b_1(A) \leq 25$ $0 \leq b_1(B) \leq 15$ $35 \leq b_1(AB) \leq b_1(A) + b_2(B)$ $0 \leq b_2(A) \leq 10$ $20 \leq b_2(B) \leq 25$ $35 \leq b_2(AB) \leq b_1(A) + b_2(B)$
N_{21}	ϕ
N_{22}	ϕ

Table 4.12: Nash Equilibria for problem defined in Table 4.11



(a) (b)

Figure 4.7: Panel (a): Utility of Agent 2 in the region R_{22} for the combinatorial auction problem defined in Table 4.11, Panel (b): Utility of Agent 1 in the region R_{12} for the combinatorial auction problem defined in Table 4.11.

for any point in the region R_{21} not being a Nash Equilibrium is that the constraint $b_1(B) \geq [v_2(AB) - v_2(A)]$ defined in Table 4.8 and 4.9 for the region R_{21} can never be true. The value of $v_2(AB) - v_2(A)$ is 30 and the maximum value of $b_1(B)$ is $v_1(B)$ which is equal to 15. Agent 2 can always deviate from region R_{21} to R_{22} and increase its utility. Similarly, for the combinatorial auction problem defined in Table 4.11, we can say that any point in the region R_{22} can never be a Nash equilibrium because of the violation of the constraint $b_2(AB) \geq v_1(AB)$ defined in Table 4.6 and 4.7. Figure 4.7 shows the utility gradients of agent 1 and agent 2 for regions R_{12} and R_{22} respectively. The utility go Agent 2 in the region R_{22} is only dependent on agent 1's bid on the bundle AB. As illustrated in panel (b) of Figure 4.7, the utility of agent 1 in the region R_{12} is only dependent on agent 2's bid for bundle B and AB.

4.3 Preliminary experimental results for 3-Agent 3-Item combinatorial auction problem

So far we have provided a solution to a general 2-agent 2-item problem. In order for the geometric approach to work on problems of larger size, we need to write the utility

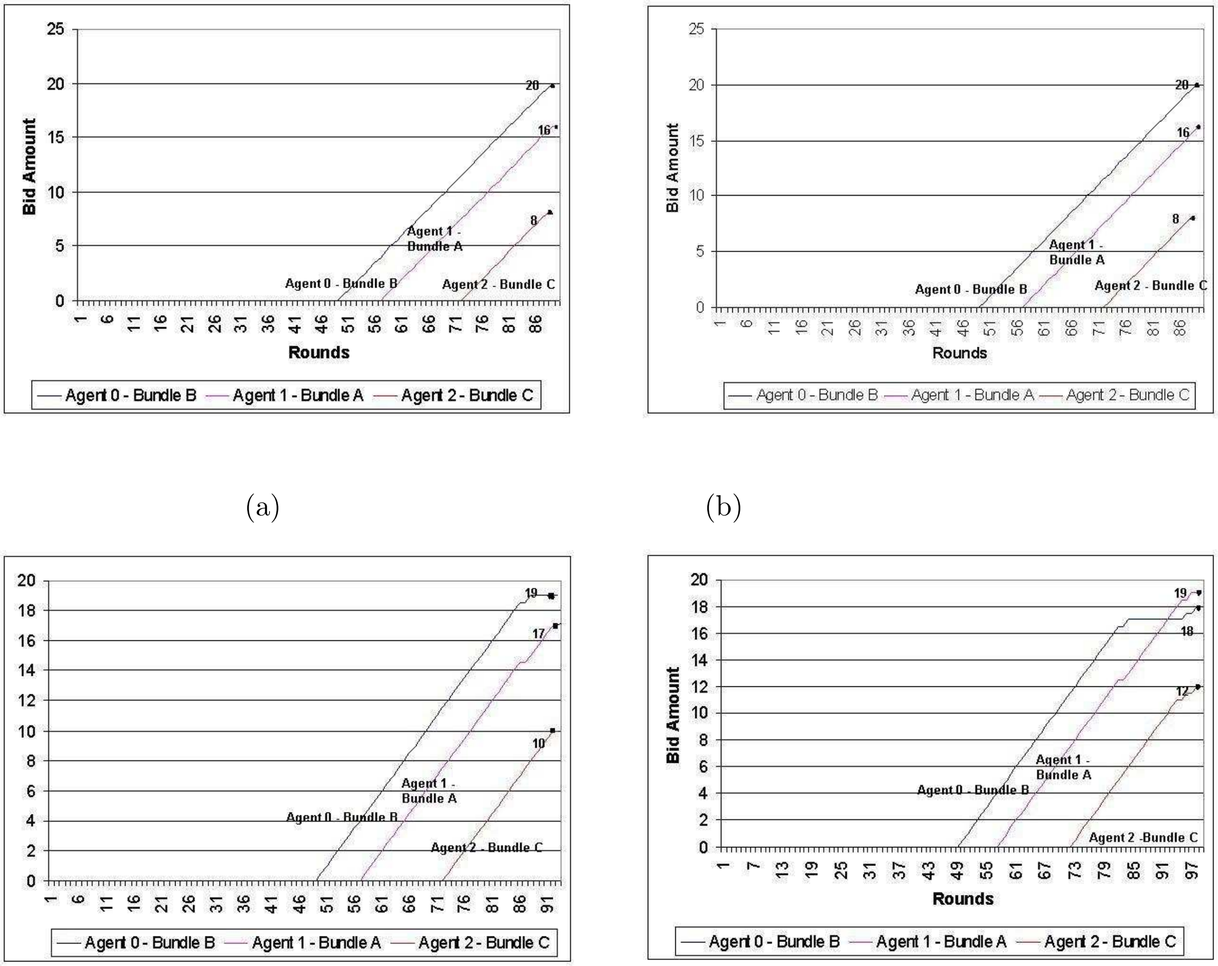
	A	**B**	**AB**	**C**	**AC**	**BC**	**ABC**
Agent 0	16	22*	27	16	31	23	46
Agent 1	23*	13	29	16	29	32	51
Agent 2	13	16	27	18*	32	36	54

Table 4.13: Valuations of 3 agents for a 3 item combinatorial auction problem

functions of each agent for each of the regions. The work presented in this section is our current progress towards understanding problems consisting of 3-agent and 3-item. Consider a problem defined in Table 4.13. We conducted experiments with the aim to study the effect of an agent's change in the bid for a bundle on its utility for a particular region. In the example problem of Table 4.13, for the allocation [$Item\ B \rightarrow Agent$ 0, $Item\ A \rightarrow Agent$ 1, $Item\ C \rightarrow Agent$ 2] to be the most efficient allocation in terms of maximizing the social welfare, the condition $v_0(AB) < 45$ must be true. We study the effect of changing the value of $v_0(AB)$ on the resultant bidding patterns and the utility of each agent. We perform experiments to gain intuition about the utility of agents in a region. Figure 4.8 shows the incremental bid values for bundles in the most efficient allocation. Figure 4.8 contains four panels and each panel has different value of $v_0(AB)$ keeping every thing else the same. The utility of agent 0 in the region R_{102} depends on its final bid on bundle B. Similarly, the utility of agent 1 and agent 2 in the region R_{102} depends on their final bids for bundle A and C respectively. We performed experiments to see how the final bid prices change as we modify the proxy bid vector. Figure 4.9 shows the change in the value of the most efficient allocation and the allocation involving bundle AB (to agent 0) for different values of $v_0(AB)$ in Table 4.13. We conducted these experiments to determine the utility gradient of agents in each region by using plots shown in Figure 4.10.

We conduct experiments to visualize the strategy space and best response functions of bidders in the form of 2D and 3D plots. We study the mathematical equations representing the multi-dimensional figures, showing specific features or patterns in the strategy space. 2D and 3D plots are used to plot the change in the surplus of a bidder for varying bids and to plot the best response strategies of a bidder to the fixed strategies of other bidders in the auction. We notice that there are specific patterns in these graphs.

The strategy space for each agent is high-dimensional where the number of dimensions is equal to the total number of bundles. The bid for each bundle represents one



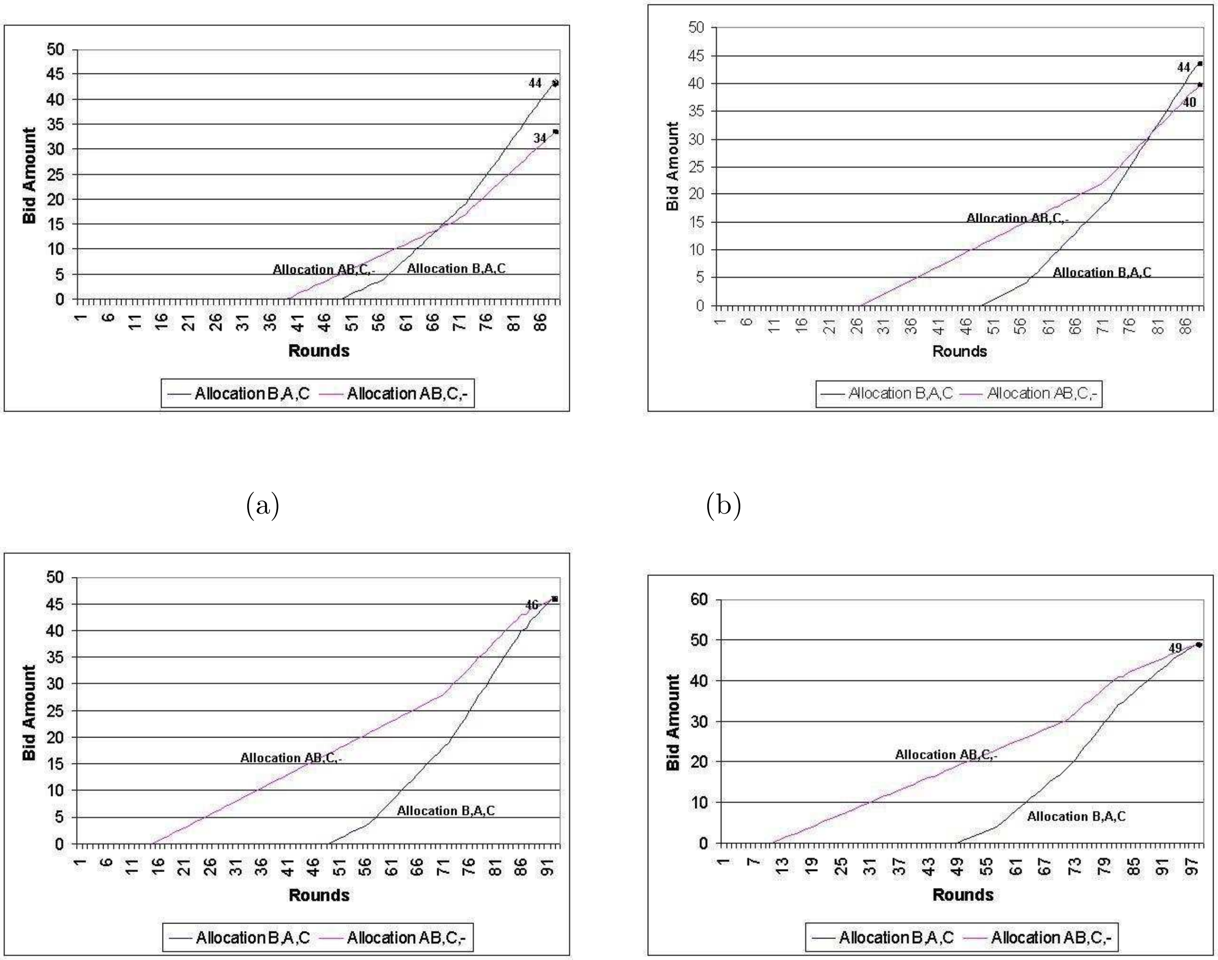
(a)

(b)

(c)

(d)

Figure 4.8: Change in the bidding pattern observed after modifying the value of $v_0(AB)$ in Table 4.13. Panel (a): $v_0(AB) = 27$, Panel (b): $v_0(AB) = 33$, Panel (c): $v_0(AB) = 39$, Panel (d): $v_0(AB) = 41$.



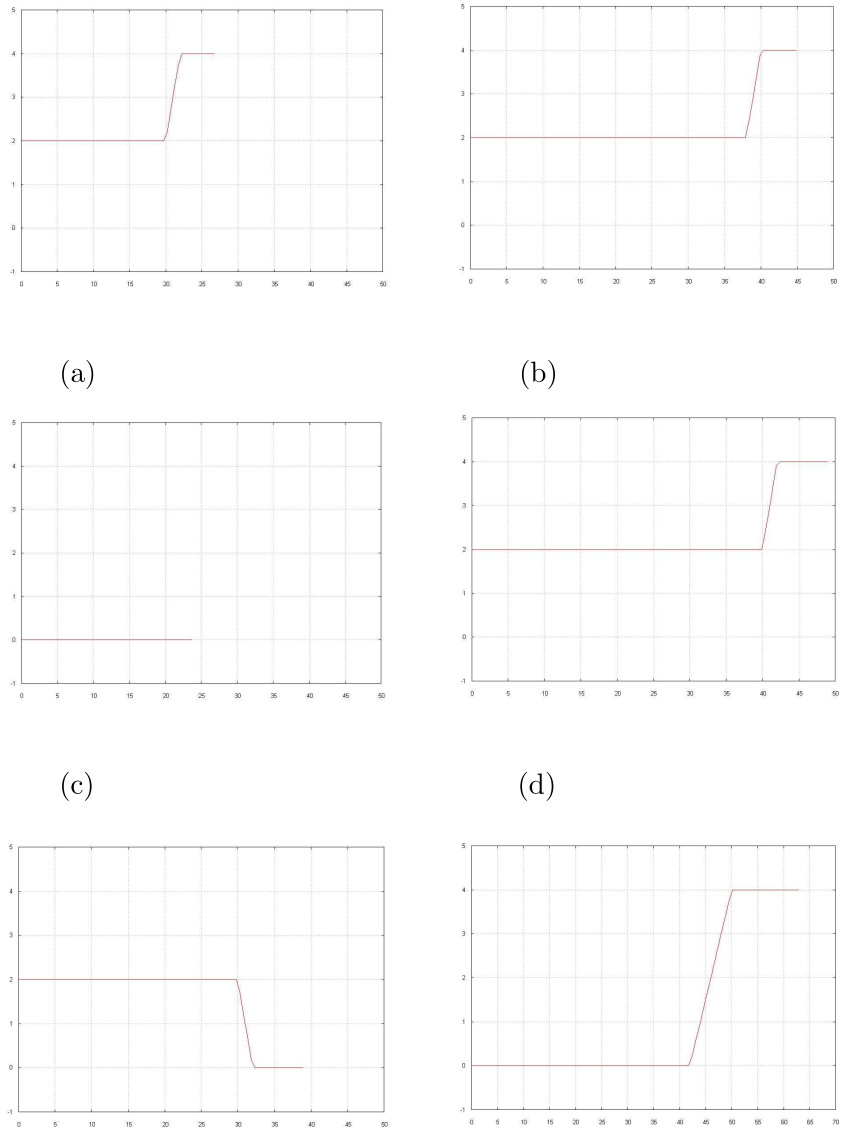
(a)

(b)

(c)

(d)

Figure 4.9: Change in the value of the most efficient allocation and the allocation involving bundle AB to agent 0 observed after modifying the value of $v_0(AB)$ in Table 4.13. Panel (a): $v_0(AB) = 27$, Panel (b): $v_0(AB) = 33$, Panel (c): $v_0(AB) = 39$, Panel (d): $v_0(AB) = 41$.



(a) (b)

(c) (d)

(a) (b)

Figure 4.10: For the problem in Table 4.13 Panel (a) and (b): Utility of agent 0 as a function of v_A and v_{AB}, Panel (c) and (d): Utility of agent 0 as a function of v_C and v_{AC}, Panel (e) and (f): Utility of agent 0 as a function of v_{BC} and v_{ABC}.

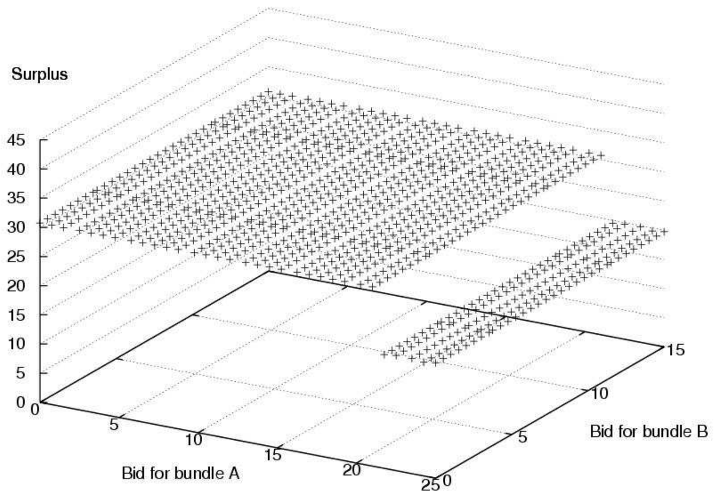


Figure 4.11: Surplus of Agent 1 for varying bids for bundle A and bundle B. The bid for bundle AB for Agent 1 is fixed at 30.5 and the bid for Agent 2 is fixed at [(A 3.5) (B 8.75) (AB 14.25)]. The plot is for Ascending k-bundle Auction with valuations defined in Table 4.11

dimension in the strategy space. For example, in a combinatorial auction with 2 items (A and B) and 3 bundles (A, B and AB), the strategy for an agent is a point in the cube, where each axis of the cube represents bids for bundle A, bundle B and bundle AB respectively. For high-dimensions a direct graphic representation of the strategy space is not possible. However, it is possible to illustrate some features of the strategy space by creating two or three-dimensional slices. In this section we present the results of slicing using 2-D and 3-D plots. Landscapes for various combinatorial optimization problems have been studied from the point of view of geometric properties such as smoothness, ruggedness and neutrality [36]. One of the reasons for studying the landscapes of optimization problems is that other combinatorial optimization problems with similar landscapes will tend themselves to similar solutions. For example, the properties of the landscape of the Travelling Salesman Problem (TSP) can be used to get a better performance from a heuristic search algorithm like simulated annealing [36].

In the following equations and graphs we plot the best response functions and surplus of agents in a multi-dimensional space.

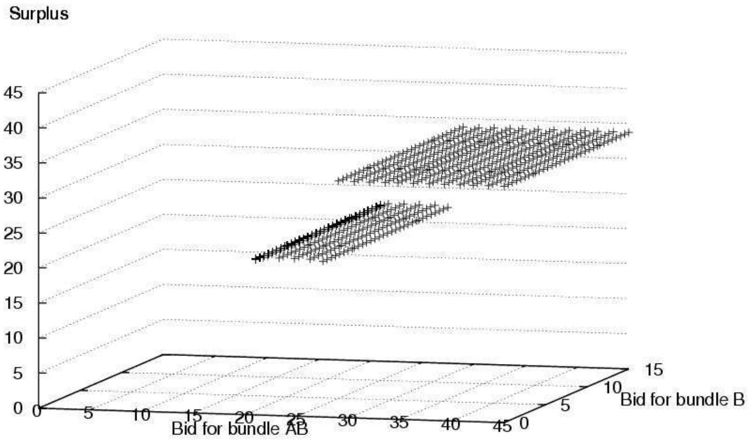


Figure 4.12: Surplus of Agent 1 for varying bids for bundle AB and bundle B. The bid for bundle A for Agent 1 is fixed at 21 and the bid for Agent 2 is fixed at [(A 4.75) (B 8.75) (AB 11.50)]. The plot is for Ascending k-bundle Auction with valuations defined in Table 4.11

$$s_1 = \ 30.75 \begin{cases} 0 \ \leq b_1(A) \ \leq \ 21.75 \\ 0 \ \leq b_1(B) \ \leq \ 15 \end{cases} \tag{4.5}$$

$$s_1 = \ 19.75 \begin{cases} 21.75 \ \leq b_1(A) \ \leq \ 25 \\ 0 \qquad \leq b_1(B) \ \leq \ 15 \end{cases} \tag{4.6}$$

$$s_1 = \ 22.50 \begin{cases} 21 \ \leq b_1(AB) \ \leq \ 29.75 \\ 0.0 \ \leq b_1(B) \ \leq \ 15 \end{cases} \tag{4.7}$$

$$s_1 = \ 33.75 \begin{cases} 29.75 \ \leq b_1(AB) \ \leq \ 45 \\ 0 \qquad \leq b_1(B) \ \leq \ 15 \end{cases} \tag{4.8}$$

Agent 1 has to coordinate its bid for bundle A, B and AB in such a way that the auctioneer allocates bundle A to agent 1 and bundle B to agent 2. The allocation $f^* = \{A, B\}$ results in the maximum surplus for agent 1, given that the bid for agent 2 is $[(A\ 10)\ (B\ 25)\ (AB\ 40)]$. The best response strategy for agent 1 is to construct the bid vector

$$s_1 = \ 03.00 \begin{cases} 0.00 \ \leq b_1(A) \ \leq \ 12.00 \\ 13.25 \ \leq b_1(B) \ \leq \ 15.00 \end{cases} \tag{4.9}$$

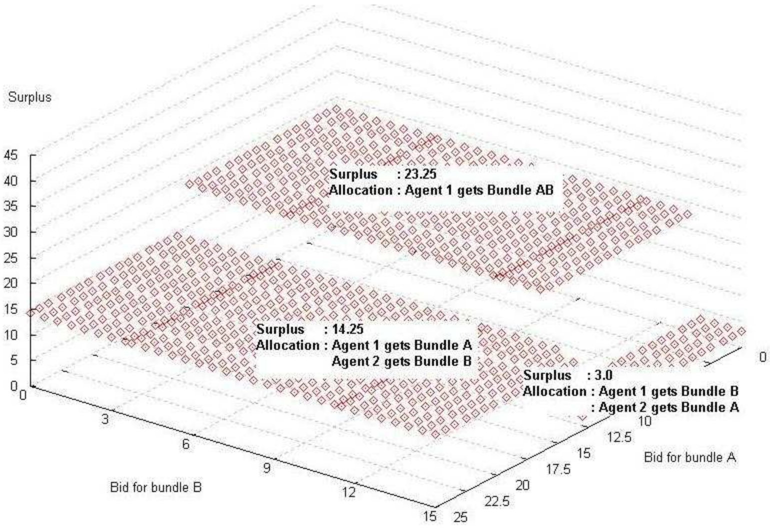


Figure 4.13: Surplus of Agent 1 for varying bids for bundle A and bundle B. The bid for bundle AB for Agent 1 is fixed at 23.25 and the bid for Agent 2 is fixed at [(A 10) (B 11.25) (AB 22]. The plot is for Ascending Package Auction with valuations defined in Table 4.11

$$s_1 = 14.25 \begin{cases} 12.00 \leq b_1(A) \leq 25.00 \\ 0.00 \leq b_1(B) \leq 13.25 \end{cases} \tag{4.10}$$

$$s_1 = 22.75 \begin{cases} 0.00 \leq b_1(A) \leq 12.00 \\ 0.00 \leq b_1(B) \leq 13.25 \end{cases} \tag{4.11}$$

The 3D geometric figure representing the best response strategy of Agent 1 when Agent 2 bids [(A 6.75) (B 16.75) (AB 26.75)] is a heptahedron[1] as shown in the Figure 4.15. The polyhedron of the Figure 4.15 is convex[2]. The twelve vertices and the seven faces of the heptahedron can be determined by solving the equations (4.9), (4.10) and (4.11). Figure 4.17 is one of the face of the heptahedron.

4.4 Conclusion

In this chapter, I present a geometric approach for determining pure strategy Nash equilibria of combinatorial auctions. We model combinatorial auctions as multi-player

[1] A heptahedron is a polyhedron with seven faces

[2] A convex polyhedron can be defined as a polyhedron for which a line connecting any two (non-coplanar) points on the surface always lies in the interior of the polyhedron

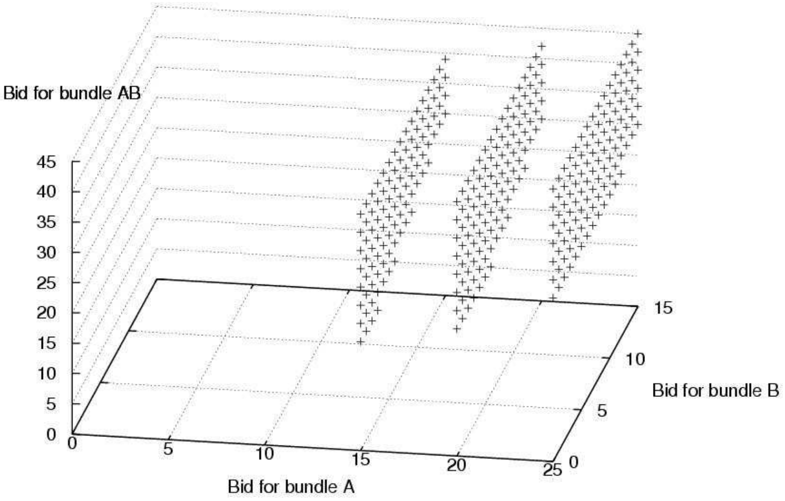


Figure 4.14: Best response strategies of Agent 1 when Agent 2 bids [(A 10) (B 25) (AB 40)]. The surplus to Agent 1 is 10.25. The plot is for combinatorial auction problem of Table 4.11

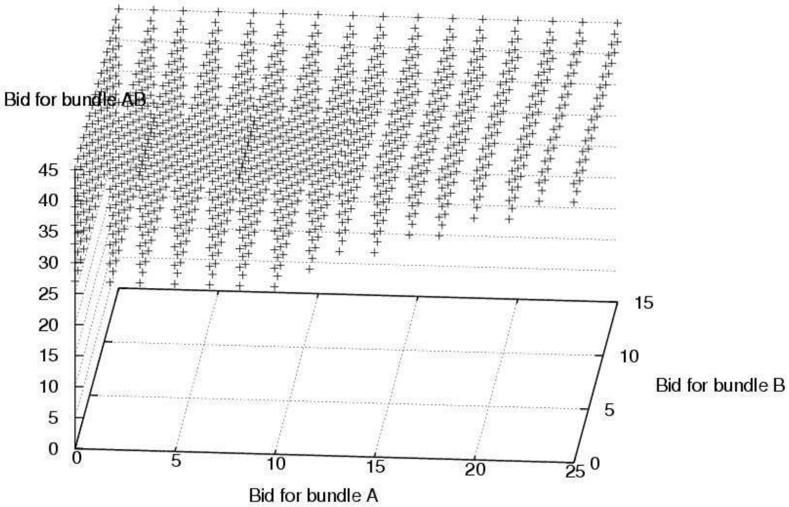


Figure 4.15: Best response strategies of Agent 1 when Agent 2 bids [(A 6.75) (B 16.75) (AB 26.75)]. The surplus to Agent 1 is 18.25. The plot is for combinatorial auction problem of Table 4.11

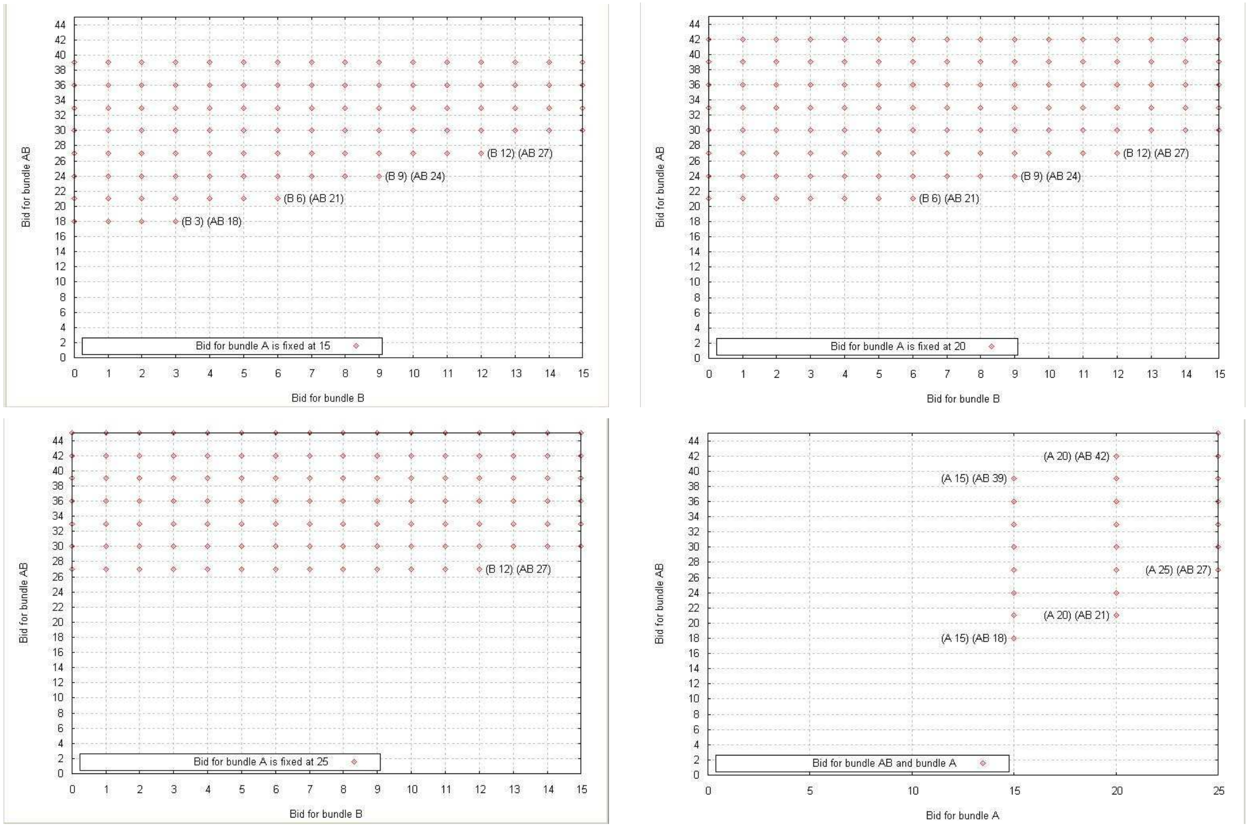


Figure 4.16: Figures showing the pattern for the best response strategy of Agent 1 when Agent 2 bids [(A 10) (B 25) (AB 40)]. The plot is for combinatorial auction problem of Table 4.11. The figures are the 2D view of the 3D plot of Figure 4.14

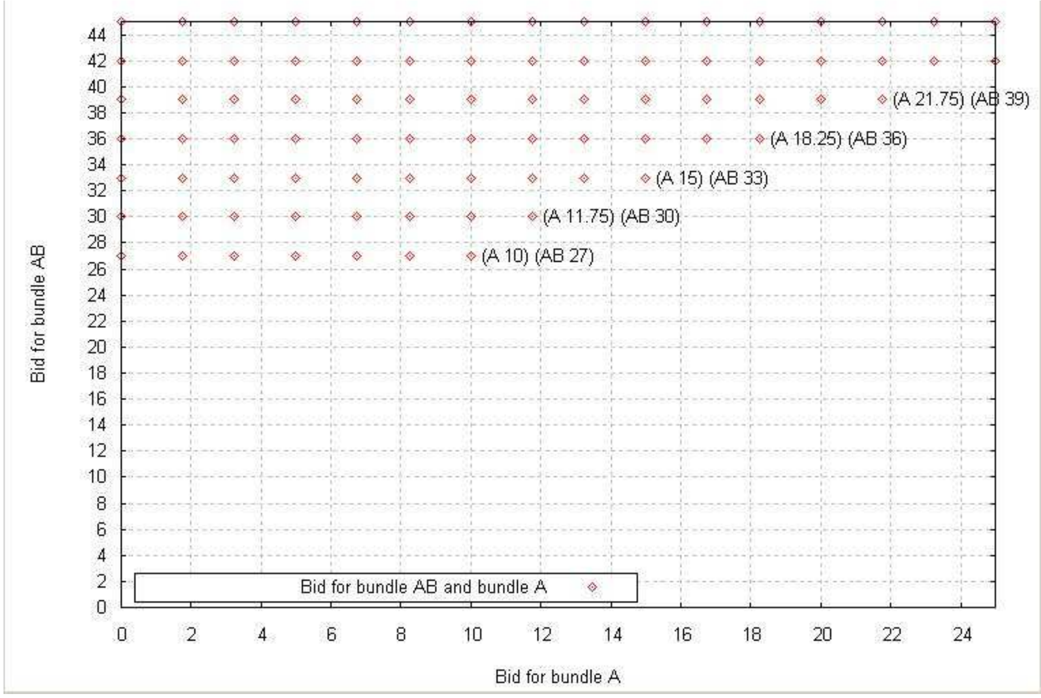


Figure 4.17: Figures showing the pattern for the best response strategy of Agent 1 when Agent 2 bids [(A 6.75) (B 16.75) (AB 26.75)]. The plot is for combinatorial auction problem of Table 4.11. The Figure is a 2D view of the 3D plot of Figure 4.15

complete information games and assume that the bidders participating in the auction follow a myopic best response bidding strategy. Myopic best response bidding to prices is a simple bidding strategy in which, in each round, the myopic bidder bids on the bundle that gives it the highest surplus as if this were the last round of the auction. Even in a restricted case of bidders following a myopic best response bidding strategy, the size of the strategy space is infinite and it is impossible to compute Nash equilibria by representing the game in normal form and populating the payoff matrix for each strategy profile. For combinatorial auctions computing the payoff for each cell in the normal form game is NP-hard and the size of the normal form game grows exponentially with the number of bidders and items. Even for a small and moderate size combinatorial auction problem, it is not possible to determine Nash equilibria using current tools. We adopt a geometric approach that takes advantage of the patterns and structure of the payoff matrix. Our approach partitions the space of outcomes into regions and we derive equations for the utility of bidders in each region and the conditions under which bidders have no incentive to deviate from one region to another. We present a solution to a general two agent and two item combinatorial auction problem using the technique. We illustrate the underlying fundamentals of our method using simple first-price and second-price sealed-bid auctions. We observe that the payoff matrix of a combinatorial auction game has a structure that can be taken advantage of in determining Nash equilibria. We believe that geometric approach can be applied for finding Nash equilibria in combinatorial auctions and we are currently working on scaling the technique to solve bigger problems.

Chapter 5

A Linear Programming Approach to Directly Compute the Outcome of the Ascending Package Auction

In Chapter 2, I present a search algorithm based on tabu search and best response dynamics to compute pure-strategy Nash equilibrium of normal-form games. The main advantage of the algorithm is that it does not require us to know the complete payoff matrix upfront and is advantageous in situations where it is computationally expensive to compute the payoff matrix. We observe that computational savings can be achieved if we compute the payoffs only when it is required. Even though the algorithm is applicable to any situation where it is time consuming to determine the payoffs of the player as a result of joint actions, the main research motivation behind the algorithm was the need to devise an algorithm to quickly compute pure strategy Nash equilibrium of combinatorial auctions. The motivation behind this chapter is to devise an algorithm to reduce the time to compute the payoff matrix itself. Unlike the approaches presented in Chapter 2, which are applicable to a large class of problems, the algorithm presented in this chapter is applicable to a specific type of combinatorial auction called the Ascending Package Auction [2]. In this chapter, we present a linear program to compute the outcome of the Ascending Package Auction directly without actually running the auction. The advantage of direct computation is the time saved in computing the payoffs of bidders, enabling faster computation of the values of the cell in a normal form game. We provide an upper bound on the number of optimization problems to solve to compute the results of the linear program. Any technique that speeds the process of computing the payoffs in turn speeds the process of determining the Nash equilibrium by realizing overall time savings in determining the payoff matrix. Throughout this chapter our assumption is that the bidders submit a single bid vector to the proxy agent who in turn bids on behalf of the bidders in a straightforward manner. Any bid vector satisfying the free disposal property and up to the valuation of the bidder for the various combinations of items constitutes a strategy.

The problem of determining the outcome (item allocation and payments by the bidders) of combinatorial auctions with myopic bidders without actually running the iterative auction is referred to as the Proxy Auction Problem (PAP) [48, 49]. Recently, there has been work on solving the proxy auction problem for the Ascending k-Bundle Auction and the Ascending Package Auction by exploiting the bidding patterns and the price trajectories obtained as a result of running the auction with proxy bidders [48, 49]. The linear program presented in this chapter is the result of analyzing the outcome and the proxy bidding pattern of the Ascending Package Auction. We observed the change in auction outcome and bidding pattern by varying the number of bidders, items and the valuation profile of the bidders and derived a linear program.

Following is an outline for the rest of the chapter. Section 5.1 defines the Proxy Auction Problem and its advantages. Section 5.2 provides a formal description of the linear program to solve the proxy auction problem for the Ascending Package Auction and lists some important aspects of the algorithm. In Section 5.3, we present some worked out examples for 3-Agent 3-Item and 4-Agent 4-Item problems. Section 5.4 is the conclusion and future work. Finally, Section 6 provides a high level overview of the dissertation contributions to conclude the thesis.

5.1 Proxy Auction Problem

Consider a setting with one seller and n buyers. The buyers are also referred to as bidders and the seller as the auctioneer. The set of buyers is denoted by N. The seller has a set M of indivisible and heterogenous items to sell. The number of items are equal to m. The buyers are indexed by i and the items by j. Any subset of the items is called a bundle, and denoted by $b \subseteq M$. There are $2^m - 1$ bundles excluding the empty set. The value of buyer i for bundle b is denoted by $v_i(b)$. Each buyer i has a valuation function for each bundle and a value of zero for the empty set ϕ (i.e., $v_i(\phi) = 0$). We make the standard assumption of free disposal which means that the value of a subset of a bundle is less than or equal to the value of the bundle (i.e., $\forall b^{'} \subset b$, $v_i(b^{'}) \leq v_i(b)$). Let $r_i(b)$ denote the proxy bid of agent i for the bundle b. Let C denote a subset of bidders, that is, $C \subseteq N$. Let V^* denote the value of the optimal allocation based on the reported proxy bids of all the agents and V^*_{-C} denote the value of the optimal allocation based on the reported proxy bids without the agents in the set C. Let f^* corresponds to the assignment of items to the agents for the revenue maximizing allocation based on the reported proxy bids. Let s_i be the surplus of agent i based on the reported proxy bids. The proxy bid need not be qual to the true valuations of the bidder. The proxy bid vector of an agent represents the agent's strategy and the agents may have an incentive to misrepresent their proxy bid to get a higher surplus. Let p_i be the payment by agent i to the auctioneer in exchange for the bundle. The payment p_i of agent i is 0 if the agent is not part of the auction allocation. The surplus s_i of an agent i is equal to the different between the true valuation and the payment for the allocation f i.e $s_i(f) = v_i(f) - p_i(f)$.

A solution to the proxy auction problem is to determine f^* (the auction allocation which in most cases is the optimal allocation based on the reported proxy bids) and $\forall i$, p_i (the payment of every agent i). Once the auction allocation and payments are known then the surplus to each bidder and the auction revenue can be easily determined. The surplus to an agent is the difference between the agent's valuation for the allocated bundle and its payment for the allocated bundle. The auction revenue is the sum of the payments from all the bidders.

	A	**B**	**AB**
Agent 1	8	7	9
Agent 2	1	3	9
Agent 3	2	1	10

Table 5.1: Example problem with 3 agents and 2 items

5.1.1 Advantages of a direct solution over simulation

A natural way to solve the proxy auction problem is to determine the outcome of the auction by running the auction with myopic bidders. This is referred as solving the proxy auction problem by simulation. The auction proceeds in rounds and in each round the buyers place offers on a subset of the bundles. At the beginning of the auction the ask price for each bundle is 0. The proxy bidders bid myopically and place bids on bundles that receives the maximum surplus to the bidder it represents. The maximum amount that a proxy bidder can bid for a bundle is the amount reported in the proxy statement for that bundle. At each round the auctioneer determines the revenue maximizing allocation and the ask bid prices for the next round. The proxy bidding policy enables bidders to revise their bids based on the feedback from the auctioneer (provisional allocation and ask prices). The auction terminates when no further bids are submitted. Determining the outcome of the auction through the bidding process is referred to as the simulation approach [48, 49]. Table 5.2 shows the incremental bids placed by the proxy agent for an example problem of Table 5.1 for the Ascending Package Auction. The bid increment used for the bidding process shown in Table is 5.2 1 and each column shows the agents proxy bid vector. As shown in Table 5.2 the number of rounds it took for the auction to terminate is 34.

The number of rounds it takes for the auction to terminate depends on the bid increment. At each round of the iterative combinatorial auction, the auctioneer solves an NP-hard problem referred to as the Winner Determination Problem (WDP). Determining the provisional allocation and prices at each round thus requires solving an optimization problems and that makes combinatorial auctions computationally complex to clear [5, 42, 8].The number of hard optimization problems to solve in the simulation approach thus depends on the bid increment. The smaller the bid increment the more number of rounds are required for the auction to terminate. If the bid increment is increased then the auction will terminate faster but the results in terms of the bidder payments and surplus will be less accurate. Table 5.3 illustrates the effect of the bid increment on the number of rounds required to terminate for the problem in Table 5.1.

The problem for which the results in Table 5.3 are presented is a very simple problem with just three agents and two items. Even for such a basic problem the number of rounds required for the auction to terminate is considerable which means solving a large number of NP hard optimization problem. As the problem size grows the number of rounds taken by the auction to terminate also increases. The research motivation is to develop a technique to solve the proxy auction problem in a more efficiently. In Section 5.2, we present a linear programming formulation for the Ascending Package Auction to solve the proxy auction problem.

5.2 LP Formulation

After analyzing the bidding patterns and results for several problems of varying size, valuations and allocation, we derived a linear program that can be used to directly predict the outcome of Ascending Package Auction. The objective function of the linear program is to maximize the sum of surplus of all the agents. Formally, the objective function can be written as equation (5.1).

$$Maximize \sum_{i}^{|N|} s_i \tag{5.1}$$

where N is the set of agents and s_i is the surplus of agent i. Let $Com(n, k)$ be a set of possible combinations of n things taken k at a time i.e for all positive integers n and k, where $k \leq n$. The size of the set $Com(n, k)\ is\ \frac{n!}{(n-k)!.k!}$. Let C be a subset of N. The number of constraints nC is shown in equation (5.2).

$$nC \ = \ \sum_{i=i}^{|N|-1} Com(n, i) \ + \ N \tag{5.2}$$

The constraints for the linear program are generated using the following pseudocode.

Rounds	Agent 1	Agent 2	Agent 3
1	0 0 1	0 0 0	0 0 0
2	0 0 1	0 0 1	0 0 0
3	0 0 1	0 0 1	0 0 1
4	1 0 2	0 0 1	0 0 1
5	1 0 2	0 0 2	0 0 1
6	1 0 2	0 0 2	0 0 2
7	1 0 2	0 0 2	0 0 2
8	1 0 2	0 0 3	0 0 2
9	1 0 2	0 0 3	0 0 3
10	2 1 3	0 0 3	0 0 3
11	2 1 3	0 0 4	0 0 3
12	2 1 3	0 0 4	0 0 4
13	3 2 4	0 0 4	0 0 4
14	3 2 4	0 0 5	0 0 4
15	3 2 4	0 0 5	0 0 5
16	4 3 5	0 0 5	0 0 5
17	4 3 5	0 0 6	0 0 5
18	4 3 5	0 0 6	0 0 6
19	5 4 6	0 0 6	0 0 6
20	5 4 6	0 1 7	0 0 6
21	5 4 6	0 1 7	0 0 7
22	6 5 7	0 1 7	0 0 7
23	6 5 7	0 2 8	0 0 7
24	6 5 7	0 2 8	0 0 8
25	6 5 7	0 2 8	0 0 8
26	6 5 7	0 2 8	0 0 8
27	6 5 7	0 2 8	1 0 9
28	7 6 8	0 2 8	1 0 9
29	7 6 8	0 2 8	1 0 9
30	7 6 8	0 2 8	2 1 10
31	8 7 9	0 2 8	2 1 10
32	8 7 9	0 2 8	2 1 10
33	8 7 9	0 2 8	2 1 10
34	8 7 9	0 2 8	2 1 10

Table 5.2: Example problem with 3 agents and 2 items

Bid increment	Number of rounds
0.1	315
0.2	159
0.4	84
0.8	42
1.0	34
1.5	23
2.0	19

Table 5.3: Effect of bid increment on the number of rounds required for the auction to terminate

```
Algorithm 5.2.1: CONSTRAINTS()

 for $i \leftarrow 0$ to $N-1$for every element $C$ of set $Com(n,k)$
    Generate constraint : $\sum_{i \in C} s_i \leq (V^* - V^*_{-C})$

 $\forall_i,\ s_i \geq 0$
```

The number of optimization problems to be solved using the linear programming approach can be written as equation 5.3

$$nC \;=\; \sum_{i=i}^{N-1} Com(n,i) \;+\; 2 \tag{5.3}$$

5.3 Worked Examples

Following are some worked out problems for 3-agent 3-item and 4-agent 4-item problems. We implemented the algorithm and the linear program in the Java programming language and present the results of running the program for three different types of problems.

5.3.1 Example 1

Table 5.5 shows the working of the linear program for the problem defined in Table 5.4. The value of objective function for the Linear Program in Table 5.5 is 21. The value of the decision variables are $s_1 = 4,\ s_2 = 9, s_3 = 8$. The auction allocation is {Agent 1 → Bundle B, Agent 2 → Bundle B, Agent 3 → Bundle C}. The payment

	A	**B**	**AB**	**C**	**AC**	**BC**	**ABC**
Agent 1	16	22*	27	16	31	23	46
Agent 2	23*	13	29	16	29	32	48
Agent 3	13	16	27	18*	32	36	50

Table 5.4: Proxy bids

for Agent 1 is $p_1 = r_1(B) - s_1 = 22 - 4 = 18$. Similarly, the payment for Agent 2 is $p_2 = r_2(A) - s_2 = 23 - 9 = 14$ and Agent 3 is $p_3 = r_3(C) - s_3 = 18 - 8 = 10$. The revenue to the auctioneer is $\sum_{i=1}^{N} p_i \ = 42$.

5.3.2 Example 2

Table 5.7 shows the working of the linear program for the problem defined in Table 5.6. The value of optimal solution to the Linear Program in Table 5.7 is 8. The value of the decision variables are $s_1 = 7,\ s_2 = 1, s_3 = 0$. The auction allocation is {Agent 1 → Bundle A, Agent 2 → Bundle BC}. The payment for Agent 1 is $p_1 = r_1(A) - s_1 = 20 - 7 = 13$. Similarly, the payment for Agent 2 is $p_2 = r_2(BC) - s_2 = 42 - 1 = 41$ and Agent 3 is $p_3 = 0$. The revenue to the auctioneer is $\sum_{i=1}^{N} p_i \ = 54$.

5.3.3 Example 3

Table 5.10 shows the working of the linear program for the problem defined in Table 5.8 and 5.9. The value of optimal solution to the Linear Program in Table 5.10 is 17. The value of the decision variables are $s_1 = 8,\ s_2 = 5, s_3 = 3, s_4 = 1$. The auction allocation is {Agent 1 → Bundle A, Agent 2 → Bundle B, Agent 3 → Bundle C, Agent 4 → Bundle D}. The payment for Agent 1 is $p_1 = r_1(A) - s_1 = 18 - 8 = 10$. Similarly, the payments for all other agents can be determined. The revenue to the auctioneer is $\sum_{i=1}^{N} p_i \ = 61$.

5.4 Conclusion

In this Chapter, I present a linear programming approach to directly compute the outcome of the Ascending Proxy Auction. The auction allocation, final round bids, auction revenue, bidder surplus and payments can be determined by solving a linear program.

Objective Function	Maximize $\sum_i^N s_i$	Maximize $(s_1 + s_2 + s_3)$
Constraints (Without 1 Bidder)	$C_1 = \{1\},\ \sum_{i \in C_1} s_i \leq (V^* - V^*_{-C_1})$ $C_2 = \{2\},\ \sum_{i \in C_1} s_i \leq (V^* - V^*_{-C_2})$ $C_3 = \{3\},\ \sum_{i \in C_3} s_i \leq (V^* - V^*_{-C_3})$	$s_1 \leq (63 - 59 = 4)$ $s_2 \leq (63 - 54 = 9)$ $s_3 \leq (63 - 51 = 12)$
Constraints (Without 2 Bidders)	$C_{12} = \{1,2\},\ \sum_{i \in C_{12}} s_i \leq (V^* - V^*_{-C_{12}})$ $C_{23} = \{2,3\},\ \sum_{i \in C_{23}} s_i \leq (V^* - V^*_{-C_{23}})$ $C_{13} = \{1,3\},\ \sum_{i \in C_{13}} s_i \leq (V^* - V^*_{-C_{13}})$	$s_1 + s_2 \leq (63 - 50 = 13)$ $s_2 + s_3 \leq (63 - 46 = 17)$ $s_1 + s_3 \leq (63 - 51 = 12)$
Constraints (Non-negative surplus)	$\forall_i,\ s_i \geq 0$	$s_1 \geq 0$ $s_2 \geq 0$ $s_3 \geq 0$

Table 5.5: Linear program for problem in Table 5.4

	A	**B**	**AB**	**C**	**AC**	**BC**	**ABC**
Agent 1	20*	22	27	16	31	23	56
Agent 2	13	13	29	16	29	42*	51
Agent 3	13	16	27	19	32	36	54

Table 5.6: Proxy bids

Objective Function	Maximize $\sum_i^N \ s_i$	Maximize $(s_1 \ + \ s_2 \ + \ s_3)$
Constraints (Without 1 Bidder)	$C_1 = \{1\},\ \sum_{i\in C_1} s_i \leq (V^* - V^*_{-C_1})$ $C_2 = \{2\},\ \sum_{i\in C_1} s_i \leq (V^* - V^*_{-C_2})$ $C_3 = \{3\},\ \sum_{i\in C_3} s_i \leq (V^* - V^*_{-C_3})$	$s_1 \leq (62 - 55 = 7)$ $s_2 \leq (62 - 56 = 6)$ $s_3 \leq (62 - 62 = 0)$
Constraints (Without 2 Bidders)	$C_{12} = \{1,2\},\ \sum_{i\in C_{12}} s_i \leq (V^* - V^*_{-C_{12}})$ $C_{23} = \{2,3\},\ \sum_{i\in C_{23}} s_i \leq (V^* - V^*_{-C_{23}})$ $C_{13} = \{1,3\},\ \sum_{i\in C_{13}} s_i \leq (V^* - V^*_{-C_{13}})$	$s_1 + s_2 \leq (62 - 54 = 8)$ $s_2 + s_3 \leq (62 - 56 = 6)$ $s_1 + s_3 \leq (62 - 51 = 11)$
Constraints (Non-negative surplus)	$\forall_i,\ s_i \geq 0$	$s_1 \geq 0$ $s_2 \geq 0$ $s_3 \geq 0$

Table 5.7: Linear program for problem in Table 5.6

	A	**B**	**AB**	**C**	**AC**	**BC**	**ABC**
Agent 1	18*	19	32	14	37	28	43
Agent 2	14	22*	32	16	36	32	44
Agent 3	15	14	31	22*	30	33	49
Agent 4	16	12	35	16	33	30	41

Table 5.8: Proxy bids

	D	**AD**	**BD**	**ABD**	**CD**	**ACD**	**BCD**	**ABCD**
Agent 1	19	26	38	41	33	45	47	65
Agent 2	17	34	32	44	35	41	44	58
Agent 3	12	33	38	46	31	43	45	74
Agent 4	26*	24	39	41	38	42	43	51

Table 5.9: Proxy bids

	Maximize $\sum_i^N\ s_i$	Maximize $(s_1\ +\ s_2\ +\ s_3\ +\ s_4)$
Constraint 1	$C_1 = \{1\},\ \sum_{i \in C_1} s_i \leq (V^* - V^*_{-C_1})$	$s_1 \leq (88 - 80 = 8)$
	$C_2 = \{2\},\ \sum_{i \in C_1} s_i \leq (V^* - V^*_{-C_2})$	$s_2 \leq (88 - 80 = 8)$
	$C_3 = \{3\},\ \sum_{i \in C_3} s_i \leq (V^* - V^*_{-C_3})$	$s_3 \leq (88 - 85 = 3)$
	$C_4 = \{4\},\ \sum_{i \in C_4} s_i \leq (V^* - V^*_{-C_4})$	$s_4 \leq (88 - 75 = 13)$
Constraint 2	$C_{12} = \{1,2\},\ \sum_{i \in C_{12}} s_i \leq (V^* - V^*_{-C_{12}})$	$s_1 + s_2 \leq (88 - 75 = 13)$
	$C_{13} = \{1,3\},\ \sum_{i \in C_{13}} s_i \leq (V^* - V^*_{-C_{13}})$	$s_1 + s_3 \leq (88 - 75 = 13)$
	$C_{14} = \{1,4\},\ \sum_{i \in C_{14}} s_i \leq (V^* - V^*_{-C_{14}})$	$s_1 + s_4 \leq (88 - 74 = 14)$
	$C_{23} = \{2,3\},\ \sum_{i \in C_{23}} s_i \leq (V^* - V^*_{-C_{23}})$	$s_2 + s_3 \leq (88 - 76 = 12)$
	$C_{24} = \{2,4\},\ \sum_{i \in C_{24}} s_i \leq (V^* - V^*_{-C_{24}})$	$s_2 + s_4 \leq (88 - 75 = 13)$
	$C_{34} = \{3,4\},\ \sum_{i \in C_{34}} s_i \leq (V^* - V^*_{-C_{34}})$	$s_3 + s_4 \leq (88 - 74 = 14)$
Constraint 3	$C_{123} = \{1,2,3\},\ \sum_{i \in C_{123}} s_i \leq (V^* - V^*_{-C_{123}})$	$s_1 + s_2 + s_3 \leq (88 - 51 = 37)$
	$C_{124} = \{1,2,4\},\ \sum_{i \in C_{124}} s_i \leq (V^* - V^*_{-C_{124}})$	$s_1 + s_2 + s_4 \leq (88 - 74 = 14)$
	$C_{234} = \{2,3,4\},\ \sum_{i \in C_{234}} s_i \leq (V^* - V^*_{-C_{234}})$	$s_2 + s_3 + s_4 \leq (88 - 65 = 23)$
	$C_{134} = \{1,3,4\},\ \sum_{i \in C_{134}} s_i \leq (V^* - V^*_{-C_{134}})$	$s_1 + s_3 + s_4 \leq (88 - 58 = 30)$
Constraint 4	$\forall_i,\ s_i \geq 0$	$s_1 \geq 0$
		$s_2 \geq 0$
		$s_3 \geq 0$
		$s_4 \geq 0$

Table 5.10: Linear program for problem in Table 5.8 and 5.9

We provide an upper bound on maximum number of optimization problem required to be solved by the linear program. The direct approach is independent of the bid increment, the order of the bidders and the tie breaking rules. We applied the algorithm to a large number of problems of different sizes and compared the result of the linear program with the results obtained from simulation. We observed that our algorithm based on the linear programs gives accurate results. Computing the outcome of combinatorial auction directly without actually running the auction may enable a faster computation of the payoff matrix for the purpose of determining the pure strategy Nash equilibrium with discrete strategy space. We plan to conduct further tests to do a comparison of the amount of time taken and the number of optimization problems to solve by simulation approach and the linear programming approach. We present a linear programming approach to solve the Proxy Auction Problem for Ascending Package Auction and in the future we plan to work on finding a direct solution to other types of iterative combinatorial auctions like Ascending k-bundle auction and iBundle auction. We have empirical results to support our claim that the linear programming approach gives accurate results and in the future we plan to work on providing a theoretical proof of the correctness of the linear program to solve the Proxy Auction Problem.

Chapter 6

Dissertation Contributions

In my dissertation, I developed four novel techniques addressing the problems in finding Nash equilibria in combinatorial auctions. The work presented in this dissertation is progress towards algorithms for finding Nash equilibria in a game for which it is computationally expensive to compute the payoffs and for which the size of the payoff matrix is very large. Following is a high level overview of the four approaches developed in this dissertation:

Approach 1 An approach to compute pure strategy Nash equilibria of a game without computing the whole pay-off matrix. This reduces the amount of time required to compute the payoff matrix thereby reducing the total amount of time required to find a Nash equilibrium.

Approach 2 An approach to find an approximate Nash equilibrium. The approach is useful in situations where it is computationally expensive to find a Nash equilibrium by enumeration because of the very large size of the solution space. The approach compromises completeness in return for computational efficiency.

Approach 3 An approach that takes advantage of the structure and pattern of the payoff matrix to find Nash equilibria without actually running the auction and constructing a payoff matrix.

Approach 4 A linear programming approach to directly compute the outcome of running a specific type of combinatorial auction with proxy bidders. The goal of this approach is to reduce the amount of time required to determine the payoffs for a joint action thereby reducing the total amount of time required to compute Nash equilibria.

Bibliography

[1] E. Aarts and J. K. Lenstra (eds.). Local Search in Combinatorial Optimization. John Wiley and Sons, Chichester, UK. 1997.

[2] L. M. Ausubel and P. Milgrom, Ascending Auctions with Package Bidding. Frontiers of Theoretical Economics: Vol. 1: No. 1, Article 1. 2002.

[3] R. Cassady, Auctions and Auctioneering. Berkely University of California Press. 1967.

[4] P. Cramton, The FCC Spectrum Auctions: An Early Assessment, Journal of Economics and Management Strategy, 6:3, 431-495. 1997.

[5] S. deVries and R. Vohra, Combinatorial auctions: a survey. INFORMS Journal of Computing. 2002.

[6] D. deWerra and A. Hertz, Tabu search techniques: a tutorial and an application to neural networks. OR Spektrum 11, 131 - 141. 1989.

[7] D. Fudenberg and J. Tirole, Game Theory, MIT Press, ISBN: 0262061414. 1994.

[8] Y. Fujishima, K. Leyton-Brown and Y. Shoham, Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In Proceedings of the International Joint Conference on Artificial Intelligence, Stockholm, Sweden. 1999.

[9] E. Gimnez-Funes, L. Godo, J. A. Rodrguez-Aguilar and P. Garcia-Calvs, Designing Bidding Strategies for Trading Agents in Electronic Auctions. Proceedings of the Third International Conference on Multi-Agent Systems. 136-143. 1998.

[10] H. Ginitis, Game Theory Evolving: A Problem-Centered Introduction to Modeling Strategic Behavior. Princeton University Press. Princeton, New Jersey. 2000.

[11] F. Glover, Future paths for integer programming and links to artificial intelligence. Computers and Operations Research 13, 533 - 549. 1986.

[12] F. Glover, Tabu search: part I. ORSA Journal on Computing 1, 190 - 206. 1989.

[13] F. Glover, Tabu search: part II. ORSA Journal on Computing 2, 4 - 32. 1990.

[14] F. Glover and M. Laguna. Tabu Search. Kluwer Academic Publishers, Norwell, MA. 2004

[15] F. Glover, W. Kochenberger and A. Gary (Eds.), Handbook of Metaheuristics Series: International Series in Operations Research and Management Science, Vol. 57 ISBN: 0-306-48056-5. 2003.

[16] S. Govindan and R. Wilson. A Global Newton Method to Compute Nash Equilibria, Journal of Economic Theory, 110(1), 2003.

[17] G. van der Laan, A. J. J. Talman, and L. van Der Heyden, Simplicial variable dimension algorithms for solving the nonlinear complementarity problem on a product of unit simplices using a general labelling, 377-397, Mathematics of Operations Research. 1987.

[18] P. Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. Talk presented at the Congress on Numerical Methods in Combinatorial Optimization. 1986.

[19] F. Kelly and R. Steinberg, A combinatorial auction with multiple winners for universal services. Management Science, 46(4). 586-596. 2000.

[20] C. E. Lemke and J. T. Howson, Equilibrium points of bimatrix games, 413-423, Journal of the Society of Industrial and Applied Mathematics, 12. 1964.

[21] A. J. McKenzie, Evolutionary Game Theory, The Stanford Encyclopedia of Philosophy. 2003.

[22] R. McKelvey and A. McLennan, Computation of equilibria in finite games, 87-142, Handbook of Computational Economics, Edited by H. Amman, D. Kendrick, J. Rust, Elsevier. 1996.

[23] R. McKelvey, D. Richard , A. McLennan, M. Andrew, and T.L. Theodore, Gambit: Software Tools for Game Theory, Version 0.97.0.6. 2004

[24] McMillan, J. Selling spectrum rights. Journal of Economic Perspectives 8 (1994), 145-62.

[25] R. B. Myerson, Game Theory: An Analysis of Conflict. MIT Press, Cambridge, MA. 1991.

[26] J. Nash, Equilibrium points in n-person games. Proceedings of the National Academy Of Sciences, 36:48-49. 1950.

[27] J. von Neumann and O. Morgenstern, Theory of Games and Economic Behavior. Princeton University Press. 1944.

[28] N. Nilsson, Problem-Solving Methods in Artificial Intelligence. McGraw-Hill, New York. 1971.

[29] E. Nudelman, J. Wortman, K. Leyton-Brown and Y. Shoham. Run the GAMUT: A Comprehensive Approach to Evaluating Game-Theoretic Algorithms, AAMAS. 2004.

[30] M. J. Osborne, An Introduction to Game Theory. Oxford University Press. 2004.

[31] D. C. Parkes, iBundle: An efficient ascending price bundle auction. First ACM Conference on Electronic Commerce. 148-157. 1999.

[32] D. C. Parkes and L. Ungar, Iterative combinatorial auctions: Theory and practice. Seventh National Conference on Artificial Intelligence. 74-81. 2000.

[33] P. P. Phadke, An evolutionary approach to finding bidding strategies in Combinatorial Auction. Electronic Theses and Dissertations at NC State. 2002.

[34] S. J. Rassenti, V. L. Smith, and R. L. Bulfin. A Combinatorial Auction Mechanism for Airport Time Slot Allocation. Bell Journal. of Economics Volume 13, No. 2, 4012-417. 1982.

[35] C. Reeves (ed.), Modern Heuristic Techniques for Combinatorial Problems. John Wiley and Sons, Publishers, Norwell. 1997.

[36] Reidys, C. M. and Stadler, P. F. Combinatorial landscapes. SIAM Review, 44:3–54, 2002.

[37] M. G. Resende, Pinho de Sousa, J. (Eds.), Metaheuristics Computer Decision-Making Series: Applied Optimization, Vol. 86. 2004.

[38] M. H. Rothkopf, A. Pekec and R. M. Harstad, Computationally Manageable Combinatorial Auctions. Management Science, v.44 n.8. 1131-1147. 1998.

[39] L. Samuelson, Evolutionary Games and Equilibrium Selection. MIT Press Series on Economic Learning and Social Evolution. 2002.

[40] J. M. Smith, Evolution and the Theory of Games. Cambridge: Cambridge University Press, 1982.

[41] J. M. Smith, and G. R. Price. 1973. The logic of animal conflict. Nature 246: 15-18. 1973.

[42] T. Sandholm, Algorithm for Optimal Winner Determination in Combinatorial Auctions. Artificial Intelligence, 135, 1-54. 2002.

[43] J. B. Van Huyck, R. C. Battalio and R. O. Beil, Tacit Coordination Games, Strategic Uncertainty, and Coordination Failure, American Economic Review. 1990.

[44] H. R. Varian, Economic Mechanism Design for Computerized Agents. In proceedings of the first Usenix Conference on Electronics Commerce, New York. 1995.

[45] M. P. Wellman, P. R. Wurman, K. O'Malley, R. Bangera, S. Lin, D. Reeves, and W. E. Walsh. Designing the market game for a trading agent competition. IEEE Internet Computing. 2001.

[46] P. R. Wurman and M. P. Wellman, Equilibrium prices in bundle auctions. AAAI-99 Workshop on Artificial Intelligence for Electronic Commerce, 56-61. 1999.

[47] P. R. Wurman and M. P. Wellman, AkBA: A Progressive, Anonymous-Price Combinatorial Auction. Second ACM Conference on Electronic Commerce, 21-29. 2000.

[48] P. R. Wurman, G. Cai, J. Zhong, and A. Sureka. An Algorithm for Computing the Outcome of Combinatorial Auctions with Proxy Bidding. Fifth International Conference on Electronic Commerce. 2003.

[49] P. R. Wurman, G. Cai, A. Sureka: Computing the outcome of proxy bidding in combinatorial auctions extended abstract. ACM Conference on Electronic Commerce (ACM-EC): 242-243. 2003.